

Vadim Ermolayev  
Heinrich C. Mayr  
Mykola Nikitchenko  
Aleksander Spivakovsky  
Grygoriy Zholtkevych (Eds.)

Communications in Computer and Information Science

347

# ICT in Education, Research, and Industrial Applications

8th International Conference, ICTERI 2012  
Kherson, Ukraine, June 2012  
Revised Selected Papers

Editorial Board

Simone Diniz Junqueira Barbosa

*Pontifical Catholic University of Rio de Janeiro (PUC-Rio),  
Rio de Janeiro, Brazil*

Phoebe Chen

*La Trobe University, Melbourne, Australia*

Alfredo Cuzzocrea

*ICAR-CNR and University of Calabria, Italy*

Xiaoyong Du

*Renmin University of China, Beijing, China*

Joaquim Filipe

*Polytechnic Institute of Setúbal, Portugal*

Orhun Kara

*TÜBİTAK BİLGEM and Middle East Technical University, Turkey*

Tai-hoon Kim

*Konkuk University, Chung-ju, Chungbuk, Korea*

Igor Kotenko

*St. Petersburg Institute for Informatics and Automation  
of the Russian Academy of Sciences, Russia*

Dominik Ślęzak

*University of Warsaw and Infobright, Poland*

Xiaokang Yang

*Shanghai Jiao Tong University, China*

Vadim Ermolayev  
Heinrich C. Mayr  
Mykola Nikitchenko  
Aleksander Spivakovsky  
Grygoriy Zholtkevych (Eds.)

# ICT in Education, Research, and Industrial Applications

8th International Conference, ICTERI 2012  
Kherson, Ukraine, June 6-10, 2012  
Revised Selected Papers



Springer

## Volume Editors

Vadim Ermolayev  
Zaporozhye National University  
Department of Information Technologies  
66, Zhukovskogo Street, 69600 Zaporozhye, Ukraine  
E-mail: vadim@ermolayev.com

Heinrich C. Mayr  
Alpen-Adria-Universität Klagenfurt  
Universitätsstrasse 65, 9020 Klagenfurt, Austria  
E-mail: heinrich.mayr@aau.at

Mykola Nikitchenko  
Taras Shevchenko National University of Kyiv  
Department of Theory and Technology of Programming  
64, Volodymyrska Street, 01033 Kyiv, Ukraine  
E-mail: nikitchenko@unicyb.kiev.ua

Aleksander Spivakovsky  
Kherson State University  
27, 40-rokiv Zhovtnya Street, 73000 Kherson, Ukraine  
E-mail: spivakovsky@ksu.ks.ua

Grygoriy Zholtkevych  
V.N. Karazin Kharkiv National University  
School of Mathematics and Mechanics  
4, Svobody Sqr., 61022 Kharkov, Ukraine  
E-mail: g.zholtkevych@gmail.com

ISSN 1865-0929 e-ISSN 1865-0937  
ISBN 978-3-642-35736-7 e-ISBN 978-3-642-35737-4  
DOI 10.1007/978-3-642-35737-4  
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012954138

CR Subject Classification (1998): D.2, F.3, D.3, C.2, H.4, I.2

© Springer-Verlag Berlin Heidelberg 2013

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

*Typesetting:* Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

# Preface

It is our pleasure to present you the proceedings of ICTERI 2012, the eighth International Conference on Information and Communication Technologies (ICT) in Education, Research, and Industrial Applications: Integration, Harmonization, and Knowledge Transfer. The conference was held in Kherson, Ukraine on June 6–10, 2012. This volume is composed of the revised and substantially extended versions of the best ICTERI 2012 papers. The selection was made by the Steering Committee based on the quality, anticipated reader interest, and coverage of the conference scope.

ICTERI as a conference series is concerned with interrelated topics that are vibrant for both the academic and industrial communities:

- ICT infrastructures, integration, and interoperability
- Machine intelligence, knowledge engineering (KE), and knowledge management (KM) for ICT
- Model-based software system development
- Methodological and didactical aspects of teaching ICT and using ICT in education
- Cooperation between academia and industry in ICT

A look at Google Analytics statistics proves broad and growing professional interest in ICTERI and its topics. Indeed, between the launch of the conference web site in November 2011 and the beginning of the conference in June 2012 we received about 3 500 visits from 75 countries (332 cities). Sixty-two per cent of those visits were by returning visitors. Between the end of the conference in June and the end of September 2012 we received an additional 900 visits, of which approximately 49 per cent were by new visitors. In fact the growing interest of the professional community was the indicator that encouraged us to offer this selection of chapters that best cover the scope of the conference and come with relevant research and development results.

This book begins with an invited contribution presenting the substance of one of ICTERI 2012 invited talks given by Prof. Martin Strecker. The chapter deals with the issues of abstraction and verification of properties in real-time Java programs. The rest of the volume is structured in four topical parts:

- ICT Frameworks, Infrastructures, Integration, and Deployment
- Formal Logic and Knowledge-Based Frameworks
- ICT-Based Systems Modeling, Specification, and Verification
- ICT in Teaching and Learning

Part 1 begins with a chapter presenting a new formal approach to developing information processing systems based on quantum automata and quantum computations. The second chapter reports on recent developments in parallelizing legacy software code using rewriting rules and algebraic models of programs. Finally, the third chapter presents the deployment of ICT that enables managing a university using best practices adopted from business corporate management.

Part 2 of the volume focuses on formal logic and knowledge based systems as an important part of the ICT landscape. It starts with a chapter reporting the development of a business-to-consumer communication framework using several channels. In particular, the chapter presents an approach to cross-channel context-dependent message transformation based on the use of semantic technology. The second chapter is more theoretical and reports recent results in solving satisfiability and validity problems for a particular type of formal logic system dealing with program models. Finally, the third chapter elaborates an existence criterion of global-in-time trajectories of non-deterministic Markovian systems.

Part 3 deals with modeling, specification, and verification of ICT-based systems. It begins with a chapter on a case study that combined verification and model driven engineering in a development process for Java programs. The next chapter elaborates an algorithmic approach for reachability checking. The last chapter in this part describes an approach to verifying UML designs looking at potential cross-diagram inconsistencies.

Part 4 of the volume offers reports on the use of ICT in teaching and learning practices. It is opened by a chapter presenting the educational experiment of using a peer-review approach and respective ICT for increasing motivation and learning quality of computer science students. The second chapter reports on the increase of the professional competencies of those future music teachers who used music art multimedia products in their studies. The third chapter is focused on forming a taxonomical structure of general principles for learning software engineering as a profession. The learning approach is supported by a tool suite that was built based on this taxonomy. The fourth chapter reports on the outcomes of introducing new ICT-intensive didactics in the introductory course on computer literacy for future elementary school teachers.

On the whole ICTERI 2012 attracted 70 submissions. The Conference Program Committee selected the best 34 of those 70. In the second selection phase the Steering Committee chose 18 papers as candidates for the proceedings volume. Out of those 18 we finally accepted the 14 most interesting chapters, which have been substantially revised and extended. This resulted in an acceptance rate of 20 per cent.

This volume would not have appeared without the support of many people. First, we would like to thank the members of our Board of Reviewers for providing timely and thorough reviews, and also for being very cooperative in doing additional review work at short notice. We are very grateful to all the

authors for their continuous commitment and intensive work. Furthermore, we would like to thank all the people who contributed to the success of ICTERI 2012. Without their effort there would have been no substance for this volume. Finally, we would like to acknowledge the support of our technical assistant Olga Tatarintseva, who invested considerable resources in checking our submissions.

September 2012

Vadim Ermolayev  
Heinrich C. Mayr  
Mykola Nikitchenko  
Aleksander Spivakovsky  
Grygoriy Zholtkevych





## Program Committee

Mizal Alobaidi	Tikrit University, Iraq
Costin Badica	University of Craiova, Romania
Tobias Bürger	Capgemini, Germany
Andrey Bulat	Kherson State University, Ukraine
Maxim Davidovsky	Zaporozhye National University, Ukraine
Anatoliy Doroshenko	National Technical University NTU KPI, Ukraine
Louis Feraud	Paul Sabatier University (Toulouse 3), IRIT, France
Jose Manuel Gomez-Perez	Intelligent Software Components (iSOCO) S.A., Spain
Vladimir Gorodetsky	St. Petersburg Institute for Informatics and Automation of the Russian Academy of Science, Russian Federation
Sung-Kook Han	Won Kwang University, South Korea
Mitja Jermol	Jožef Stefan Institute, Slovenia
Jason J. Jung	Yeungnam University, South Korea
Nataliya Keberle	Zaporozhye National University, Ukraine
Ron S. Kenett	KPA Group, Israel
Christian Kop	Alpen-Adria-Universität Klagenfurt, Austria
Hennadiy Kravtsov	Kherson State University, Ukraine
Vladyslav Kruglyk	Kherson State University, Ukraine
Mikhail Lvov	Kherson State University, Ukraine
Mihhail Matskin	Royal Institute of Technology (KTH), Sweden
Natalia Morse	National University of Life and Environmental Sciences, Ukraine
Julia Neidhardt	Vienna University of Technology, Austria
Andriy Nikolov	Knowledge Media Institute, The Open University, UK
Aljosa Pasic	ATOS Origin, Spain
Vladimir Peschanenko	Kherson State University, Ukraine
Sergey Rakov	Ukrainian Center for Education Quality Assessment, Ukraine
Kyryl Rukkas	V.N. Karazin Kharkiv National University, Ukraine
Abdel-Badeeh Salem	Ain Shams University Abbasia, Egypt
Wolfgang Schreiner	Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Austria
A.V. Senthil Kumar	Hindustan College of Arts and Science, India

Vladimir A. Shekhovtsov	Alpen-Adria-Universität Klagenfurt, Austria
Oleksandr Sokolov	National Aerospace University “Kharkiv Aviation Institute”, Ukraine
Marcus Spies	Ludwig-Maximilians-Universität München, Germany
Martin Strecker	Paul Sabatier University (Toulouse 3), IRIT, France
Vagan Terziyan	University of Jyväskylä, Finland
Marcel Tilly	European Microsoft Innovation Center, Germany
Nikolay Tkachuk	National Technical University “Kharkiv Polytechnic Institute”, Ukraine
Yuriy Tryus	Cherkasy State Technological University, Ukraine
Mikhail Ugrumov	National Aerospace University “Kharkiv Aviation Institute”, Ukraine
Helmut Veith	Vienna University of Technology, Austria
Maryna Vladimirova	V.N. Karazin Kharkiv National University, Ukraine
Paul Warren	British Telecom, UK
Irina Zaretskaya	V.N. Karazin Kharkiv National University, Ukraine

## Board of Reviewers

Costin Badica	University of Craiova, Romania
Tobias Bürger	PAYBACK GmbH, Germany
Anatoliy Doroshenko	National Technical University NTU KPI, Ukraine
Vadim Ermolayev	Zaporozhye National University, Ukraine
Jason J. Jung	Yeungnam University, South Korea
Natalya Keberle	Zaporozhye National University, Ukraine
Christian Kop	Alpen-Adria-Universität Klagenfurt, Austria
Hennadiy Kravtsov	Kherson State University, Ukraine
Mykola Nikitchenko	Taras Shevchenko National University of Kyiv, Ukraine
Aleksander Spivakovsky	Kherson State University, Ukraine
Martin Strecker	Paul Sabatier University (Toulouse 3), IRIT, France
Paul Warren	Knowledge Media Institute, The Open University, UK
Grygoriy Zholtkevych	V.N. Karazin Kharkiv National University, Ukraine

## **Additional Reviewers**

Maria Del Carmen Calatrava  
Moreno

Vienna University of Technology, Austria

## **ICTERI 2012 Sponsors**

Kherson State University

[www.ksu.ks.ua](http://www.ksu.ks.ua)

DataArt

[www.dataart.com](http://www.dataart.com)

Volkswagen Center Kherson

[volkswagen.ks.ua](http://volkswagen.ks.ua)

# Table of Contents

## Invited Contribution

Abstraction and Verification of Properties of a Real-Time Java . . . . .	1
<i>Nadezhda Baklanova and Martin Strecker</i>	

## ICT Frameworks, Infrastructures, Integration, and Deployment

Abstract Quantum Automata as Formal Models of Quantum Information Processing Systems . . . . .	19
<i>Mizal Alobaidi, Andriy Batyiv, and Grygoriy Zholtkevych</i>	
Parallelizing Legacy Fortran Programs Using Rewriting Rules Technique and Algebraic Program Models . . . . .	39
<i>Anatoliy Doroshenko and Kostiantyn Zhereb</i>	
University as a Corporation Which Serves Educational Interests . . . . .	60
<i>Alexander Spivakovsky, Lyudmila Alferova, and Eugene Alferov</i>	

## Formal Logic and Knowledge-Based Frameworks

A Multi-channel Communication Framework . . . . .	72
<i>Michal Nagy</i>	
Satisfiability and Validity Problems in Many-Sorted Composition-Nominative Pure Predicate Logics . . . . .	89
<i>Mykola S. Nikitchenko and Valentyn G. Tymofiev</i>	
A Criterion for Existence of Global-in-Time Trajectories of Non-deterministic Markovian Systems . . . . .	111
<i>Ievgen Ivanov</i>	

## ICT-Based Systems Modeling, Specification, and Verification

Combining Verification and MDE Illustrated by a Formal Java Development . . . . .	131
<i>Selma Djedjai, Mohamed Mezghiche, and Martin Strecker</i>	
About One Efficient Algorithm for Reachability Checking in Modeling and Its Implementation . . . . .	149
<i>Alexander Letichevsky, Olexander Letychevskyi, and Vladimir Peschanenko</i>	

Cross-Diagram UML Design Verification ..... 165  
*Iryna Zaretska, Oleksandra Kulankhina, and Hlib Mykhailenko*

**ICT in Teaching and Learning**

Coursework Peer Reviews Increase Students' Motivation and Quality  
of Learning..... 177  
*Vadim Ermolayev, Natalya Keberle, and Sergey Borue*

Influence of Music Art Multimedia Production on Professional  
Competence of the Future Music Teachers ..... 195  
*Lyudmila Gavrilova*

General Disciplines and Tools for E-Learning Software Engineering ..... 212  
*Ekaterina Lavrisheva and Alexei Ostrowski*

Formation of Digital Competence of Future Teachers of Elementary  
School ..... 230  
*Nataliya Kushnir and Anna Manzhula*

**Author Index** ..... 245

# Abstraction and Verification of Properties of a Real-Time Java

Nadezhda Baklanova and Martin Strecker

IRIT (Institut de Recherche en Informatique de Toulouse)  
Université de Toulouse\*  
118 route de Narbonne, F-31062 Toulouse CEDEX 9, France  
{nadezhda.baklanova,martin.strecker}@irit.fr

**Abstract.** We present a tool for analysing resource sharing conflicts in multithreaded Java programs. Java programs are translated to timed automata models verified afterwards by the UPPAAL model checker. Analysed programs are annotated with timing information indicating the execution duration of a particular statement. Based on the timing information, the analysis of execution paths is performed, which gives an answer whether resource sharing conflicts are possible in a multithreaded Java program. If the analysis succeeds, resource locks may be eliminated from the Java program.

**Keywords:** timed automaton, Java, multithreading, deadlock, resource sharing conflict, Uppaal.

## 1 Introduction

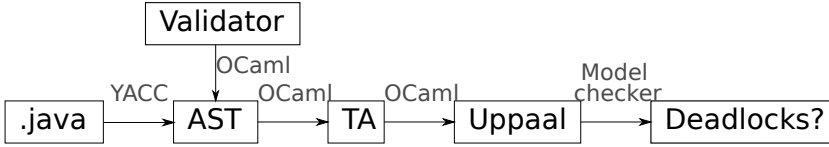
Parallel computations quickly develop nowadays, and the problem of effective debugging multithreaded programs arises. It is known to be a very difficult problem for a software developer, and thorough testing cannot discover all the fatal errors in a program due to unpredictability of execution. One type of errors are resource sharing conflicts. In order to avoid them, one may want to guarantee that the same resource is not accessed by different threads at the same moment of time. If one concentrates on this aspect, the behavior of a program may be naturally modelled by timed automata, and then one may find error-prone places in the program using a timed automata model checker.

In order to achieve this goal, we need to enrich the Java language with annotations indicating time information. The annotations show how much time is required for executing a statement and, consequently, how much time is required for a thread to have an exclusive access to a resource. It allows to avoid usage of **synchronized** statements in programs after verifying that no resource is used simultaneously by two or more threads. It improves predictability of execution and guarantees there will be no delays due to locking conflicts. Based on an annotated Java program, a timed automaton is generated taking into account the

---

\* Part of this research has been supported by the project *Verisync* (ANR-10-BLAN-0310).

time required for execution of statements. Finally, we check the generated automaton for possible resource sharing conflicts using the UPPAAL model checker in the generated automaton. The transformation sequence is shown in the diagram [1](#) below.



**Fig. 1.** Java program verification process

The overall aim is thus to replace a lock-driven protocol for resource conflict avoidance by a time-driven approach. If a check on the abstract level of timed automata indicates no resource access conflict, then also the underlying Java program can be expected to run without conflicting access to resources. In this case, locking even becomes superfluous. If, however, the check fails, nothing can be said about the behaviour of the Java program when executed, just like for an ill-typed program.

The purpose of the present chapter is to sketch the overall approach and define the correspondence between Java and timed automata, without giving a proof of the soundness of the abstraction, which remains for future work. A preliminary version of this chapter has appeared in [1](#).

## 1.1 Related Work

There are several tools for scheduling analysis of real-time tasks. Verification of scheduling strategies with timed automata is considered in [2](#). However, it operates with a high level notion of abstract tasks and does not look inside the source code. The authors perform schedulability analysis with the fixed-priority scheduling strategy by translating a system to be verified to a timed automaton and verifying it afterwards with the TIMES tool.

Another approach is used in [3](#)[4](#). Here SCJ source code analysis is performed using timed automata. An automaton is generated from the source code, and every statement is mapped to a certain part of the automaton. The timing model is based on WCET computation and predefined periods of tasks. The translation procedure described in [3](#) contains an inconsistency between Java semantics and model semantics of the generated system. Locking mechanism is implemented in UPPAAL model as monitors which are incremented when a lock is acquired and decreased when a lock is released. However, there are no checks before acquiring the lock in the model. It makes the situation when two threads have locked the same resource at the same time possible, but it does not correspond to the JVM behavior. In order to manage this problem we suggest

to define two semantics of Java execution. One treats locks in Java manner, i.e. checks the resource monitor before acquiring the lock and does not allow double locking made by different threads. Another one just stores the number of times a resource lock has been acquired but does not check whether a resource has already been locked. These two semantics are equivalent for programs without resource sharing conflicts therefore the programs verified by our tool are correct during execution on JVM.

An approach in the opposite direction is described in [5]. The authors generate RTSJ code from a Uppaal model. Uppaal model of a system is supposed to be verified, and if the generation procedure is assumed to be correct, the output is a verified RTSJ program.

A translation from SystemC to UPPAAL is presented in [6]. One of the purposes of this work is to give a formal semantics to the (only informally defined) SystemC language. The differences between SystemC and Java, as far as the translation to UPPAAL is concerned, still has to be explored.

In [7] a schedulability analysis of a set of tasks is performed by exhaustive search combined with UPPAAL for determining when the search is complete. Again, the internal structure of tasks is not taken into account which makes impossible to do conclusions about thread interactions. The authors listed the limitations they had encountered: lack of memory and lack of UPPAAL integer range.

The paper [8] contains schedulability analysis of multithreaded SCJ (Safety Critical Java) programs and takes resource sharing into account. Resources are considered to be locked during the whole execution of a task. Analysis is performed by UPPAAL modeling taking into account the resource locks. This model is not fine-grained, and the negative result may not be relevant in cases when developers try to minimize the length of critical sections.

A tool for automatic verifying the determinism of Java programs is described in [9]. In particular, parallel Java programs are checked for absence of race conditions. It allows not to use Java `synchronized` statements. The tool does not use any external checkers, the verification uses the internal abstract representation of a Java program.

A theoretical approach to managing resources in parallel programs is suggested in [10]. It is based on an enhanced version of rely-guarantee reasoning and allows to verify memory safety and of parallel programs.

## 2 Preliminaries

### 2.1 Real-Time Java

Specification of the Real-Time profile for Java was developed in the first half of the 2000s and aimed at making work with Java threads predictable and suitable for real-time applications. The specification addressed defining an explicit scheduler and scheduling strategies, advanced memory management and raw memory access, resource locking taking thread priorities into account, refined notion of time, several additions for threads, asynchronous event handling etc. [11]. The



real-time specification introduced thread priorities, thread deadlines and explicit notion of scheduler which did not exist in usual Java.

In the middle of the 2000s the work on the specification of Safety-Critical Java was started. The main goal was to allow the SCJ applications to be highly reliable [12]. The SCJ specification defines more strictly the subset of possible programs and pays a lot of attention to the VM performance requirements [13]. Until now, there is no reference implementation of SCJ machine, however, there are projects such as the Open SCJ project [1] aiming at an open-source implementation of SCJ virtual machine.

## 2.2 Uppaal

UPPAAL is a tool for modeling timed automata and verifying their properties. A timed automaton [14] is a Büchi automaton enhanced with clocks. States and edges may have Boolean constraints on clocks called invariants and guards respectively; edges may also have abstract “actions”. There are two kinds of possible transitions: delay transition and action transition. During a delay transition an automaton stays in the same state, and time advances. During an action transition an automaton takes an edge and changes the state; some clocks may be reset to 0. A timed automaton is allowed to stay in a certain state as long as its invariant is true, and an automaton may take a particular edge, if its guard is true in the current moment of time.

In Uppaal “actions” are concrete arithmetic actions with variables or arrays whose values are preserved between TA states. Variables and array elements may also be used in edge constraints.

Properties to be verified in UPPAAL are to be expressed in a subset of TCTL logic allowing a single path quantifier directly followed by a U operator [15].

## 3 Sample Usage

### 3.1 Input Program

Before describing our approach more in detail, we illustrate it here with a small example. The outermost class containing the `main` method is called `Threads`. Two threads `t1`, `t2` are declared in the `main` method. `Run1` and `Run2` are nested classes inside the `Threads` class implementing the `Runnable` interface.

```
Threads ts;
Run1 r1;
Run2 r2;
Thread t1,t2;
ts=new Threads();
r1=ts.new Run1();
r2=ts.new Run2();
ts.res=new Res();
```

---

<sup>1</sup> <http://www.ovmj.net/oscj/>

```
t1=new Thread(r1,"t1");
t2=new Thread(r2,"t2");
```

Methods called on the thread start are the following:

```
private class Run1 implements Runnable{
public void run(){
    int value,i;
    //@ 1 @//
    i=0;
    while(i<10){
        synchronized(res){
            //@ 2 @//
            value=Calendar.getInstance().get(Calendar.MILLISECOND);
            //@ 5 @//
            res.set(value);
        }
        try{

            Thread.sleep(10);
        }
        catch(InterruptedException e){
            System.out.println(e.getMessage());
        }
        //@ 2 @//
        i++;
    }
}
}
```

```
private class Run2 implements Runnable{
public void run(){
    int value,i;
    //@ 1 @//
    i=0;
    try{

        Thread.sleep(9);
    }
    catch(InterruptedException e){
        System.out.println(e.getMessage());
    }
    while(i<10){
        synchronized(res){
            //@ 4 @//
            value=res.get();
        }

        try{
            Thread.sleep(8);
        }
    }
}
```

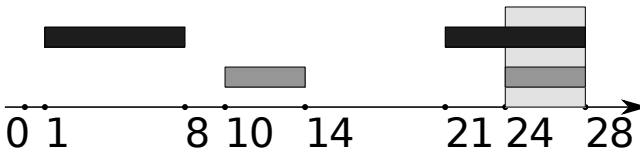
```

    catch(InterruptedException e){
        System.out.println(e.getMessage());
    }
    //@ 2 @//
    i++;
}
}
}

```

Here `res` is a resource declared in the main class. Calls of the `res.get()` and `res.set()` methods are “actions” using the locked resources which are preceded by timing annotations showing the amount of time the “action” requires. During the translation they are considered to be abstract statements inside the locked region taking  $n$  time units for execution.

Both threads do some “actions” requiring an exclusive lock with the resource and then sleep for some time. The `synchronized` statements are a potential source of resource sharing conflict if both threads wake up simultaneously. One can see the resource access conflict in the execution timeline [2](#) showing times when the threads demand an exclusive lock for the resource.



**Fig. 2.** Execution timeline.  $t_1$  is dark gray,  $t_2$  is gray. Conflict between 24 and 28 is shown in light gray.

`res` has type `Res` which is a simple class allowing to read and write to one field.

```

class Res{
    private int i;
    public void set(int j){
        i=j;
    }
    public int get(){
        return i;
    }
}
}

```

### 3.2 Generated System

On the basis of the annotated program, our framework generates the following model which is passed to the model checker UPPAAL for verification.

The generated timed automata are shown in Figure [3](#).

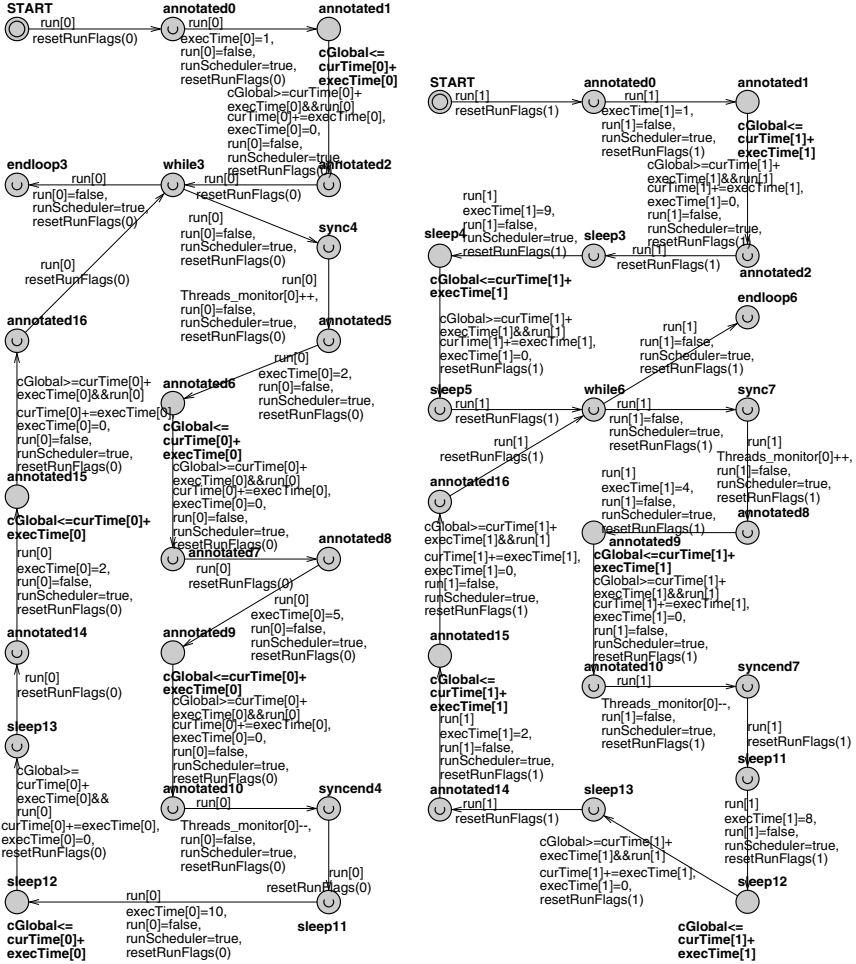


Fig. 3. Generated automata for threads  $t_1$  and  $t_2$

The generated checking function is

```
int check_Threads(int m[1]){
    return forall(i: int[0,0]) m[i] <= 1;
}
```

The generated formula for model checking is

$$A[]check\_Threads(Threads\_monitor).$$

When performing verification with UPPAAL, this formula is evaluated to false, and the generated trace for counterexample stops in states *aut\_Run1.t1.annotated9* and *aut\_Run2.t2.annotated8* during the second loop iteration at the time moment 24.

## 4 Translation from Java to Abstract Syntax Tree

### 4.1 Source Language

Considering the idea of “extended Java”, a possibility to write annotations for every Java statement is added to Java syntax. These annotations contain time required for executing the whole block or statement next of the annotation. The annotations have syntax of Java comments, therefore, the annotated programs can be compiled to bytecode by usual Java compilers. We assume that a developer has information about execution time of particular statements. The time in the samples is in abstract time units but one can annotate a program with real values in microseconds based on computer architecture, compiler version, running software etc. Annotations may either contain an exact execution time or an interval in which the execution time lies.

Unfortunately, standard Java annotations cannot be added to arbitrary statements. Even the latest extension of Java annotations implemented in JDK 7 [\[12\]](#) does not allow to annotate executable statements (assignments, loops, conditions etc.) which are of the most interest for us. For this reason we used self-written parser of the “extended Java” language. The parser is written with OcamlYACC and recognizes Java with several restrictions. The parser produces an AST from Java code as a set of OCaml objects.

Java programs consist of a number of classes containing methods. Classes may have fields for storing object information. We concentrate on a subset of Java containing the most important syntax constructions. Target Java programs may contain the following statements:

```

type stmt =
  Skip
    (* empty statement *)
  | Expr of expr
    (* expression statement *)
  | Assign of var * expr
    (* assignment statement: a=5+4; *)
  | Seq of stmt * stmt
    (* sequence of two statements: a=4; b=5; *)
  | Cond of expr * stmt * stmt
    (* conditional statement: if(a=1) {...} else {...} *)
  | While of expr * stmt
    (* loop statement: while(a<5){...} *)
  | Return of expr
    (* value return: return a; *)
  | AnnotStmt of annot * stmt
    (* annotated statement: //@ 4 @// a=3; *)
  | SyncStmt of expr * stmt
    (* synchronized statement: synchronized(a){...} *)

```

This is the representation of Java statements in AST written as OCaml type. Internally, statements and expressions have the same type, however, there is a difference on the semantics level. Expressions are supposed not to produce side effects whereas statements can induce state changes.

Cast operators are not supported for now. Since we perform static analysis, dynamic features are excluded, namely, arrays or references to `this` instead of specifying an object name explicitly. `try...catch` constructions can be parsed, however, code in the `catch` block is not translated, i.e. `try block1 catch block2` is considered to be equivalent to `block1`.

The analysed programs must have a proper structure which would guarantee the correctness of the generated model. In order to make the AST generator simpler, all used packages are supposed to be imported in the header of a program. Local variables must be declared in the beginning of methods before statements, and declaration statements cannot be combined with assignments. This assumption helps to avoid problems with scope of the variables declared in the middle of a method. It is ensured in the parsing step.

We have developed a number of checking functions for testing the program correspondence to the structure requirements. These functions work after parsing on the semantics checking step. They return a Boolean value showing whether a check was successful. The checking functions traverse a method body recursively performing checks of the interesting cases. Currently, there are the following checks:

- **checkAliases**. The objects which can be accessed by different threads should not have more than one name, i.e. the correct program should not introduce aliases for them. If there are aliases in a program, there is no easy way of determining whether two variables point to the same object. This leads to inability to know which objects are locked at a particular moment of time.
- **checkAnnotCoverage**. The whole AST except the main method must be annotated with timing information. Each leaf or one of its parents must have an annotation in order to avoid undetermined execution duration. Method calls currently are not translated. However, since method calls are always leaves in the AST, we can use timing information instead of looking inside the method structure. The only exception must be a call of the `Thread.sleep` method since it takes time for execution but does not load CPU. Currently, we do not support `wait/notify` statements.
- **checkSyncArgument**. Argument of the `synchronized` statement is assumed to be an explicit object name, not an expression.
- **checkNestedSyncs**. Reentrant locks are not allowed, i.e. when the same thread acquires lock of the same object several times.
- **checkAnnotSync**. Annotated statements cannot contain `synchronized` statements since the automata generator treats annotated statements as atomic entities. If an annotated statement contains `synchronized` block, and during runtime there is a conflict between threads for the locked resource, our model cannot catch this conflict.
- **checkMainMethodPosition**. Program `main` method must be in the first class in order to make the generator simpler.

- `checkThreadConstructors`. Threads are supposed to be declared in the `main` method which is an entry point of the program and must be declared in the first class of a program. The `main` method cannot contain any code except thread declarations, initializations and calls for starting the threads. Threads are assumed to be created with the constructor

```
Thread(Runnable target, String name),
```

so the name of the object containing the thread logic is explicitly specified. `Runnable` object should implement `Runnable` interface or extend `Thread` class and override `run` method.

The `checkAliases` function first builds a list `initObjects` of the objects which can be accessed by different threads and then searches for assignments to these objects other than initializations. If an object is assigned a `new ...` expression, check for this assignment succeeds; if the expression which is assigned contains anything else except a constructor call, check for this assignment fails.

```
let rec checkAliasesInExpr initObjects=function
  | Assign(CallObject(o),e)->
    if mem o sharedFields then
      (match e with
       | CallMethod(New c,f,ps)->true
       | Null->true
       | _->false)
    else true
```

For the other cases the function looks inside statement bodies. Below there are semi-formal rules for these cases.

$$\frac{\text{check } e_1 \quad \text{check } e_2}{\text{check } (\text{Seq } e_1 \ e_2)} \quad \frac{\text{check } e_1 \quad \text{check } e_2}{\text{check } (\text{Cond } c \ e_1 \ e_2)} \quad \frac{\text{check } e}{\text{check } (\text{While } c \ e)}$$

$$\frac{\text{check } e}{\text{check } (\text{Annot } a \ e)} \quad \frac{\text{check } e}{\text{check } (\text{Sync } obj \ e)}$$

The rest of checking functions are evident, and we do not show details of their implementation here.

## 5 Translation from Abstract Syntax Tree to Timed Automaton

### 5.1 Model of Java Program Execution

Multithreaded Java programs have a scheduler which selects a thread to be executed in the next moment of time. It non-deterministically selects a thread from those eligible for execution, and it can suddenly stop thread's execution and start executing another thread. Usual Java schedulers do not support thread priorities or task deadlines.

We model a Java scheduler as a separate automaton with three states:

- *waitScheduling*, where a scheduler waits for some time before starting the next scheduling cycle,
- *updateStatus*, where eligibility status of all threads is updated,
- *runThread*, where the scheduler gives control to any eligible thread.

If no thread can be scheduled, the scheduler returns to *waitScheduling* and waits for some time. Then it tries to schedule some thread again. When scheduled, a thread executes an atomic action and returns control back to the scheduler. The scheduler returns to the state *updateStatus* and updates thread eligibility flags.

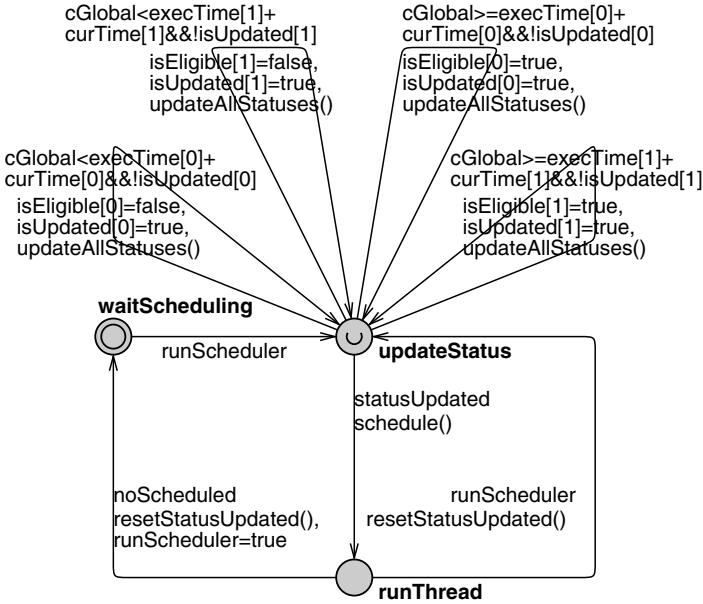


Fig. 4. Model of the scheduler for two threads

Here,  $execTime[i]$  is an array containing the execution time of the next instruction for all threads taken from annotation values.  $cGlobal$  is a global clock.  $runScheduler$  is a Boolean flag indicating that it is scheduler's turn to proceed,  $statusUpdated$  shows that statuses of all threads were updated,  $isScheduled[i]$  is an array indicating whether a particular thread is scheduled for execution.  $run[i]$  is a global Boolean array showing that a thread with number  $i$  may proceed.

## 5.2 Semantics of Annotated Statements

Suppose we have an annotated statement

```
//@ 5 @//
a = b - 4;
```



which claims that the statement  $a = b - 4$ ; takes 5 time units for execution. This time is considered as exact execution time, i.e. the exact amount of time when the thread executing this statement loads CPU. We consider the time when an annotation expires as a hard deadline. If a program does miss this deadline, the situation is critically incorrect, and the program cannot be verified because of incorrect annotations.

Formally, a small-step semantics of annotated statements may be written as following:

$$\frac{\frac{G \vdash (e, s) \xrightarrow{\delta} (e', s') \quad t - \delta \geq 0}{G \vdash (\text{Annot } t \ e; \ s) \xrightarrow{\delta} (\text{Annot } (t - \delta) \ e', \ s')}}{t \geq 0}}{G \vdash (\text{Annot } t \ (\text{Val } v), \ s) \xrightarrow{t} (\text{Val } v, \ s)}$$

Here  $G$  is a generated system, and the relation  $G \vdash (e, s) \xrightarrow{t} (e', s')$  means there is a possible reduction of an expression  $e$  to an expression  $e'$  which takes time  $t$  and changes state from  $s$  to  $s'$ .

These rules should be read like

- if an expression  $e$  in the body of an annotated statement can be reduced to an expression  $e'$ , and the system state is changed from  $s$  to  $s'$ , and the reduction takes  $\delta$  time units, and the deadline specified in the annotated statement is not missed, then we may reduce the initial annotated statement to the new one with the new body,  $e'$ , being in the state  $s'$ , and the new deadline,  $(t - \delta)$ .
- or, if an expression in the body of annotated statement has already been reduced up to the end, i.e. to a single value  $\text{Val } v$ , and the deadline specified in the annotation is not missed, we may reduce the annotated statement to the value of its body staying in the same state, and this reduction would take the rest of time specified in the annotation.

We assume that each thread is executed on its own processor i.e. the execution is purely parallel. Eligible threads do not wait until other threads free the processor. As soon as a thread becomes eligible for execution it starts executing.

### 5.3 Automata Generation

The generator translates each thread of a program to a separate automaton. At the beginning it creates a set of OCaml objects representing a timed automaton, and after that the timed automaton is printed in the format recognizable by UPPAAL, which performs model checking.

The Ocaml type for an automaton looks like

```
type ta = Empty
  | TA of (node list) * (urgent list) *
        (committed list) * (edge list) * start *
        final
```

Here **start** and **final** are start and final states of the timed automaton. The final state is required because a timed automaton is generated recursively, and it is necessary to determine where the previously generated parts finish, although there is no such a notion in the definition of timed automata. Committed and urgent are state characteristics specific for UPPAAL, however, they can be modeled by a standard timed automaton, i.e. they do not increase the expressiveness of the traditional TA model. Final states of the generated timed automata are always urgent, that means, the automata are not allowed to rest in these states for any time. One may find definitions related to timed automata in [15].

Since method calls are not translated, only **run** methods of **Runnable** objects are translated to timed automata because they are the only methods which can contain executable code. Each thread declared and initialized in the **main** method is mapped to a separate automaton (template in the UPPAAL terminology). The system has one global clock and a global array of object monitors.

An object monitor is an integer variable which is incremented when this object is locked and decremented when the lock is released. In UPPAAL model monitors are implemented as an array of integers, each object is encoded as an array item; consistency of indices is guaranteed by the automata generator.

All statements except **Thread.sleep** and the annotated ones are assumed not to take any time for execution; for this reason all the states without timing information are made urgent in Uppaal model. Time is not allowed to pass when an automaton is in urgent state.

Statements annotated with timing information are treated as a “black box” and are supposed contain **synchronized** blocks. Otherwise, a possible situation is when a thread tries to access an object field which is locked by another thread. In this situation JVM keeps the thread waiting until the lock is released, however, our translation does not notice this delay and produces an incorrect automaton.

The generated system has one global clock, **cGlobal** and several auxiliary variables. There are Boolean flags for each automaton, **run** and **runScheduler**, which are set to true if this automaton may advance in the current time moment. An integer array **curTime** represents the time when an automaton entered the state corresponding to an annotation statement. Another integer array, **execTime**, stores the duration of the currently executed statement. Finally, an integer array **<class\_name>\_monitor** stores the number of object locks for each shared object.

Annotations in Java programs contain relative time but timed automata use global time, therefore we need to keep track of how much time has passed since a program has been started. The only statements allowing time to increase are annotated statements and calls of **Thread.sleep**. Values for **execTime** are taken from timing annotations or method argument. Suppose  $t$  was the global time when an automaton entered a state corresponding to an annotated statement. When it leaves this state, model time and **curTime** variable are increased by real execution time, **execTime**. **curTime** values may be different in different automata but the global time is always equal to **curTime** when its corresponding automaton is executing.

Basic items for building timed automata are statements: each statement is translated into a part of timed automaton.

Translation from AST to timed automata skips field and variable declarations because they do not change the state of a program. At the same time, all the objects declared in the main program class get a monitor.

Boolean conditions inside `while` and `if` statements are not translated. It is assumed that any of the two possible ways can be taken during runtime.

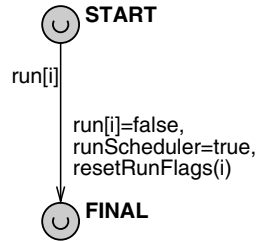
`Skip` and `Return` statements are mapped to an empty automaton because they do not influence the state of a program.

The rules for mapping other AST statements to the parts of a timed automaton are listed in the table below.

**Table 1.** Translation rules

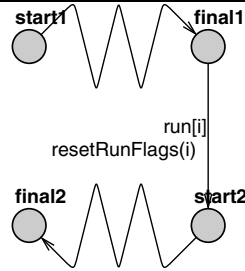
---

**Assign(v,e):** add two urgent states: *ASSIGNMENT1* and *ASSIGNMENT2*, which are start and final, and a transition with a guard and an update between them. The guard checks `run[i]`, i.e. whether this automaton is allowed to proceed in the current moment of time. The update sets `run[i]` to false, `runScheduler` to true and other `run[j ≠ i]` to false.



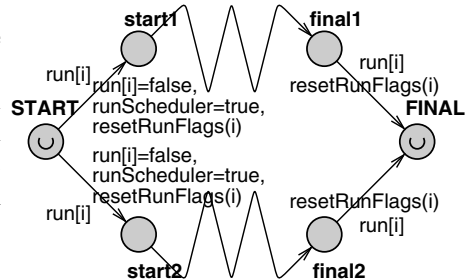

---

**Seq(c1,c2):** suppose *a1* and *a2* are the automata for *c1* and *c2* respectively, add an edge from *final1* to *start2*, *start1* is the start state, *final2* is the final state. The edge has a guard checking `run[i]` and an update setting `run[j ≠ i]` to false.

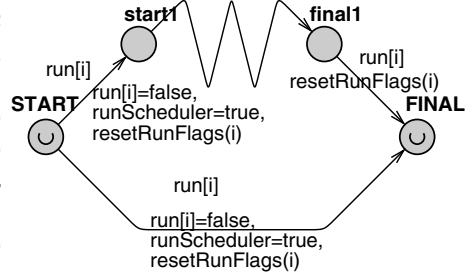



---

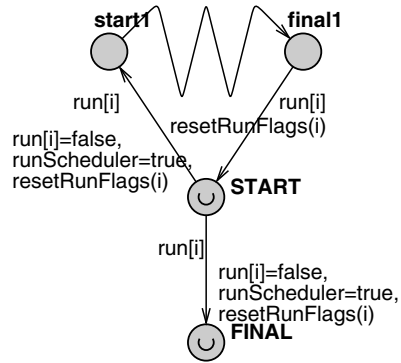
**Cond(e,c1,c2):** suppose *a1* and *a2* are the automata for *c1* and *c2* respectively, add two urgent states *START* and *FINAL*, which are the start and final states of the new automaton, and edges from *START* to *start1* and *start2*, from *final1* and *final2* to *FINAL*. The edges from *START* to *start1*, *start2* have guards checking `run[i]` and updates resetting `run[i]` to false,



`runScheduler` to true and `run[j ≠ i]` to false. The edges from `final1` and `final2` have only guards checking `run[i]` and updates setting `run[j ≠ i]` to false. If one of the branches is absent, e.g. there is no else branch, a transition from `START` to `FINAL` is added. This transition has a guard checking `run[i]` and updates resetting `run[i]` to false, `runScheduler` to true and `run[j ≠ i]` to false.

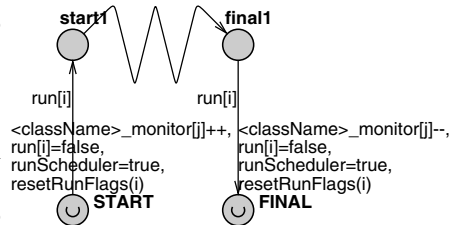


**Loop(e,c1):** suppose *a1* is the automaton for *c1*, add two urgent states *START* and *FINAL*, which are the start and final states of the new automaton, and edges from *START* to *start1*, from *final1* to *FINAL* and from *START* to *FINAL*. The edges from *START* to *start1* and from *final1* to *START* have guards checking `run[i]` and updates resetting `run[i]` to false, `runScheduler` to true and `run[j ≠ i]` to false. The edge from *START* to *FINAL* has only a guard checking `run[i]` and an update setting `run[j ≠ i]` to false.



**Expr(e):** not translated except methods for thread management. For translation of `Thread.sleep` see the `Annot` item.

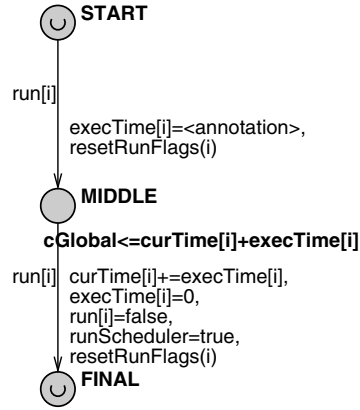
**Sync(e,c1):** suppose *a1* is the automaton for *c1*, add two urgent states *START* and *FINAL*, which are the start and final states of the new automaton, and edges from *START* to *start1*, from *final1* to *FINAL*. We assume that expression *e* is a field declared in the outermost class. Its monitor is incremented when the edge from *START* to *start1* is taken and decremented when the edge from *final1* to *FINAL* is taken. Also, both edges have guards checking `run[i]` and updates resetting `run[i]` to false, `runScheduler` to true and `run[j ≠ i]` to false.



---

`Annot(a, c1)`, `Thread.sleep(a)`: add three states: *START*, *MIDDLE* and *FINAL*, and edges from *START* to *MIDDLE*, and from *MIDDLE* to *FINAL*. *START* and *FINAL* are the start and final states of the new automaton, both are urgent. The edge from *START* to *MIDDLE* has a guard checking `run[i]`, an update setting `execTime` to the execution time indicated in the annotation and another update setting `run[j ≠ i]` to false. The edge from *MIDDLE* to *FINAL* has a guard checking `run[i]` and several updates. First, there is an update increasing `curTime` to `execTime`. Second, there is an update resetting `execTime` to zero. Third, there are updates setting `run[i]` to false, `runScheduler` to true and `run[j ≠ i]` to false.

---



## 5.4 Model Checking

Our initial goal was to check whether there are possible resource sharing conflicts during program execution. UPPAAL provides an ability to check properties of timed automata expressed with TCTL formulas [16]. The basics of TCTL and its applications are described in [17]. Together with automata code our generator produces a file with properties to check. The negative property of the generated system is whether there are two threads accessing the same resource at the same time. We check the positive variant of it. There is a function `check_<class_name>` checking that all the elements of the monitors array are less or equal to 1. If this property holds for all states of all possible paths, the system does not have resource sharing conflicts. In UPPAAL syntax the property looks like

$$A[]check\_(<className>)((<className>)\_monitor).$$

If the property does not hold, UPPAAL produces a trace violating the check.

## 6 Conclusions

We presented the very first steps of an approach for generating timed automata from Java programs. The Java language is extended with timing annotations, which makes possible to check resource sharing conflicts and deadlocks in a generated system. We expect that replacing a lock-controlled resource access policy

by a time-driven approach allows for better temporal and functional predictability, while allowing for greater flexibility than, say, synchronous languages.

The approach has been implemented in a prototype tool, and first tests allow to assume that this approach works. However, the number of states increases rapidly with the growth of program size. That makes this approach difficult to apply for large systems. In order to avoid state explosion, large parts of code should be included into annotated statements. It allows to abstract from particular statements and generate an automaton with quite a few states.

### 6.1 Interval Annotations

We have considered an approach to make our analysis more precise. Currently, an annotation is exact time required for execution of a statement. Certainly, it is not a realistic model as one can never know before execution itself how much time it will take. The real execution time depends on the contents of the processor cache, also on the compiler optimisation level and many other things. For this reason we considered a simple model where timing annotation is an interval, and execution time must lie within it. However, this naive model cannot represent the execution flow correctly; consider an example when a loop body has an interval annotation.

```
while ( true ) {
  //@ 3 - 5 @//
  ... //some actions
}
```

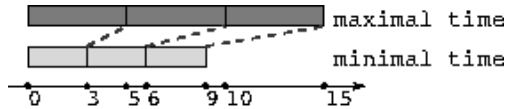


Fig. 5. Execution timeline for intervals

On the execution timeline to the right one can see that after the third round of loop execution the interval of non-determinism became longer than the loop execution time itself. The better model is still a question for further investigation; one of the possible examples is the model discussed in [18]. The approach suggested by the authors is to keep non-determinism for separate steps of execution, however, sets of instructions have a hard deadline. This may help to solve the mentioned unlimited growth between best and worst execution time.

### 6.2 Future Work

Further work may be performed in two directions: Firstly, more Java source code statements and thread-specific methods should be translated to timed automata. Secondly, the adequacy of the translation algorithm is expected to be verified with a proof assistant, based on a formal semantics of Real-Time Java. The final aim of the future work is to support the constructions of Real-Time Java and have a formally verified translation procedure.

## References

1. Baklanova, N., Strecker, M., Féraud, L.: Resource sharing conflicts checking in multithreaded Java programs. In: Journées FAC 2012 (2012)
2. Fersman, E., Mokrushin, L., Pettersson, P., Yi, W.: Schedulability analysis of fixed-priority systems using timed automata. *Theor. Comput. Sci.* 354, 301–317 (2006)
3. Bøgholm, T., Kragh-Hansen, H., Olsen, P.: Model based schedulability analysis of real-time systems. Master’s thesis, Aalborg University (2008)
4. Bøgholm, T., Kragh-Hansen, H., Olsen, P., Thomsen, B., Larsen, K.G.: Model-based schedulability analysis of safety critical hard real-time Java programs. In: Bollella, G., Locke, C.D. (eds.) JTRES. ACM International Conference Proceeding Series, vol. 343, pp. 106–114. ACM (2008)
5. Hakimipour, N., Strooper, P., Wellings, A.: A model-based development approach for the verification of real-time java code. *Concurrency and Computation: Practice and Experience* 23(13), 1583–1606 (2011)
6. Herber, P., Pockrandt, M., Glesner, S.: Transforming systemc transaction level models into uppaal timed automata. In: 2011 9th IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE), pp. 161–170 (2011)
7. Cordovilla, M., Boniol, F., Noulard, E., Pagetti, C.: Multiprocessor schedulability analyser. In: Chu, W.C., Wong, W.E., Palakal, M.J., Hung, C.C. (eds.) SAC, pp. 735–741. ACM (2011)
8. Ravn, A.P., Schoeberl, M.: Cyclic executive for safety-critical java on chip-multiprocessors. In: Kalibera, T., Vitek, J. (eds.) JTRES. ACM International Conference Proceeding Series, pp. 63–69. ACM (2010)
9. Vechev, M., Yahav, E., Raman, R., Sarkar, V.: Automatic Verification of Determinism for Structured Parallel Programs. In: Cousot, R., Martel, M. (eds.) SAS 2010. LNCS, vol. 6337, pp. 455–471. Springer, Heidelberg (2010)
10. Tofan, B., Schellhorn, G., Bäuml, S., Reif, W.: Embedding rely-guarantee reasoning in temporal logic. Technical Report 2010-07, Informatik (2010)
11. The Real-Time for Java Expert Group: The Real-Time Specification for Java (2006)
12. The Open Group JSR: JSR-302 Safety Critical Java Technology Specification (2010), <http://jcp.org/en/jsr/detail?id=308>
13. Henties, T., Hunt, J.J., Locke, D., Nilsen, K., Schoeberl, M., Vitek, J.: Java for safety-critical applications. In: 2nd International Workshop on the Certification of Safety-Critical Software Controlled Systems, SafeCert 2009 (March 2009)
14. Alur, R., Dill, D.L.: A theory of timed automata. *Theoretical Computer Science* 126, 183–235 (1994)
15. Bengtsson, J.E., Yi, W.: Timed Automata: Semantics, Algorithms and Tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) ACPN 2003. LNCS, vol. 3098, pp. 87–124. Springer, Heidelberg (2004), [http://dx.doi.org/10.1007/978-3-540-27755-2\\_3](http://dx.doi.org/10.1007/978-3-540-27755-2_3)
16. Alur, R., Courcoubetis, C., Dill, D.: Model-checking for real-time systems. In: Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science, LICS 1990, pp. 414–425 (June 1990)
17. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
18. Henzinger, T.A., Horowitz, B., Kirsch, C.M.: Giotto: a time-triggered language for embedded programming. *Proceedings of the IEEE* 91(1), 84–99 (2003)

# Abstract Quantum Automata as Formal Models of Quantum Information Processing Systems

Mizal Alobaidi<sup>1</sup>, Andriy Batyiv<sup>2</sup>, and Grygoriy Zholtkevych<sup>2</sup>

<sup>1</sup> Tikrit University,  
Faculty of Computer Science and Mathematics, P.O. Box-42, Tikrit, Iraq  
mizalobaidi@yahoo.com

<sup>2</sup> V.N. Karazin Kharkiv National University,  
School of Mathematics and Mechanics, 4, Svobody Sqr., 61022, Kharkiv, Ukraine  
generatorglukoff@gmail.com, zholtkevych@univer.kharkov.ua

**Abstract.** Nowadays, quantum computation is considered as a perspective way to overcome the computational complexity barrier. Development of a quantum programming technology requires to build theoretical background of quantum computing similarly to the classical computability theory and the classical computational complexity theory. The challenge to develop such theoretical background was posed by Yu.I. Manin. An attempt to build a mathematically rigorous model for quantum information processing systems in compliance with the concept of Yu.I. Manin is presented in the chapter. The attempt carries out by identifying elementary constituents of quantum computational processes. They are called quantum actions and their properties are studied in the chapter. In particular, the equivalence criterion of quantum actions in terms of their generating operators has been found; the special class of quantum actions has been characterised in terms of generating operators too. This class is formed by quantum actions leading to the collapse of quantum states. Further in the chapter, the mathematical model of quantum information processing systems. It is defined as an ensemble of interacting quantum actions on the common memory. The term "abstract quantum automata" is introduced to denote such model. At the end of the chapter models of some important quantum information processing systems are presented.

**Keywords:** Algorithmic solvability, computational complexity, quantum algorithm, formal specification, labelled transition system, operational semantics, Kraus' family, quantum action, abstract quantum automaton.

## 1 Introduction

In theoretical computer science it is accepted that to determine the possibility of using a computer to solve a problem it is necessary to answer two main questions. The questions are

- "Does there exist an algorithm solving the problem?";



- "Does there exist a possibility to provide enough computing power to run the algorithm?"

Studies aimed to find an answer to the first question have been begun by the works of A. Church [4], A.M. Turing [20,21], and E.L. Post [19]. Modern theory of computability is the result of the studies.

Similarly, efforts, aimed to developing methods for evaluating computational resources needed for problems solving, have led to the theory of computational complexity [5,11]. This theory provides methods to estimate the computational complexity of a solvable problem, and such estimation determines limits of the practical applicability of a computer to solve this problem.

Modern research shows that our computational capabilities make possible to solve problems with polynomial computational complexity. Unfortunately, science, engineering, and technology require to solve a lot of problems with more than polynomial computational complexity. Thus, developing non-classical computational systems, which have computing power much more than computing power of classical computational systems, is a challenge for computer scientists.

The idea to use quantum systems as computing devices appeared in the early eighties of the twentieth century. The idea's authors expected that such systems would provide overcoming the barrier of computational complexity.

In this context, the Yu.I. Manin's monograph [12], articles written by R.P. Feynman [9], and D. Deutsch [6] should be mentioned.

The hypothesis that quantum computers can overcome the complexity barrier, is based on the following reasoning: "... the quantum state space has far greater capacity than the classical one: for a classical system with  $N$  states, its quantum version allows superposition accommodates  $c^N$  states. When we join two classical systems, their number of states  $N_1$  and  $N_2$  are multiplied, and in the quantum case we get the exponential growth  $c^{N_1 N_2}$ . ... These crude estimates show that the quantum behaviour of the system might be much more complex than its classical simulation" [13].

And indeed, in 1992 D. Deutsch and R. Jozsa found the example of a problem, which has the estimation of the quantum computational complexity  $O(n)$ , while its classical computational complexity has exact estimation  $O(2^n)$  [7]. As known now, in addition to Deutsch–Jozsa problem there is quite a number of problems such that their solving process can be accelerated by using a quantum computer.

Thus, two questions, which have formulated at the beginning of the chapter, arise in the quantum case too.

To answer the first question it is necessary to develop a quantum analogue of the theory of computability. In our opinion, the general framework for such a theory was formulated by Yu.I. Manin [13]: "... we need a mathematical theory of quantum automata. Such a theory would provide us with mathematical models of deterministic processes with quite unusual properties. ... The first difficulty we must overcome is the choice of the correct balance between the mathematical and the physical principles. The quantum automaton has to be an abstract one: its mathematical model must appeal only to the general principles of quantum physics, without prescribing a physical implementation. Then

the model of evolution is the unitary rotation in a finite dimensional Hilbert space, and the decomposition of the system into its virtual parts corresponds to the tensor product decomposition of the state space ("quantum entanglement"). Somewhere in this picture we must accommodate interaction, which is described by density matrices and probabilities".

Therefore, we can determine two main objectives of the chapter: firstly, to build a rigorous mathematical model that provides tools for specification and verification of quantum information processing systems; secondly, to demonstrate usefulness of the model.

It should be stressed that following to Yu.I. Manin [12,13] we called the models introduced below by quantum automata. But there are some other models of quantum computational systems called quantum automata too [115]. As a rule they are obtained by applying a quantization procedure to classical state machines. Description of the relationship between our models and these models is an open problem.

## 2 Basic Notions and Notation

We need a few basic notions and notation. First of all, we note that Dirac notation [8] is used in the chapter as usual in quantum informatics [17].

In addition, we use the following notions and notation.

An  $n$ -dimensional linear space with an inner product is denoted by  $\mathcal{H}_n$ . It is called an  $n$ -dimensional Hilbert space. Let's suppose that the inner product is conjugate-linear in the first argument and linear in the second argument.

For a finite set  $\Lambda$  a Hilbert space of all complex-valued functions on  $\Lambda$  with inner product  $\langle f | g \rangle = \sum_{\alpha \in \Lambda} \overline{f(\alpha)}g(\alpha)$  is denoted by  $l^2(\Lambda)$ . Further for an arbitrary  $\alpha \in \Lambda$  by  $|\alpha\rangle$  we denote the function  $\delta(\alpha, \cdot)$  where  $\delta$  is the Kronecker delta. It is evident that the collection  $\{|\alpha\rangle : \alpha \in \Lambda\}$  forms an orthonormal basis in  $l^2(\Lambda)$ .

For a Hilbert space  $\mathcal{H}$  by  $\mathbf{1}$  the identity linear operator on  $\mathcal{H}$  is denoted.

For a Hilbert space  $\mathcal{H}$  and a linear operator  $A: \mathcal{H}_n \rightarrow \mathcal{H}$  by  $A^\dagger$  its adjoint operator is denoted. Remind that the relation between an operator and its adjoint operator is defined by the equation  $\langle \psi'' | A^\dagger | \psi' \rangle = \overline{\langle \psi' | A | \psi'' \rangle}$ .

For a Hilbert space  $\mathcal{H}$  a linear operator  $W: \mathcal{H}_n \rightarrow \mathcal{H}$  is called an isometric operator if for an arbitrary  $|\psi\rangle \in \mathcal{H}_n$  the next equality is true  $\|W|\psi\rangle\| = \|\psi\|$ .

For a linear operator  $A$  on a Hilbert space  $\mathcal{H}_n$  its trace is defined by the equation  $\text{Tr}(A) = \sum_{k=0}^{n-1} \langle k | A | k \rangle$  where vectors  $|0\rangle, \dots, |n-1\rangle$  form some orthonormal basis in  $\mathcal{H}_n$ .

A linear operator  $A$  on a Hilbert space  $\mathcal{H}_n$  is called a density operator if it is a nonnegative definite operator (i.e.  $\langle \psi | A | \psi \rangle \geq 0$  for all  $|\psi\rangle \in \mathcal{H}_n$ ) and its trace is equal to unit. It is known that each density operator can be represented as a convex combination of one-dimensional ortho-projectors [10]. Of course, a one-dimensional ortho-projector is a density operator. The set of all density operators is convex and the subset of one-dimensional ortho-projectors is the subset of its extreme points [10]. In the chapter the set of density operators on a space  $\mathcal{H}_n$  is denoted by  $\mathcal{S}_n$ .

In the chapter we use the next construction.

Let  $\mathcal{H}_n$  be an  $n$ -dimensional Hilbert space and  $\Lambda$  be a finite set. For each  $\alpha \in \Lambda$  let's define an operator  $J(\alpha): \mathcal{H}_n \rightarrow \mathcal{H}_n \otimes l^2(\Lambda)$  by the formula

$$J(\alpha)|\psi\rangle = |\psi\rangle \otimes |\alpha\rangle. \quad (1)$$

Properties of the operator family  $\{J(\alpha): \alpha \in \Lambda\}$  are established by the next lemma.

**Lemma 1.** *The next identities are true:*

$$J(\alpha)^\dagger \sum_{\alpha' \in \Lambda} (|\psi(\alpha')\rangle \otimes |\alpha'\rangle) = |\psi(\alpha)\rangle \quad (2)$$

$$J(\alpha')^\dagger J(\alpha'') = \delta(\alpha', \alpha'') \cdot \mathbf{1} \quad (3)$$

$$J(\alpha')J(\alpha'')^\dagger = \mathbf{1} \otimes |\alpha'\rangle\langle\alpha''| \quad (4)$$

*Proof.* Identities (2) and (3) are proved by direct calculation.

To prove identity (4) one can calculate  $J(\alpha')J(\alpha'')^\dagger$  and  $\mathbf{1} \otimes |\alpha'\rangle\langle\alpha''|$  on the same vector:

$$J(\alpha')J(\alpha'')^\dagger \sum_{\alpha \in \Lambda} (|\psi(\alpha)\rangle \otimes |\alpha\rangle) = J(\alpha')|\psi(\alpha'')\rangle = |\psi(\alpha'')\rangle \otimes |\alpha'\rangle,$$

$$(\mathbf{1} \otimes |\alpha'\rangle\langle\alpha''|) \sum_{\alpha \in \Lambda} (|\psi(\alpha)\rangle \otimes |\alpha\rangle) = \sum_{\alpha \in \Lambda} (|\psi(\alpha)\rangle \otimes |\alpha'\rangle\langle\alpha''| \alpha) = |\psi(\alpha'')\rangle \otimes |\alpha'\rangle.$$

Now one can conclude that identity (4) is true by comparing the two previous rows □

### 3 Physical Principles of Quantum Informatics

In the section we describe physical principles of quantum informatics. As in [17], they are formulated as postulates of quantum mechanics for systems with finite quantity of levels (finite-level quantum systems). The postulates introduce basic notions used to construct mathematical models of the systems.

#### 3.1 The Postulate of the State Space

**Postulate:** an  $n$ -dimensional Hilbert space  $\mathcal{H}_n$  is associated to any quantum physical system with  $n$  levels. This space is known as the state space of the system. The system is completely described by its pure state, which is a one-dimensional subspace of the state space. This subspace is uniquely represented by the ortho-projector  $|\psi\rangle\langle\psi|$  on the unit vector  $|\psi\rangle$  which generates the subspace.

In contrast to pure states mixed states are used to describe quantum systems whose state is not completely known.

Rather more detailed suppose we know that a quantum system is in one of a number of states  $\{|\psi_k\rangle\langle\psi_k| : k = 1, \dots, m\}$  with respective probabilities

$\{p_k: k = 1, \dots, m\}$ . We shall call  $\{p_k, |\psi_k\rangle\langle\psi_k| : k = 1, \dots, m\}$  an ensemble of pure states. The density operator for the system is defined by the equation  $\rho = \sum_{k=1}^m p_k |\psi_k\rangle\langle\psi_k|$ .

We identify mixed states with density operators. The statement that pure states are described by one-dimensional ortho-projectors allows to consider pure states as indecomposable states.

### 3.2 The Postulate of a Composite System

**Postulate:** the state space of a composite physical system is the tensor product of the state spaces of the component physical systems. Moreover, if we have systems indexed by  $k = 1, \dots, m$ , and the state of the system with number  $k$  is described by the density operator  $\rho_k$ , then the joint state of the total system before any interactions is  $\rho_1 \otimes \dots \otimes \rho_m$ .

### 3.3 The Postulate of a Quantum Evolution

**Postulate:** the evolution of a closed quantum system is described by a unitary transformation. That is, the state  $|\psi\rangle\langle\psi|$  of the system at time  $t_1$  is related to the state  $|\psi'\rangle\langle\psi'|$  of the system at time  $t_2$  by a unitary operator  $U$  which depends only on the times  $t_1$  and  $t_2$ ,  $|\psi'\rangle\langle\psi'| = U|\psi\rangle\langle\psi|U^\dagger$ .

If we have an ensemble of pure system states and this ensemble is described by the density operator  $\rho$  at the time  $t_1$  then the density operator  $\rho'$  at the time  $t_2$  can be calculated by the formula  $\rho' = U\rho U^\dagger$ .

### 3.4 The Postulate of a Quantum Measurement

**Postulate:** quantum measurements are described by a  $\Lambda$ -indexed finite family  $\mathcal{K} = \{K(\alpha): \alpha \in \Lambda\}$  of operators, where  $\Lambda$  is a finite set. These are operators acting on the state space of the system being measured. The symbol  $\alpha$  refers to the measurement outcome that may occur in the experiment. If the state of the quantum system is described by the density operator  $\rho$  immediately before the measurement then the probability that result  $\alpha$  occurs is given by the following formula

$$\Pr(\alpha | \rho) = \text{Tr}(\rho K(\alpha)^\dagger K(\alpha)) \quad (5)$$

and the state of the system immediately after the measurement is described by the density operator

$$\text{Eff}[\rho | \alpha] = \frac{K(\alpha)\rho K(\alpha)^\dagger}{\Pr(\alpha | \rho)} \quad (6)$$

Any  $\Lambda$ -indexed family  $\mathcal{K} = \{K(\alpha): \alpha \in \Lambda\}$ , which describes a quantum measurement, satisfies the **completeness condition**

$$\sum_{\alpha \in \Lambda} K(\alpha)^\dagger K(\alpha) = \mathbf{1}. \quad (7)$$

This condition ensures correctness of the definitions given by formulae (5) and (6).

A  $\Lambda$ -indexed family that satisfies the correctness condition is called a Kraus' family.

## 4 Quantum Measurements and Isometric Operators

The quantum evolution postulate and the quantum measurement postulate describe two different ways of changing a system state. It does not seem natural. Hence, we can set the problem: unify descriptions for evolutions and measurements of a finite-level quantum system. The aim of the section is to solve the problem.

### 4.1 Generating Operator of Kraus' Family

**Theorem 1.** *Let  $\Lambda$  be a finite set and  $\mathcal{K} = \{K(\alpha): \alpha \in \Lambda\}$  be a  $\Lambda$ -indexed family of operators on an  $n$ -dimensional Hilbert space  $\mathcal{H}_n$ .  $\mathcal{K}$  is a Kraus' family if and only if there exists an isometric operator  $W: \mathcal{H}_n \rightarrow \mathcal{H}_n \otimes l^2(\Lambda)$  such that the next condition is true*

$$K(\alpha) = J(\alpha)^\dagger W \text{ for all } \alpha \in \Lambda. \quad (8)$$

Moreover, the isometric operator  $W$  is uniquely defined by condition (8).

*Proof.* Suppose that  $\mathcal{K} = \{K(\alpha): \alpha \in \Lambda\}$  is a Kraus' family. Let's define an operator  $W: \mathcal{H}_n \rightarrow \mathcal{H}_n \otimes l^2(\Lambda)$  by the formula

$$W = \sum_{\alpha \in \Lambda} J(\alpha)K(\alpha). \quad (9)$$

We claim that the operator  $W$  is an isometric operator. To prove this it is sufficient to prove the equality  $W^\dagger W = \mathbf{1}$ .

Really,

$$\begin{aligned} W^\dagger W &= \left( \sum_{\alpha' \in \Lambda} J(\alpha')K(\alpha') \right)^\dagger \left( \sum_{\alpha'' \in \Lambda} J(\alpha'')K(\alpha'') \right) = \\ & \sum_{\alpha', \alpha'' \in \Lambda} K(\alpha')^\dagger J(\alpha')^\dagger J(\alpha'')K(\alpha''). \end{aligned}$$

Using this equality, identity (3), and the completeness condition (7) we can fulfil the next transformations

$$W^\dagger W = \sum_{\alpha \in \Lambda} K(\alpha)^\dagger K(\alpha) = \mathbf{1}.$$

Further, for any  $\alpha \in \Lambda$  one can obtain using identity (3)

$$J(\alpha)^\dagger W = J(\alpha)^\dagger \left( \sum_{\alpha' \in \Lambda} J(\alpha')K(\alpha') \right) = \sum_{\alpha' \in \Lambda} J(\alpha)^\dagger J(\alpha')K(\alpha') = K(\alpha).$$

Now suppose that  $K(\alpha) = J(\alpha)^\dagger W$  for all  $\alpha \in \Lambda$  and for some isometric operator  $W: \mathcal{H}_n \rightarrow \mathcal{H}_n \otimes l^2(\Lambda)$ , then

$$\sum_{\alpha \in \Lambda} K(\alpha)^\dagger K(\alpha) = \sum_{\alpha \in \Lambda} W^\dagger J(\alpha) J(\alpha)^\dagger W = W^\dagger \left( \sum_{\alpha \in \Lambda} J(\alpha) J(\alpha)^\dagger \right) W.$$

Using this equality, identity (4), and the isometric property for the operator  $W$  we obtain

$$\sum_{\alpha \in \Lambda} K(\alpha)^\dagger K(\alpha) = W^\dagger \left( \sum_{\alpha \in \Lambda} (\mathbf{1} \otimes |\alpha\rangle)\langle\alpha| \right) W = W^\dagger W = \mathbf{1}.$$

Hence, the completeness condition for the family  $\{K(\alpha): \alpha \in \Lambda\}$  is true.

Verify uniqueness of  $W$  now. Suppose that there are two different isometric operators  $W_1$  and  $W_2$  such that the next statement holds

$$J(\alpha)^\dagger W_1 = J(\alpha)^\dagger W_2 \text{ for all } \alpha \in \Lambda. \quad (10)$$

Multiplying the equality contained in statement (10) by  $J(\alpha)$  from left and summing over  $\alpha$  we get

$$\left( \sum_{\alpha \in \Lambda} J(\alpha) J(\alpha)^\dagger \right) W_1 = \left( \sum_{\alpha \in \Lambda} J(\alpha) J(\alpha)^\dagger \right) W_2.$$

From this equality and identity (4) one can derive that  $W_1 = W_2$ . The obtained contradiction proves uniqueness of the operator  $W$   $\square$

Theorem 1 grounds the next definition.

**Definition 1.** Let  $\mathcal{H}_n$  be an  $n$ -dimensional Hilbert space,  $\Lambda$  be a finite set, and  $\mathcal{K} = \{K(\alpha): \alpha \in \Lambda\}$  be a  $\Lambda$ -indexed Kraus' family then the isometric operator  $W: \mathcal{H}_n \rightarrow \mathcal{H}_n \otimes l^2(\Lambda)$  from Theorem 1 we call the **generating operator** of the Kraus' family  $\mathcal{K}$ .

## 4.2 Unified Model of Quantum Evolution and Quantum Measurement

Taking in account Theorem 1 we can reformulate the postulate of a quantum measurement.

**Corollary 1 (of Theorem 1).** The postulate of a quantum measurement is equivalent to the next statement: quantum measurements are described by an isometric operator  $W: \mathcal{H}_n \rightarrow \mathcal{H}_n \otimes l^2(\Lambda)$ , where  $\Lambda$  is a finite set. The index  $\alpha$  refers to the measurement outcome that may occur in the experiment. If the state of the quantum system is described by the density operator  $\rho$  immediately before the measurement then the probability that result  $\alpha$  occurs is given by the following formula

$$\Pr(\alpha | \rho) = \text{Tr}(\rho W^\dagger (\mathbf{1} \otimes |\alpha\rangle\langle\alpha|) W) \quad (5)$$

and the state of the system immediately after the measurement is described by the density operator

$$\text{Eff}[\rho \mid \alpha] = \frac{J(\alpha)^\dagger W \rho W^\dagger J(\alpha)}{\text{Pr}(\alpha \mid \rho)}. \quad (6)$$

**Definition 2.** *In the context of Corollary 7 we say that the isometric operator  $W$  is the generating operator of the quantum measurement.*

Now let's consider a quantum measurement which has a single outcome, i.e.  $\Lambda = \{\lambda\}$ .

In the case the next equalities  $l^2(\Lambda) = \mathbb{C}$  and  $\mathcal{H}_n \otimes l^2(\Lambda) = \mathcal{H}_n$  are true. Hence, any generating operator  $W: \mathcal{H}_n \rightarrow \mathcal{H}_n \otimes l^2(\Lambda)$  is a unitary operator  $W: \mathcal{H}_n \rightarrow \mathcal{H}_n$ . Formula (5) gives  $\text{Pr}(\lambda \mid \rho) = 1$  and from formula (6) we obtain  $\text{Eff}[\rho \mid \lambda] = W \rho W^\dagger$ .

Therefore, we can consider a quantum evolution as a quantum measurement with a one-element set of outcomes.

## 5 Quantum Actions

Unifying descriptions of a quantum evolution and a quantum measurement allows to introduce the generalising notion for evolutionary and measuring processes. We use the term a quantum action to designate this notion.

For physical reasons, we do not distinguish between actions that have the same pair  $(\text{Pr}(\cdot \mid \cdot), \text{Eff}[\cdot \mid \cdot])$ . But such pair does not determine an appropriate isometric operator uniquely as it shows the next simple proposition.

**Proposition 1.** *Let  $\mathcal{H}_n$  be a state space of  $n$ -level quantum system,  $\Lambda$  be a finite set, and  $W_1, W_2: \mathcal{H}_n \rightarrow \mathcal{H}_n \otimes l^2(\Lambda)$  be isometric operators such that  $W_2 = \left( \sum_{\alpha \in \Lambda} (\mathbf{1} \otimes e^{i\theta(\alpha)} |\alpha\rangle\langle\alpha|) \right) W_1$  for some  $\theta: \Lambda \rightarrow [0, 2\pi)$ , then for any  $\rho \in \mathcal{S}_n$  and  $\alpha \in \Lambda$  the next equalities are held*

$$\text{Tr}(\rho W_1^\dagger (\mathbf{1} \otimes |\alpha\rangle\langle\alpha|) W_1) = \text{Tr}(\rho W_2^\dagger (\mathbf{1} \otimes |\alpha\rangle\langle\alpha|) W_2), \quad (11)$$

$$J(\alpha)^\dagger W_1 \rho W_1^\dagger J(\alpha) = J(\alpha)^\dagger W_2 \rho W_2^\dagger J(\alpha). \quad (12)$$

*Proof.* The proposition is proved by a direct calculation □

### 5.1 Equivalence of Generating Operators for Quantum Measurements

Proposition 1 demonstrates that there are different isometric operators that generate the same pair  $(\text{Pr}(\cdot \mid \cdot), \text{Eff}[\cdot \mid \cdot])$ . Hence, different isometric operators  $W_1$  and  $W_2$  satisfying conditions (11) and (12) describe the same quantum action. This reasoning leads us to the next definition.

**Definition 3.** Let  $\mathcal{H}_n$  be a state space of  $n$ -level quantum system,  $\Lambda$  be a finite set, and  $W_1, W_2: \mathcal{H}_n \rightarrow \mathcal{H}_n \otimes l^2(\Lambda)$  be isometric operators, then we shall say that operators  $W_1$  and  $W_2$  **generate the same quantum action** if and only if conditions (I1) and (I2) are held.

It is evident that Definition 3 identifies equivalence relations on the sets of suitable isometric operators. The explicit form of these relations is given by the next theorem.

**Theorem 2.** Let  $\mathcal{H}_n$  be a state space of  $n$ -level quantum system,  $\Lambda$  be a finite set, and  $W_1, W_2: \mathcal{H}_n \rightarrow \mathcal{H}_n \otimes l^2(\Lambda)$  be isometric operators.

These operators generate the same quantum action if and only if there exists a function  $\theta: \Lambda \rightarrow [0, 2\pi)$  such that  $W_2 = \left( \sum_{\alpha \in \Lambda} (\mathbf{1} \otimes e^{i\theta(\alpha)} |\alpha\rangle\langle\alpha|) \right) W_1$ .

*Proof.* Taking in account Proposition 1 we can conclude that to prove the theorem it is sufficient to establish existence of a function  $\theta$  for isometric operators generate the same quantum action.

Let's suppose that isometric operators  $W_1, W_2: \mathcal{H}_n \rightarrow \mathcal{H}_n \otimes l^2(\Lambda)$  satisfy conditions (I1) and (I2).

Define the next vectors  $|\omega_k^{(s)}(\alpha)\rangle = J(\alpha)^\dagger W_s |k\rangle$  where  $|0\rangle, \dots, |n-1\rangle$  form an orthonormal basis in  $\mathcal{H}_n$ ,  $\alpha \in \Lambda$ , and  $s = 1, 2$ .

Easy to see, that

$$\langle \omega_k^{(s)}(\alpha) | \omega_k^{(s)}(\alpha) \rangle = \text{Tr}(|k\rangle\langle k| W_s^\dagger (\mathbf{1} \otimes |\alpha\rangle\langle\alpha|) W_s).$$

Taking in account equality (I1) one can obtain that the previous equality implies the equality

$$\langle \omega_k^{(1)}(\alpha) | \omega_k^{(1)}(\alpha) \rangle = \langle \omega_k^{(2)}(\alpha) | \omega_k^{(2)}(\alpha) \rangle. \quad (13)$$

For  $\alpha \in \Lambda$  by  $\Gamma(\alpha)$  denote the set of integers such that  $k \in \Gamma(\alpha)$  if and only if  $0 \leq k < n$  and  $\langle \omega_k^{(1)}(\alpha) | \omega_k^{(1)}(\alpha) \rangle = \langle \omega_k^{(2)}(\alpha) | \omega_k^{(2)}(\alpha) \rangle > 0$ . From equality (I2) it follows that for  $k \in \Gamma(\alpha)$  the next equality is true

$$|\omega_k^{(1)}(\alpha)\rangle \langle \omega_k^{(1)}(\alpha)| = |\omega_k^{(2)}(\alpha)\rangle \langle \omega_k^{(2)}(\alpha)|.$$

Therefore, for all  $\alpha \in \Lambda$  and any  $k \in \Gamma(\alpha)$  the next condition holds

$$|\omega_k^{(2)}(\alpha)\rangle = |\omega_k^{(1)}(\alpha)\rangle \frac{\langle \omega_k^{(1)}(\alpha) | \omega_k^{(2)}(\alpha) \rangle}{\langle \omega_k^{(2)}(\alpha) | \omega_k^{(2)}(\alpha) \rangle}.$$

Owing to (I3) the last equality means that for each  $\alpha \in \Lambda$  and any  $k \in \Gamma(\alpha)$  there exists  $\theta(\alpha, k)$  with the following properties

$$0 \leq \theta(\alpha, k) < 2\pi, \quad (14)$$

$$|\omega_k^{(2)}(\alpha)\rangle = e^{i\theta(\alpha, k)} |\omega_k^{(1)}(\alpha)\rangle. \quad (15)$$



Further, from equality (I2) one can derive that

$$|\omega_k^{(1)}(\alpha)\rangle\langle\omega_l^{(1)}(\alpha)| = |\omega_k^{(2)}(\alpha)\rangle\langle\omega_l^{(2)}(\alpha)|$$

where  $k, l \in \Gamma(\alpha)$  for some  $\alpha \in \Lambda$ .

This equality can be rewritten by using (I5) in the next form

$$|\omega_k^{(1)}(\alpha)\rangle\langle\omega_l^{(1)}(\alpha)| = e^{i(\theta(\alpha,k)-\theta(\alpha,l))} |\omega_k^{(1)}(\alpha)\rangle\langle\omega_l^{(1)}(\alpha)|.$$

Easy to see that the last equality implies  $e^{i(\theta(\alpha,k)-\theta(\alpha,l))} = 1$  and, therefore,  $\theta(\alpha, k) = \theta(\alpha, l)$ .

Thus, we proved that for any  $\alpha \in \Lambda$  and  $k \in \Gamma(\alpha)$  the next condition holds

$$|\omega_k^{(2)}(\alpha)\rangle = e^{i\theta(\alpha)} |\omega_k^{(1)}(\alpha)\rangle \text{ where } 0 \leq \theta(\alpha) < 2\pi.$$

If  $k \in \{0, \dots, n-1\} \setminus \Gamma(\alpha)$  for some  $\alpha \in \Lambda$  then  $|\omega_k^{(1)}(\alpha)\rangle = |\omega_k^{(2)}(\alpha)\rangle = 0$  and the previous condition holds too.

Using the definition of  $|\omega_k^{(s)}(\alpha)\rangle$  where  $s = 1, 2$  we obtain

$$J(\alpha)^\dagger W_2 = e^{i\theta(\alpha)} J(\alpha)^\dagger W_1.$$

Multiplying the last equality by  $J(\alpha)$  from left and summing over  $\alpha \in \Lambda$  we get

$$W_2 = \left( \sum_{\alpha \in \Lambda} (\mathbf{1} \otimes e^{i\theta(\alpha)} |\alpha\rangle\langle\alpha|) \right) W_1.$$

Theorem is proved □

**Corollary 2.** *Let  $\mathcal{H}_n$  be a state space of an  $n$ -level quantum system,  $\Lambda$  be a finite set of outcomes,  $W_1, W_2: \mathcal{H}_n \rightarrow \mathcal{H}_n \otimes l^2(\Lambda)$  be isometric operators that generate the same quantum action,  $\mathcal{K}_1 = \{K_1(\alpha): \alpha \in \Lambda\}$  and  $\mathcal{K}_2 = \{K_2(\alpha): \alpha \in \Lambda\}$  be respective Kraus' families then there exists a function  $\theta: \Lambda \rightarrow [0, 2\pi)$  such that the next condition holds*

$$K_2(\alpha) = e^{i\theta(\alpha)} K_1(\alpha) \text{ for each } \alpha \in \Lambda.$$

## 5.2 Formal Definition of Quantum Action

Now we can define the notion of a quantum action formally.

As it has been noted above, a quantum action is described by an appropriate pair  $(\text{Pr}[\cdot | \cdot], \text{Eff}[\cdot | \cdot])$ . Hence, we can suppose that a quantum action is a class of isometric operators that generate the same quantum action in compliance with Definition 3.

**Definition 4.** *Let  $\mathcal{H}_n$  be a state space of an  $n$ -level quantum system and  $\Lambda$  be a finite set then we call a **quantum action** on the system with set of outcomes  $\Lambda$  a class of isometric operators from  $\mathcal{H}_n$  into  $\mathcal{H}_n \otimes l^2(\Lambda)$  that generate the same quantum action.*

Moreover, if  $W: \mathcal{H}_n \rightarrow \mathcal{H}_n \otimes l^2(\Lambda)$  is a representative of such class then the probability to obtain an outcome  $\alpha$  is determined by formula (51) and the state after acting is determined by formula (61).

Hence, Definition 4 sets the following meaning of an action on a quantum system: if immediately before the action the system is described by the pure state  $\rho = |\psi\rangle\langle\psi|$  then the ensemble  $\{\text{Pr}(\alpha | \rho), \text{Eff}[\rho | \alpha] : \alpha \in \Lambda\}$  is an a priori description of the system immediately after the action.

### 5.3 Quantum Actions Leading to State Collapse

Let's remind that the important class of quantum measurements was introduced by J. von Neumann in [16]. Measurements of this class are described by Kraus' families formed by orthoprojectors. Hence, Kraus' families of measurements belonging to this class are orthogonal identity decompositions [10]. By a direct calculation one can prove the next proposition.

**Proposition 2.** *Let  $\mathcal{H}_n$  be a state space of  $n$ -level quantum system,  $\Lambda$  be a finite set,  $\mathcal{E} = \{E(\alpha) : \alpha \in \Lambda\}$  be an orthogonal identity decomposition,  $\rho$  be any density operator, and  $\alpha', \alpha''$  be any elements of  $\Lambda$ ; then*

$$\text{Pr}(\alpha'' | \text{Eff}[\rho | \alpha']) = \delta(\alpha'', \alpha'), \tag{16}$$

$$\text{Eff}[\text{Eff}[\rho | \alpha'] | \alpha'] = \text{Eff}[\rho | \alpha']. \tag{17}$$

Conditions (16) and (17) we call the **collapse conditions**. Hence, each von Neumann's measurement (it is called a projective measurement too) describes a quantum action which satisfies the collapse conditions.

We claim that each measurement satisfying the collapse conditions is equivalent to a projective measurement in compliance with Definition 3. More precise statement is given by the next theorem.

**Theorem 3.** *Let  $\mathcal{H}_n$  be a state space of  $n$ -level quantum system,  $\Lambda$  be a finite set of outcomes, and  $\mathcal{K} = \{K(\alpha) : \alpha \in \Lambda\}$  be a Kraus' family such that associated quantum action satisfies the collapse conditions then the family  $\mathcal{K}$  is equivalent to some orthogonal identity decomposition.*

*Proof.* Note that the next condition follows from equalities (16):

$$(K(\alpha)^2)^\dagger K(\alpha)^2 = K(\alpha)^\dagger K(\alpha) \text{ for all } \alpha \in \Lambda.$$

Hence,  $|K(\alpha)^2| = |K(\alpha)|$  and  $K(\alpha)^2 = U(\alpha)K(\alpha)$  for some unitary operator  $U(\alpha)$ .

Denote by  $\mathcal{L}(\alpha)$  the subspaces  $K(\alpha)\mathcal{H}_n$  and consider the decomposition  $\mathcal{H}_n = \mathcal{L}(\alpha) \oplus \mathcal{L}(\alpha)^\perp$ . The subspace  $\mathcal{L}(\alpha)$  is an invariant subspace of operator  $K(\alpha)$ . Therefore,  $K(\alpha) = \left( \begin{array}{c|c} K_1(\alpha) & K_2(\alpha) \\ \hline \mathbf{0} & \mathbf{0} \end{array} \right)$  where  $K_1(\alpha)$  is an operator on  $\mathcal{L}(\alpha)$  and  $K_2(\alpha)$  is an operator from  $\mathcal{L}(\alpha)^\perp$  into  $\mathcal{L}(\alpha)$ . The equality  $K(\alpha)^2 = U(\alpha)K(\alpha)$  implies that  $K_1(\alpha)$  is a unitary operator on  $\mathcal{L}(\alpha)$ .

Let  $\rho_1$  be some density operator on  $\mathcal{L}(\alpha)$  then  $\rho = \left( \begin{array}{c|c} \rho_1 & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} \end{array} \right)$  is a density operator on  $\mathcal{H}_n$ .

From (17) we get

$$K(\alpha)\rho K(\alpha)^\dagger = K(\alpha)^2\rho(K(\alpha)^2)^\dagger.$$

Using this equality and a matrix representation for  $K(\alpha)$  we obtain

$$K_1(\alpha)\rho_1 K_1(\alpha)^\dagger = K_1(\alpha)^2\rho_1(K_1(\alpha)^2)^\dagger.$$

Therefore, from unitarity of the operator  $K_1(\alpha)$  one can derive

$$K_1(\alpha)\rho_1 = \rho_1 K_1(\alpha).$$

Hence,

$$K_1(\alpha) = e^{i\theta(\alpha)} \cdot \mathbf{1} \text{ where } 0 \leq \theta(\alpha) < 2\pi.$$

Using Theorem 2 we can change the Kraus' family  $\{K(\alpha) : \alpha \in \Lambda\}$  by the Kraus' family  $\{K'(\alpha) = e^{-i\theta(\alpha)}K(\alpha) : \alpha \in \Lambda\}$ . After such changing we obtain that

$$K'(\alpha) = \left( \frac{\mathbf{1} | K'_2(\alpha)}{\mathbf{0}} \right) \text{ where } K'_2(\alpha) = e^{-i\theta(\alpha)}K_2(\alpha).$$

Easy to see that

$$K'(\alpha)^2 = K'(\alpha). \quad (18)$$

Note, that from (16) it follows that

$$\text{if } \alpha' \neq \alpha'' \text{ then } K'(\alpha')K'(\alpha'') = K'(\alpha'')^\dagger K'(\alpha')^\dagger = \mathbf{0}. \quad (19)$$

Multiplying from left the equality  $\sum_{\alpha' \in \Lambda} K'(\alpha')^\dagger K'(\alpha') = \mathbf{1}$  by  $K'(\alpha)^\dagger$  and using (18) and (19) we obtain

$$K'(\alpha)^\dagger K'(\alpha) = K'(\alpha)^\dagger.$$

Similarly, multiplying from right the equality  $\sum_{\alpha' \in \Lambda} K'(\alpha')^\dagger K'(\alpha') = \mathbf{1}$  by  $K'(\alpha)$  we obtain

$$K'(\alpha)^\dagger K'(\alpha) = K'(\alpha).$$

The last two equalities imply  $K'(\alpha)^\dagger = K'(\alpha)$ .

Therefore  $K'(\alpha)$  is an ortho-projector □

## 6 Abstract Quantum Automata

Each computer program can be presented as a finite set of interacting commands (or actions). The interaction is determined by calculating the successor for each command. This calculation is performed by a command itself. Chains of actions arise as a result of such calculations. They are known in computer science as control flows. The structure of control flows for a program is described by a finite state machine.

Hence, we can try to specify quantum systems of information processing by the similar way. Namely, we shall present a quantum system of information processing as a finite set of interacting quantum actions on the common finite-level quantum memory. Similarly to the classical case, the interaction between actions is determined by control flows. We use the notion of a deterministic labelled transition system for modelling control flows.

## 6.1 Labelled Transition Systems

Below we give the notion of a labelled transition system as it was defined in the R. Milner's monograph [14].

**Definition 5.** A *labelled transition system* over a finite set  $\Lambda$  is a pair  $(X, T)$  consisting of

- a finite set  $X$  of configurations;
- a ternary relation  $T \subseteq X \times \Lambda \times X$ , known as a transition relation.

If  $(x, \alpha, x') \in T$  we write  $x \xrightarrow{\alpha} x'$  and we call  $x$  the source and  $x'$  the target of the transition.

If  $x \xrightarrow{\alpha_1} x_1$  &  $x_1 \xrightarrow{\alpha_2} x_2$  & ... &  $x_{n-1} \xrightarrow{\alpha_n} x_n$  we call the alternate sequence  $x, \alpha_1, x_1, \alpha_2, \dots, \alpha_n, x_n$  the walk and write  $x \xrightarrow{\alpha_1} x_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} x_n$ .

The important class of labelled transition system is identified by the next definition.

**Definition 6.** A labelled transition system over a finite set  $\Lambda$  is called *deterministic* if for any  $x, x', x'' \in X$  and any  $\alpha \in \Lambda$  the following statement is true

$$x \xrightarrow{\alpha} x' \text{ \& } x \xrightarrow{\alpha} x'' \text{ implies } x' = x''.$$

## 6.2 Formal Definition of an Abstract Quantum Automaton

We shall consider the finite set  $\Omega$  of quantum actions on a common  $n$ -level quantum memory which is described by the state space  $\mathcal{H}_n$ . For an arbitrary  $q \in \Omega$  let denote by  $W_q$ ,  $\text{Pr}_q(\cdot | \cdot)$ ,  $\text{Eff}_q[\cdot | \cdot]$  its generating operator, probability of an outcome, and the state after performing it respectively.

**Definition 7.** A tuple  $\mathfrak{A}(\mathcal{H}_n, \Lambda, \Omega, q_\circ, H, T)$  we call by an *abstract quantum automaton* if it consists of

- a state space of an  $n$ -level quantum system  $\mathcal{H}_n$ ;
- a finite set of labels  $\Lambda$ ;
- a finite set of quantum actions  $\Omega$  on the state space  $\mathcal{H}_n$  such that their outcomes are elements of  $\Lambda$ ;
- a fixed quantum action  $q_\circ$  from  $\Omega$ ;
- a non-empty finite set  $H$ ;
- a deterministic labelled transition system  $(X, T)$  over  $\Lambda$  where  $X = \Omega \cup H$ ,

and satisfies the following conditions

$$x' \xrightarrow{\alpha} x'' \text{ implies } x' \in \Omega; \tag{20}$$

$$q \xrightarrow{\alpha} x \text{ if and only if there exists a density operator } \rho \text{ such that } \text{Pr}_{q'}(\alpha | \rho) > 0. \tag{21}$$

To define dynamics of an abstract quantum automaton we are in need of the following definition.

**Definition 8.** Let  $\mathfrak{A}(\mathcal{H}_n, \Lambda, \mathfrak{Q}, \mathfrak{q}_o, H, T)$  be an abstract quantum automaton, then a pair  $(\rho, x) \in \mathcal{S}_n \times X$  where  $X = \mathfrak{Q} \cup H$  is called an **automaton snapshot**.

Now let's describe an algorithm that determines a behaviour of an abstract quantum automaton:

1. let's assign  $t := 0$ ,  $x := \mathfrak{q}_o$ ,  $\rho := \rho_{\text{in}}$  where  $\rho_{\text{in}}$  is an initial state of the quantum memory;
2. let's assign  $x(t) := x$ , and  $\rho(t) := \rho$ ;
3. if  $x \in H$  then terminate performing algorithm;
4. let's choose an element  $\alpha_{t+1} \in \Lambda$  according to the stochastic distribution  $\Pr_x(\cdot | \rho)$ ;
5. let's transform  $\rho := \text{Eff}_x[\rho | \alpha_{t+1}]$ ;
6. let's choose  $x_{\text{new}}$  using the condition  $x \xrightarrow{\alpha_{t+1}} x_{\text{new}}$  and assign  $x := x_{\text{new}}$ ;
7. let's assign  $t := t + 1$ ;
8. go to item [2](#).

It is evident, that the algorithm of an abstract quantum automaton generates finite or infinite alternating sequences of the form

$$(\rho(0), x(0) = \mathfrak{q}_o), \alpha_1, (\rho(1), x(1)), \alpha_2, (\rho(2), x(2)), \alpha_3, \dots \quad (22)$$

which we call **runs**. Runs are characterised by the next simple proposition.

**Proposition 3.** Let  $\mathfrak{A}(\mathcal{H}_n, \Lambda, \mathfrak{Q}, \mathfrak{q}_o, H, T)$  be an abstract quantum automaton and  $r$  be a finite or infinite alternating sequence of the form [\(22\)](#) then  $r$  is a run if and only if it satisfies the next conditions:

$$\mathfrak{q}_o \xrightarrow{\alpha_1} x(1) \xrightarrow{\alpha_2} x(2) \xrightarrow{\alpha_3} \dots \text{ is a walk of the transition system } (X, T) \text{ over } \Lambda; \quad (23)$$

$$\text{if } x(t) \xrightarrow{\alpha_{t+1}} x(t+1) \text{ is a member of } r \text{ then } x(t) \in \mathfrak{Q}; \quad (24)$$

$$\text{if } r \text{ is finite and } \mathfrak{q}(T-1) \xrightarrow{\alpha_T} x(T) \text{ is its last member then } x(T) \in H; \quad (25)$$

$$\begin{aligned} &\text{if } \mathfrak{q}(t) \xrightarrow{\alpha_{t+1}} x(t+1) \text{ is a member of } r \text{ then} \\ &\Pr_{\mathfrak{q}(t)}(\alpha_{t+1} | \rho(t)) > 0 \text{ and } \rho(t+1) = \text{Eff}_{\mathfrak{q}(t)}[\rho(t) | \alpha_{t+1}]. \end{aligned} \quad (26)$$

*Proof.* Really, taking into account [\(20\)](#) and [\(21\)](#) it is easy to see that each run satisfies conditions [\(23\)](#), [\(24\)](#), [\(25\)](#), and [\(26\)](#). To prove the converse assertion it is necessary to use mathematical induction  $\square$

### 6.3 The Simplest Examples of Abstract Quantum Automata

We complete this section by two examples of modelling quantum information processing systems. Some two-level quantum systems are used in these examples. Such systems are called qubits.

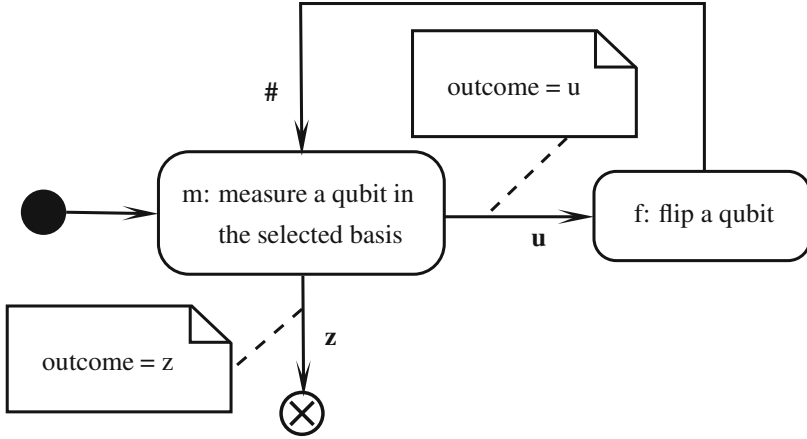


Fig. 1. Cleaning a qubit

*Example 1.* Let’s consider a quantum information process which sets a qubit into the state  $|0\rangle\langle 0|$  where  $|0\rangle, |1\rangle$  form a fixed orthonormal basis in  $\mathcal{H}_2$ . Evidently, this problem can not be solved by any unitary transformation. Below we specify an abstract quantum automaton that solves this problem.

We use the notation of UML activity diagrams [18] to specify a transition system of an abstract quantum automaton (see Fig. 1). As one can see  $\Lambda = \{z, u, \#\}$ . Let’s define  $W_m: \mathcal{H}_2 \rightarrow \mathcal{H}_2 \otimes l^2(\Lambda)$  by the formula

$$W_m = (|0\rangle \otimes |z\rangle)\langle 0| + (|1\rangle \otimes |u\rangle)\langle 1|.$$

Further, define  $W_f: \mathcal{H}_2 \rightarrow \mathcal{H}_2 \otimes l^2(\Lambda)$  by the formula

$$W_f = (|1\rangle \otimes |\#\rangle)\langle 0| + (|0\rangle \otimes |\#\rangle)\langle 1|.$$

Easy to see that this automaton has only two runs:

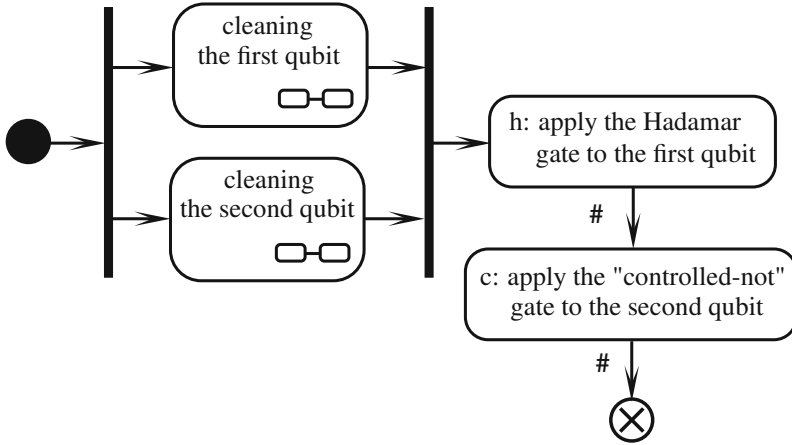
1.  $m, z, exit$ ;
2.  $m, u, f, \#, exit$ .

Direct checking shows that after handling by the automaton a qubit from a pure state  $|\psi\rangle\langle \psi|$  passes into the state  $|0\rangle\langle 0|$ .

Therefore, we have built the abstract quantum automaton that specifies the process of cleaning a qubit.

*Example 2.* This example deals with preparing an entangled pair of qubits. The process is used as a fragment of many quantum algorithms.

We shall specify an abstract quantum automaton that does it. The transition system of the automaton is shown in Fig. 2. Note that we use the previous example at the beginning of the specification.



**Fig. 2.** Preparing an entangled pair of qubits

The automaton state space equals  $\mathcal{H}_2^{\otimes 2}$  and  $\Lambda$  is equal to the set  $\{\mathbf{z}, \mathbf{u}, \#\}$  as in the previous example. As it follows from Example [11](#) the memory's state of the automaton is described by the density operator  $|0\rangle\langle 0| \otimes |0\rangle\langle 0|$  after cleaning two qubits. Apply to this pair a quantum action with the generating operator  $W_h$  defined by the formula

$$W_h = \frac{1}{\sqrt{2}} \sum_{k,l=0}^1 (((|0\rangle + (-1)^k|1\rangle) \otimes |l\rangle \otimes |\#\rangle) (\langle k| \otimes \langle l|)).$$

Then the pair passes into the state described by the vector  $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle$ . Further, apply a quantum action with the generating operator  $W_c$  defined by the formula

$$W_c = \sum_{k,l=0}^1 ((|k\rangle \otimes X^k|l\rangle \otimes |\#\rangle) (\langle k| \otimes \langle l|)), \text{ where } X|k\rangle = |k + 1 \pmod 2\rangle.$$

Easy to see that after all transformations the final state is described by the vector  $\frac{1}{\sqrt{2}}(|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle)$ .

## 7 Abstract Quantum Automata for Some Quantum Information Processing Systems

In the section abstract quantum automata for the qubit teleportation and Deutsch – Jozsa algorithm are built. This demonstrates the usefulness of the notion of an abstract quantum automaton.

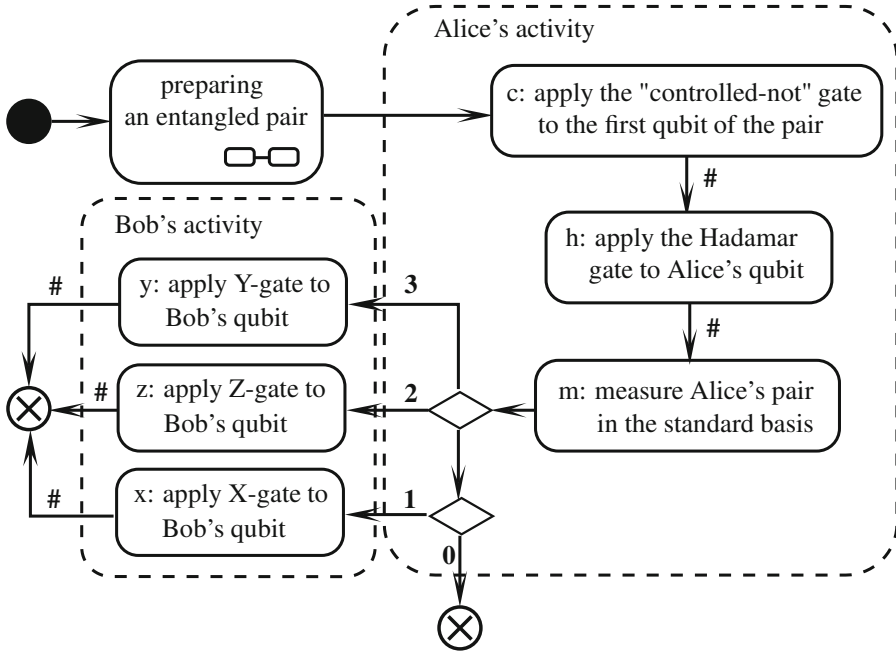


Fig. 3. Teleportation of a qubit state

### 7.1 Quantum State Teleportation

Quantum teleportation is a process by which a qubit state can be transmitted exactly from one location to another, without the qubit being transmitted through the intervening space. This phenomenon has been confirmed experimentally [2,3].

There are two participants, Alice and Bob, in the process. Their objective is to transmit a pure qubit state from Alice to Bob.

At the beginning we specify the transition system of the corresponding automaton. It is shown in the Fig. 3. Before the process Alice has a qubit that is in a state  $|\psi\rangle\langle\psi|$ .

The first step of the teleportation process a qubits entangled pair is prepared by using the abstract quantum automaton from Example 2.

Then the first qubit of the pair is sent to Alice and the second qubit of the pair is sent to Bob. Hence,  $\mathcal{H}_2^{\otimes 3}$  is the state space of the studied system,  $\Lambda = \{0, 1, 2, 3, \#\}$  is the set of labels, and  $\frac{1}{\sqrt{2}}|\psi\rangle \otimes (|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle)$  is the vector describing the system state before applying the quantum action associated with the node  $c$ .

The generating operator associated with the node  $c$  is defined by the formula

$$W_c = \sum_{k_0, k_1, k_2=0}^1 (|k_0\rangle \otimes X^{k_0}|k_1\rangle \otimes |k_2\rangle \otimes |\#\rangle) (\langle k_0| \otimes \langle k_1| \otimes \langle k_2|).$$



Let's choose some orthonormal basis in  $\mathcal{H}_2$ :  $|0\rangle, |1\rangle$  and define the generating operator associated with the node  $m$  by the following formula:

$$W_m = \sum_{k_0, k_1, k_2=0}^1 ((|k_0\rangle \otimes |k_1\rangle \otimes |k_2\rangle \otimes |k_0 + 2 \cdot k_1\rangle) (\langle k_0| \otimes \langle k_1| \otimes \langle k_2|)).$$

The respective action is performed by Alice. Then she sends the action outcome to Bob. Bob selects one among nodes  $x, y$  or  $z$  in compliance with the message received from Alice and applies the action associated with this node:

$$\begin{aligned} \mathbf{1} &: J(\#)(\mathbf{1}^{\otimes 2} \otimes X) \\ \mathbf{2} &: J(\#)(\mathbf{1}^{\otimes 2} \otimes Z), \text{ where } X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, Y = \begin{bmatrix} 0 & i \\ -i & 0 \end{bmatrix}, Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \\ \mathbf{3} &: J(\#)(\mathbf{1}^{\otimes 2} \otimes Y) \end{aligned}$$

By direct calculation one can check that after halting automaton its state is described by the density operator  $\frac{1}{2}(|0\rangle\langle 0| \otimes |0\rangle\langle 0| + |1\rangle\langle 1| \otimes |1\rangle\langle 1|) \otimes |\psi\rangle\langle\psi|$ .

## 7.2 Deutsch – Jozsa Algorithm

The Deutsch – Jozsa algorithm solves a problem, which is not important for applications. But this algorithm gives a simple example of an exponential decreasing computational complexity for solving problem at the expense of using quantum information processing system.

Let's  $f: \{0, 1\}^n \rightarrow \{0, 1\}$  be a boolean function. It is called balanced if the sets  $\{(x_1, \dots, x_n) : f(x_1, \dots, x_n) = 0\}$  and  $\{(x_1, \dots, x_n) : f(x_1, \dots, x_n) = 1\}$  have the same number of elements.

The Deutsch – Jozsa problem: let you have a boolean function of  $n$  variables and you know that it is balanced or constant. You should determine whether this function is balanced or it is constant.

It is evident that in the classical case it is necessary to compute the function  $O(2^n)$  times for answering the question. In contrast to the classical case the Deutsch – Jozsa algorithm solves this problem such that total number of its steps is estimated as  $O(n)$ .

The Deutsch – Jozsa algorithm is described by an abstract quantum automaton that uses a  $2^{n+1}$ -level quantum system as a memory. The space  $\mathcal{H}_2^{\otimes n} \otimes \mathcal{H}_2$  presents the corresponding state space.

$O(n)$  performing actions is needed to set the memory in the state which is described by the vector  $|\Psi_0\rangle = |0\rangle^{\otimes n} \otimes |1\rangle$ . This process is described by compositing automata defined above.

The generating operator of the next action is defined by the formula  $W_u = J(\#) \cdot H^{\otimes(n+1)}$  where  $H = \sum_{k,l=0}^1 (-1)^{k \cdot l} |k\rangle\langle l|$ . As result we obtain the state which is described by the vector

$$|\Psi_1\rangle = \frac{1}{\sqrt{2^n}} \left( \sum_{k_1, \dots, k_n=0}^1 |k_1\rangle \otimes \dots \otimes |k_n\rangle \right) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle).$$

The generating operator for calculating the function  $f$  is denoted by  $W_f$  and it is defined by the formula

$$W_f(|k_1\rangle \otimes \cdots \otimes |k_n\rangle \otimes |y\rangle) = |k_1\rangle \otimes \cdots \otimes |k_n\rangle \otimes |y + f(k_1, \dots, k_n) \pmod{2}\rangle \otimes |\#\rangle.$$

After applying the action generated by  $W_f$  and the action generated by  $J(\#) \cdot (\mathbf{1}^{\otimes n} \otimes H)$  we obtain the state which is described by the vector

$$|\Psi_f\rangle = \frac{1}{\sqrt{2^n}} \left( \sum_{k_1, \dots, k_n=0}^1 (-1)^{f(k_1, \dots, k_n)} |k_1\rangle \otimes \cdots \otimes |k_n\rangle \right) \otimes |1\rangle.$$

Let's denote by  $|\Xi\rangle$  the vector  $\frac{1}{\sqrt{2^n}}|0\rangle^{\otimes n}$ , by  $E(\mathbf{b})$  the ortho-projector  $|\Xi\rangle\langle\Xi| \otimes \mathbf{1}$ , and by  $E(\mathbf{c})$  the ortho-projector  $\mathbf{1} - E(\mathbf{b})$ . Then the measurement generated by the Kraus' family  $\{E(\mathbf{b}), E(\mathbf{c})\}$  gives with probability  $1 - \mathbf{b}$  if  $f$  is balanced, and  $\mathbf{c}$  if  $f$  is constant.

It is evident that complexity of this computation is estimated as  $O(n)$ .

We do not give a diagram of the corresponding automaton, because it is not difficult but is cumbersome.

## 8 Conclusion

Thus, the mathematical model for quantum information processing systems has been built in this chapter. This model describes quantum information processing as an interaction between elementary quantum actions.

Although some properties of quantum actions have been studied in the chapter the problems of investigating their structure and developing their synthesis methods are waiting to be solved.

It would be interesting to identify natural operations on the set of quantum actions and to describe the subset of irreducible elements for the operations on this set.

Finally, it is interesting to develop some software framework for simulating abstract quantum automata. Such framework can make possible to carry out computing experiments for finding new properties of quantum information processing systems.

## References

1. Ambainis, A., Freivalds, R.: 1-way quantum finite automata: strengths, weaknesses and generalizations. In: Proc. 39th Ann. Symp. on Found. Comp. Sci., pp. 332–341. IEEE (1998)
2. Bouwmeester, D., Pan, J.-W., Mattle, K., Eible, M., Weinfurter, H., Zeilinger, A.: Experimental quantum teleportation. Nature 390, 575–579 (1997)

3. Boschi, D., Branca, S., De Martini, F., Hardy, L., Popescu, S.: Experimental Realization of Teleporting an Unknown Pure Quantum State via Dual Classical and Einstein–Podolsky–Rosen Channel. *Phys. Rev. Lett.* 80, 1121–1125 (1998)
4. Church, A.: An unsolvable problem of elementary number theory. *Amer. J. Math.* 58(2), 345–363 (1936)
5. Cook, S.: The Complexity of Theorem Proving Procedures. In: *Proc. 3rd Ann. ACM Symp. on Theory of Computing*, pp. 151–158. ACM, New York (1971)
6. Deutsch, D.: Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. Roy. Soc. Lond., Series A* 400(1818), 97–117 (1985)
7. Deutsch, D., Jozsa, R.: Rapid Solution of Problems by Quantum Computation. *Proc. Roy. Soc. Lond., Series A* 439(1907), 553–558 (1992)
8. Dirac, P.A.M.: *The Principles of Quantum Mechanics*, 4th edn. Oxford University Press (1958)
9. Feynman, R.P.: Simulating Physics with Computer. *Int. J. Theor. Phys.* 21, 467–488 (1982)
10. Holevo, A.S.: *Probabilistic and Statistical Aspects of Quantum Theory*. North-Holland Publishing Company, Amsterdam (1982)
11. Karp, R.M.: *Complexity of Computation*. SIAM-AMS Proceedings, vol. 7. AMS, Providence (1974)
12. Manin, Y.I.: *Computable and Uncomputable (Cybernetics)*, Sovetskoe radio, Moscow (1980) (in Russian)
13. Manin, Y.I.: *Mathematics as metaphor: selected essays of Yuri I. Manin*. AMS (2007)
14. Milner, R.: *Communicating and Mobile System: the  $\pi$ -Calculus*. Cambridge University Press, Cambridge (1999)
15. Moore, C., Crutchfield, J.P.: Quantum automata and quantum grammars. *Theoret. Comput. Sci.* 237, 99–136 (2000)
16. Neumann von, J.: *Mathematische Grundlagen Der Quantenmechanik*. Verlag von Julius Springer, Berlin (1932)
17. Nielsen, M.A., Chuang, I.L.: *Quantum Computation and Quantum Information*. 10th Anniversary Edition. Cambridge University Press, Cambridge (2010)
18. OMG Unified Modelling Language (OMG UML), Superstructure. OMG, v 2.4.1 (2011), <http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF>
19. Post, E.L.: Finite Combinatory Processes – Formulation I. *J. Symb. Logics.* 1(3), 103–105 (1936)
20. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. *Proc. Lond. Math. Soc.* 2(42), 230–265 (1936)
21. Turing, A.M.: Computability and  $\lambda$ -Definability. *J. Symb. Logics.* 2(4), 153–163 (1937)

# Parallelizing Legacy Fortran Programs Using Rewriting Rules Technique and Algebraic Program Models

Anatoliy Doroshenko and Kostiantyn Zhereb

Institute of Software Systems of National Academy of Sciences of Ukraine,  
Glushkov prosp. 40, 03187 Kyiv, Ukraine  
{doroshenkoanatoliy2, zhereb}@gmail.com

**Abstract.** We present ongoing research in the area of transforming existing sequential Fortran programs into their parallel equivalents. We propose a semi-automated parallelization approach that uses rewriting rules technique to automate certain steps of the transformation process. A sequential source code is transformed into a parallel code for shared-memory systems, such as multicore processors. Parallelizing and optimizing transformations are formally described as rewriting rules which allows their automated application across the whole source code, and also facilitates their implementation and reuse. Using high-level algebraic models allows to describe program transformations in a more concise and stepwise manner. Performance measurements demonstrate the high efficiency of the obtained parallel programs, compared to the initial sequential programs and also to automated parallelization tools.

**Keywords:** rewriting rules technique, algebraic program models, multicore processors, Fortran, OpenMP.

## 1 Introduction

Despite being one of the first programming languages, Fortran is still widely used, in particular for solving scientific and engineering computation-intensive problems. Its popularity is due to its relative simplicity and lack of complex facilities (e.g. pointers), closeness to the mathematical description of problem and efficiency of the generated binary code. Another reason for continued use of Fortran is that in more than 50 years of its existence a vast repository of programs, libraries and routines for solving different scientific problems has been created. Algorithms implemented in such programs are still valuable, however there is a need to adapt this legacy code to new parallel computational platforms. Furthermore, due to the size and complexity of existing code, manual adaptation is not a practical option: there is a need for automated tools to facilitate the conversion of legacy code to modern parallel platforms [9].

In this chapter we describe ongoing research on parallelizing Fortran programs using formal methods: algebraic program models and rewriting rules technique. Sequential source code is transformed into a parallel code for shared-memory parallel

platform (such as multicore processors) using automated transformations. The code is represented as high-level algebraic models, facilitating their manipulation and enabling formal analysis methods, such as correctness proofs. The user has to select the suitable code fragments (usually loops) for parallelization, and the transformation to be applied (however, the selection of the suitable loops can be automated using dynamic analysis methods, as described in Section 4). Parallelizing and optimizing transformations are formally described as rewriting rules which facilitates their implementation and reuse. The rules are applied automatically across the whole source code, thus relieving the programmer from routine work and preventing copy-paste mistakes.

Our approach is aimed at two main goals: to improve runtime efficiency of programs and to increase developer's productivity. Therefore we use semi-automated parallelization approach, automating many routine steps in the parallelization process, yet enabling the programmer to make the changes that cannot be implemented using fully-automated tools. We illustrate our approach on two sample programs: a simple Gauss elimination algorithm and an applied problem of calculating electron density from the field of quantum chemistry. A comparison with selected automated tools demonstrates the efficiency of our approach.

This chapter continues our research on automation of the process of designing and development of efficient parallel programs, started in [3], [16], [17]. Our previous papers [3], [16] applied a similar approach to the development of parallel programs written in C# language for Microsoft .NET framework, while this chapter concentrates on parallelizing Fortran programs. We have already described our first experiences with Fortran programs in [17]. Now we describe an application of our approach to a real-world legacy code. Also this chapter places more significance on choosing place of application of existing program transformation (see Section 4), rather than developing new transformations.

Below we describe our approach in more detail, provide the examples of the parallelizing transformations and illustrate them with parallelization and evaluation of two programs: a small example program (Gauss elimination) and an applied quantum chemistry problem (electron density).

## 2 Our Approach: Algebraic Models and Rewriting Rules

As in our previous works [3], [16], [17], we use formal facilities, namely rewriting technique and high-level algebraic models of programs, to automate the process of parallelization of existing sequential code. Legacy source code of a sequential program written in Fortran is transformed into a parallel version targeting the shared-memory parallel platform (multicore processors). As a part of the transformation process, we create high-level algebraic models of legacy source code based on Glushkov algebra [4]. As described in [3] the models are created in two steps. First we use the target language parser (Fortran in this chapter) to build a low-level syntax model, and then rewriting rules of special form (*patterns*, see Section 2.3) to extract language-independent algebraic operators from language constructs. Using high-level algebraic models allows describing program transformations in a more concise

manner. The additional benefit of such models when applied to legacy code is that they aid in understanding of algorithms by hiding the (frequently obsolete) implementation details. To this end, using multiple levels of algebraic models can be useful – e.g. the highest level describes just general structure of algorithm, while lower levels supply the implementation details (the example of such models is described in Section 3).

After a high-level program model is created, we use the parallelizing transformations to implement a parallel version of the program on a given platform. Transformations are represented as rewriting rules and therefore can be applied in automated manner. (Selection of loops that could be transformed is performed manually – we provide GUI for exploring program models and marking selected loops with special mark symbols, see Section 3.2.) The declarative nature of rewriting rules simplifies adding new transformations. Also the transformations work with the high-level model elements (on any level of abstraction), which means they are language-independent.

As our research goes on, we plan to collect a library of reusable transformations that can be applied in different situations. Another goal is to provide the facilities powerful enough to implement any reasonable transformation. If this goal is achieved, the developer will be able to study an ideal manually created parallel version of a program, implement (as rewriting rules) the transformations that produce this version from a sequential program, and then reuse these transformations for other programs. So far, this principle has worked for us: we were able to implement all possible transformations for a simple problem (Gauss elimination, see Section 3) and then reused one of them for a more complex problem (electron density, Section 4). Of course, such simple case is not enough to validate our assumptions; in future, we plan to study more complex code, such as parallel benchmarks.

Usage of high-level algebraic models also allows to prove the correctness of the developed transformations [3]. Based on program models, we have developed the algebra-dynamic models of program execution for multicore architecture using discrete dynamic (transitional) systems [4]. For these models, we have (manually) proved that each of the proposed code transformations is correct under certain conditions, i.e. that initial and transformed programs are equivalent.

## 2.1 Algebraic Program Models

During all transformations we represent program code as an algebraic model. The model of entire program consists of a set of procedures:  $P = \{P_1, \dots, P_k\}$ . In case of Fortran program, the procedures correspond to subroutines or functions. Each procedure is described by an identifier (a name, unique within the program) and also by a code model:  $P_i = (name_i, code_i)$ . One procedure is designated as an entry point.

To model a code we use Glushkov algebra of algorithms [4]. Procedural code is represented as an expression of the algebra. Glushkov algebra is the two-sorted algebra  $A(Y, U)$  containing sets of operators  $Y$  and conditions (predicates)  $U$ . Standard operations of the algebra are determined: logic operations of conjunction

*AND*, disjunction *OR* and negation *NOT*, serial composition *THEN*, branching *IF*, iteration (loop) *WHILE*. (To simplify model description, we denote  $a;b = a \text{ THEN } b$ ). On the sets of operators and conditions basic elements are defined, and then various algebra expressions are built that can be described by compound operators and conditions. Basic operators and conditions usually depend on a subject domain. The common for all subject domains basic operator is a procedure call  $call(P_i)$ .

Algebraic program models are used to hide implementation details and represent only algorithmic structure of a program. Therefore, they could be considered language-independent. In practice, target language (e.g. Fortran) has a significant effect on the choice of basic operators and predicates. Still, if multiple language support is needed, missing basic operators from one language can be implemented as compound operators from the other.

## 2.2 Termware: Rewriting Rules Platform

Once the program model is built, we use rewriting rules to manipulate it, in particular to analyze the model and automate its transformations. We use the rewriting rules system Termware [15], initially written in Java and then ported to C#. The data in Termware is represented as *terms*, i.e. tree-like expressions of a form  $f(t_1, \dots, t_n)$ , where  $t_i$  are either terms or simple types (numbers, strings, booleans). Algebraic models of previous section have obvious representation as terms. Transformations are described as Termware *rules*, i.e. expressions of form `source [condition]-> destination [action]`.

Here `source` is a source term (a pattern that is matched across the whole input model), `condition` is an additional condition of rule application, `destination` is a transformed term, `action` is additional action that is performed when the rule fires. Each of 4 components of a rule can contain variables (denoted as `$var`), so that the rules are more generally applicable. The components `condition` and `action` are optional. They can execute any procedural code implemented in Java or C# class (Facts DB), in particular use the additional data on the program, e.g. identifiers table.

Termware supports a number of evaluation strategies, including `TopDown` (used in this chapter), `BottomUp` and a possibility to implement additional strategies. Termware system itself doesn't check that the transformation process terminates, however the rules used in this chapter are designed in such way that each model element is processed at most once (the rules in Section 4.2 provide an example), therefore the transformation process is guaranteed to terminate.

Termware could be compared to the functional programming languages such as Haskell. Its main differences are:

- Termware is not supposed to be used as a standalone language, but rather as a coordination part of a program written in Java or C#. Therefore, it lacks imperative features, such as I/O. Such features, if needed, can be implemented through actions.
- Unlike many functional languages, Termware is not strictly typed. This simplifies description of rules, although may have negative impact on performance.

In addition to rewriting system, our tools include parsers and generators for target languages that perform transformation between source code and a low-level (syntax) program model represented as Termware term. We have previously developed such tools for C# language [3], [16]; now we have developed also a Fortran parser and generator based on GCC Fortran Compiler.

### 2.3 Patterns

An important part of our transformation process is transformation between source code and algebraic program models. As already mentioned, such transformation is performed in two steps. First, source code is parsed and a syntax model is created. Then the syntax model is transformed into an algebraic model using rewriting rules of special type called *patterns*.

In general case a pattern is defined as a pair of rulesets:  $R_p$  – for extraction of a pattern from given term,  $R_g$  – for expanding a pattern. In more specific case the pattern is determined by a pair of terms  $t_p$  – a designation of pattern (an element of high-level model) and  $t_g$  – an implementation of pattern (an element of low-level model). In this case  $R_p = \{t_g \rightarrow t_p\}$  and  $R_g = \{t_p \rightarrow t_g\}$ .

For each of high-level operators we define a pattern consisting of the operator  $t_p$  and its implementation in terms of low-level model  $t_g$ . Then to create a high-level model from a low-level one we apply  $R_p$  rulesets for each pattern. Similarly, we apply  $R_g$  rulesets for transformation from high-level to low-level model.

## 3 Parallelization for Shared-Memory Systems Using OpenMP

In this section we describe the process of parallelizing sequential Fortran programs for parallel systems with shared memory, such as multicore processors. The source code of Fortran programs is parallelized by replacing suitable loops with the parallel loop constructs. To create multithreaded Fortran program we use OpenMP framework [11]. OpenMP *PARALLEL DO* directives are used to parallelize loops. For simple loops, just addition of such directive can produce quite efficient parallel code. In this case there is additional advantage of keeping transformed parallel code similar to existing sequential code. In more complex cases (when there is data dependency between iterations) there is a need of more significant transformations, such as using OpenMP library subroutines for advanced thread management. In such cases, the transformed source code contains significant changes. However, usage of high-level algebraic models allows describing these changes in concise and understandable form.

### 3.1 Program Example: Gauss Elimination

We will describe the details of our approach using as an example a Fortran program implementing Gauss elimination algorithm for solving systems of linear algebraic



equations. The Fortran source code was transformed into a low-level syntax model using a parser we have developed, then into a high-level algebraic model using Termware patterns. When working with legacy code, we found it useful to apply several levels of patterns. First we used generic linear algebra patterns, such as vector and matrix operations. The obtained algebraic model was language-independent, but still quite detailed. Then we applied patterns specific to the problem in question. In this way we obtained schematic representation of algorithm useful for its understanding and deciding where parallelizing transformations should be applied.

The high-level model of a relevant code fragment has the following form:

```
DoCnt (K, 1, N-1,
      FindMaxElement; CheckDetZero; SwapMaxRowColumn;
      CalculateRow(K); UpdateElements )
```

We will parallelize only two of the operators present in the program, namely `FindMaxElement` and `UpdateElements`. Other operators have less computational complexity, therefore their parallelization is less effective.

### 3.2 Parallelizing Loops without Dependencies

Out of two operators, the simplest is `UpdateElements`, responsible for calculating new values for elements of submatrix:

```
UpdateElements= DoCnt (I, K+1, N, Assign (S, A(I, K)),
                      DoCnt (J, K, N+1, Update(A(I, J), S)))
```

Here, **DoCnt** denotes common DO loop with counter. The iterations of the outer loop are independent, so this fragment is easily parallelized. We use the following rewriting rule:

```
DoCnt ($var, $start, $end, $body, _MARK_Parallel) ->
ParallelDoCnt ($var, $start, $end, $body)
```

The user marks the loop to be transformed with a `_MARK_Parallel` symbol to enable rule application. **ParallelDoCnt** operator is high-level model element responsible for parallel loop. In particular, for OpenMP platform it is transformed into a `OmpParallelDo` operator that describes OpenMP directive represented in Fortran as a pair of special comments: `!$OMP PARALLEL DO ... !$OMP END PARALLEL DO`.

Notice that for C language the same `OmpParallelDo` operator is represented as a single pragma statement: `#pragma omp parallel for`. Therefore, using multiple levels of patterns allows us to provide the operators that are common for given platforms, use these generic operators in most rewriting rules and then specialize them only when transforming the program model back into source code.

### 3.3 Reduction Case

While `UpdateElements` operator can be parallelized by simple application of OpenMP directive, the other operator `FindMaxElement` is more complex. It has the following form:

```
FindMaxElement= DoCnt (I, K, N,
                    DoCnt (J, K, N,
                        GetMaxIndex (Abs (A (I, J)) , I, J, Max, Imax, Jmax) ) )
```

Iterations of the loop update the same set of variables (value of the maximum element in submatrix *Max* and its indices *Imax*, *Jmax*). This is the case of reduction, when some local values are calculated on each iteration and then merged into one global value. OpenMP supports reduction loops with the `REDUCTION` clause, however it supports only a set of predefined reduction operators: while finding just maximum value can be accomplished using OpenMP directives, finding maximum value and indices where it occurs is not directly supported.

Therefore we need to provide transformations that parallelize the loop in general case of reduction. Our goal is to represent *FindMaxElement* as the following combination:

$$FindMaxElement = FindMaxElLoc1; \dots ; FindMaxElLocN; FindMaxElReduct$$

On each thread we execute a local version of the operator (*FindMaxElLoc1*, ..., *FindMaxElLocN*), and then execute the reduction operator *FindMaxElReduct* that combines the local values into one global value. In this case, the local operators *FindMaxElLoc1*, ..., *FindMaxElLocN* are independent and can be executed in parallel.

In order to obtain such decomposition, we need to look at the structure of *GetMaxIndex* operator:

$$GetMaxIndex(Val, I, J, Max, Imax, Jmax) = \\ \text{If}(Val > Max, \text{Assign}(Max, Val); \text{Assign}(Imax, I); \text{Assign}(Jmax, J))$$

Let's group the parameters of the operator in the following way:  $a = (Val, I, J)$ ,  $b = (Max, Imax, Jmax)$  and denote  $GetMaxIndex(Val, I, J, Max, Imax, Jmax) = GMI(a, b)$ . Notice that the *GMI* operator is associative:

$$GMI(GMI(a, b), c) = GMI(a, GMI(b, c))$$

Therefore, we can split the matrix into a number of regions, compute *GMI* operator on each of them (*FindMaxElLoc* operator) and then combine the obtained results using the same *GMI* operator (*FindMaxElReduct* operator):

```
FindMaxElLoc (t) = DoCnt (I, K (t) , N (t) ,
                        DoCnt (J, K, N
                            GetMaxIndex (Abs (A (I, J)) , I, J, Max (t) , Imax (t) , Jmax (t) ) ) )
```

Here,  $t$  is a number of a thread,  $K(t)$ ,  $N(t)$  – bounds of a matrix region, assigned to this thread,  $MAX(t)$ ,  $IMAX(t)$ ,  $JMAX(t)$  – the local versions of results.

```
FindMaxElReduct = DoCnt (t, 1, nthreads,
                        GetMaxIndex (Max (t) , Imax (t) , Jmax (t) , Max, Imax, Jmax) )
```

Therefore we use the following rules to parallelize *FindMaxElement* operator:

1. *FindMaxElement* -> **Parallel** (GetThreadParams; FindMaxElLoc); *FindMaxElReduct*
2. *GetThreadParams* -> **Assign** (t, GetThreadNum); **Assign** (nthreads, GetThreads)

3. FindMaxElLoc->**ParallelDoCnt** (I,K,N, **DoCnt** ((J,K,N,  
GetMaxIndex (Abs (A (I,J) ) , I,J,  
Maxloc (t) , Imaxloc (t) , Jmaxloc (t) ) )
4. FindMaxElReduct-> **DoCnt** (t,1,nthreads,  
GetMaxIndex (Maxloc (t) , Imaxloc (t) , Jmaxloc (t) ,  
Max, Imax, Jmax)

The rule 1 describes decomposition of FindMaxElement operator into number of FindMaxElLoc operators executed in parallel, followed by FindMaxElReduct operator. The rule 2 is needed to save total number of threads nthreads and position of current thread t that is used to store local copies of data. Rules 3 and 4 implement FindMaxElLoc and FindMaxElReduct operators using GetMaxIndex operator.

### 3.4 Optimizing Memory Performance

Both already described parallelizing transformations are aimed at the high-level structure of algorithm. However, as we observed in [3], low-level implementation details, e.g. memory access, can have a profound impact on the overall performance.

In the Gaussian elimination program we have observed the same effect. We noticed that for certain sizes of input matrix ( $N=256*M$ ) there was a sudden increase of the execution time. We attribute this increase to the peculiarities of memory access: namely, caching adjacent matrix elements. For such matrix size, the adjacent matrix elements were put into the same cache items, therefore increasing the number of cache misses and greatly reducing overall performance. To overcome this peculiarity, we declare the matrix size as  $N+1$  instead of  $N$ . The extra elements are not used in calculations, but they change the location of the elements and improve the efficiency of memory access. The transformation is implemented with the following rules:

1. [**Declaration** (N, Integer, \$val) : \$next] ->  
  [**Declaration** (N, Integer, \$val) :  
  [**Declaration** (MN, Integer, \$val+MShift (\$val)) : \$next]]
2. MShift (\$val) [\$val%32==0] -> 1 !-> 0
3. **Declaration** (A, Array (Double, [N,N+1])) ->  
  **Declaration** (A, Array (Double, [MN,MN+1]))
4. **Procedure** (\$name, [N: [A: \$next]]) ->  
  **Procedure** (\$name, [N: [MN: [A: \$next]]])
5. [**Parameter** (N, Integer, In) : \$next] -> [**Parameter** (N, Integer,  
  In) : [**Parameter** (MN, Integer, In) : \$next]]
6. **Call** (\$name, [N: [A: \$next]]) ->  
  **Call** (\$name, [N: [MN: [A: \$next]]])

The rule 1 adds a new parameter,  $MN$ , denoting declared matrix size. (Currently variable name is provided by the developer who should ensure that it does not coincide with existing variables. In future versions of Termware we plan to provide the means to query already defined variables in any point of program and to generate a variable name that was not defined.) The rule 2 specifies for which values of matrix size the transformation should be applied. It uses an extension of basic rule syntax: `src[cond] -> dest1 !-> dest2`, meaning that `src` is transformed into `dest1`

if `cond` holds, and into `dest2` otherwise.. The rule 3 modifies matrix declaration to use new size  $MN$  instead of  $N$ . Rules 4-6 propagate new parameter to all procedures, procedure parameters and procedure calls.

Notice that rules 4-6 are applied multiple times in a single program: for each procedure definition (rules 4-5) and for each procedure call (rule 6). One of the advantages of rewriting technique is that single rule can describe changes in multiple places, reducing the effort to make the changes and preventing the mistakes possible when applying such changes manually. Notice also that rules 1-6 work on a lower level of abstraction compared with previously described rules. The ability to describe transformations on different model levels is another advantage of proposed approach allowing to describe different types of transformations with the same tools.

## 4 Real-World Example: Electron Density Program

After developing our tools on a sample problem (Gauss elimination) we have tried them on a real-world program in the area of quantum chemistry. The program calculates electron and spin density in atoms of polycyclic aromatic hydrocarbons on a  $N*N$  grid [29]. The size of the program is 1680 lines of Fortran code. Source code is not well structured – actual calculations are mixed with I/O operations, debug code and some hardcoded data. Also it contains mix of features from different versions of language – from Fortran 77 to Fortran 95. Therefore usage of high-level algebraic models helped us to understand this legacy code and apply parallelizing transformations in the most efficient way.

### 4.1 Finding Hotspots Using Profiler

We were able to reuse the parallelizing transformations developed for Gauss elimination program also in electron density program. Only the first, most simple loop transformation was applied. However, the challenge was to select the most suitable loop for this transformation, as the program contained 54 loops and trying all of them was not a feasible option. To this end, we have used a profiler tool, Intel VTune Amplifier [7], to find hotspots in source code.

The most computationally intensive part of the program resided in the `ELDENS` subroutine, responsible for the calculation of electron density in a given point of a grid. The calculations are rather simple, but they are nested in multiple loops and therefore executed repeatedly. Fig. 1 shows the results obtained by the profiler.

Using a profiler tool provides dynamic analysis capabilities that complement static analysis performed by means of algebraic models and rewriting technique. In this way, runtime behavior of a program can be analyzed. The drawback of such an approach is dependency on input data: if program behavior can vary significantly depending on inputs (e.g. because of branching), such analysis becomes unreliable. However, for many applied problems the computationally intensive parts of algorithm are the same for all possible inputs.

### 4.2 Finding a Set of Enclosing Loops

After the hotspots have been found, we need to decide how we can parallelize them. Our transformations work by parallelizing loops; therefore, we need to find the loop

that encloses the found hotspots and can be parallelized. The easiest approach would be to take the closest (innermost) loop. It can be easily identified manually, just by looking at source code (in Fig. 1, this loop spans lines 1010-1015 and starts with `do j=ia, i` statement).

Line	Source	CPU Time	Instructions Retired
1000	cc - new version - NDDO orbitals only 04 May 2006		
1001	sum = 0.0d0	0.001s	6,000,000
1002	sum_1 = 0.0d0	0.001s	2,000,000
1003	do ii=1,natoms	0.006s	26,000,000
1004	ia=NFIRST(ii)	0.372s	1,498,000,000
1005	ib=NLAST(ii)	0.346s	444,000,000
1006	do i=ia,ib	0.379s	1,512,000,000
1007	sum = sum - PSI(i)*PSI(i)*PA(i,i)*0.5d 0	6.792s	20,576,000,000
1008	if(ical1.eq.2) sum_1=sum_1-PSI(i)*PSI(i)*PB(i,i)*0.5d 0	2.844s	2,200,000,000
1009	c write(6,'(1x,"I=",i3,"; S=",f10.5,"; SUM=",f10.5)') i, s, sum		
1010	do j=ia,i	0.462s	1,306,000,000
1011	sum = sum - PSI(i)*PSI(j)*PA(i,j)	7.428s	38,436,000,000
1012	if(ical1.eq.2) sum_1=sum_1-PSI(i)*PSI(j)*PB(i,j)	7.317s	34,388,000,000
1013	c write(6,'(1x,"I=",i3,"; J=",i3,"; S=",f10.5,"; SUM=",		
1014	c 1 f10.5)') i, s, sum		
1015	end do ! loop j	0.591s	2,066,000,000
1016	end do ! loop i	1.794s	11,412,000,000
1017	end do ! loop ii	0.084s	324,000,000
1018	sum=sum+sum	0.002s	26,000,000
1019	if(ical1.eq.2) sum_1=sum_1+sum_1		6,000,000
1020	sum_new=sum	0.001s	2,000,000
1021	c write(6,'(1x,"New version: SUM=",f10.5)') sum		
1022			
1023	cc - new version - NDDO orbitals only 04 May 2006		
1024			

Fig. 1. Results of the profiler tool

However, such approach is inefficient, because of the following reasons:

- Some of the statements that do not belong to this loop are also quite computationally intensive (see, e.g., lines 1007, 1008, 1016 in Fig.1).
- The loop itself is quite small and it is repeated multiple times. In OpenMP, an end of a parallel loop is a synchronization point, therefore, the synchronization overhead will be quite significant. (The advantages of choosing the outer loops for parallelization have been discussed in literature [49]; see also the discussion in Section 5.3)

Therefore, we would like to obtain all loops enclosing the hotspot code fragments. This includes both the loops inside ELDENS procedure and the loops in other procedures that call ELDENS. To find all such loops, we can use rewriting rules. We mark the discovered hotspot fragments by wrapping the corresponding model element inside `_MarkHotspot` term. The following rules were used:

1. `[_MarkHotspot ($x) : $y] -> _MarkHotspot ( [ $x : $y ] )`
2. `[ $x : _MarkHotspot ($y) ] -> _MarkHotspot ( [ $x : $y ] )`
3. `DoCnt ($var, $start, $end, _MarkHotspot ($body) ] -> _MarkHotspot ( DoCnt ($var, $start, $end, $body, _MARK_CAND_PARALLEL) )`

4. **Procedure**(\$name, \$params, \$return, \_MarkHotspot(\$body))  
 -> **Procedure**(\$name, \$params, \$return, \$body, \_MARK\_PASSED)  
 [addItem(\$name)]
5. **Call**(\$name, \$params) [hasItem(\$name)] ->  
 \_MarkHotspot(**Call**(\$name, \$params))

Here, the rules 1 and 2 are used to spread the mark to neighbor elements within the same loop. The rule 3 is activated once the whole loop body is marked; it marks the loop with a `_MARK_CAND_PARALLEL` symbol and promotes `_MarkHotspot` to a higher level. The rules 4 and 5 are needed to perform interprocedural analysis. The rule 4 activates when the whole body of a procedure is marked; it saves procedure name using `addItem()` method. Procedure name is stored in the Facts DB associated with the ruleset and used in the rule 5: when one of the procedures that contain hotspots is called, it is also marked with a `_MarkHotspot` term. In order to guarantee that the rewriting process terminates, the rule 4 marks processed procedure with a `_MARK_PASSED` symbol. Therefore, the rule applies to each procedure at most once.

By applying the rewriting rules, we obtained 6 candidate loops that should be considered for parallelization. In this way, using combination of static analysis (rewriting rules) and dynamic analysis (profiler), we were able to reduce number of possible loops from 54 to 6.

### 4.3 Selecting a Loop for Parallelization

The candidate loops have the following structure:

```
DO IB=1, NZ
  DO KD=1, NY
    DO MS=1, NX
      CALL ELDENS
SUBROUTINE ELDENS
  DO II=1, NATOMS
    DO I=IA, IB
      DO J=IA, I
        <hotspot>
```

When choosing the loops for parallelization, we use the following guidelines:

- try to parallelize the outermost loop, as it will both include the larger amount of code and reduce number of synchronizations (as described in Section 4.2)
- ignore the loops that cannot be parallelized (e.g. because of dependencies between loop iterations)
- ignore small loops (with a number of iterations less than a number of cores).

According to these guidelines, we ignore the outermost loop (on `IB`), as in input data often `NZ=1`. The second loop (on `KD`) can be parallelized using OpenMP directives, so we use it as the transformation target.

## 5 Performance Evaluation

To evaluate the effects of the developed transformations, we have measured the performance of different versions of the two programs. All measurements were performed on a shared memory parallel system with Intel Core2 Quad Q8200 CPU (4 cores, 2.33 GHz) and 4GB DDR2 RAM. Programs were compiled using Intel Fortran Compiler 12.1 (included in Intel Parallel Studio XE 2011).

### 5.1 Gauss Program

For the Gauss program, we have compared the performance of 4 versions: an initial sequential program (SEQ), a parallel program with the `UpdateElements` operator parallelized (PAR1), a parallel program with both `UpdateElements` and `FindMaxElement` operators parallelized (PAR2), and a program with both operators parallelized and memory optimization applied (MEM). The measurements were performed for matrix sizes from 256 to 2048, for multiples of 256 only (to demonstrate the difference of MEM program; for other values of  $N$ , the behavior of the MEM program is the same as the PAR2 program). Obtained speedup (compared to the SEQ program) is shown on Fig. 2.

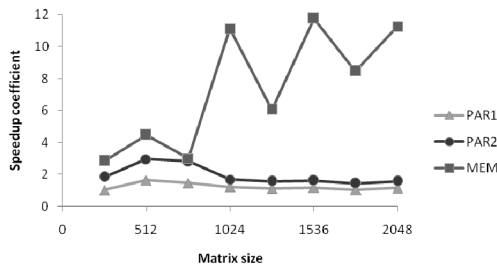


Fig. 2. Speedup of the transformed programs (Gauss elimination)

As can be seen from the diagram, all transformations result in some performance increase, although their effect differs. For small matrix sizes, both PAR1 and PAR2 show some noticeable speedup, while MEM is not very effective and is very close to PAR2. However, for larger matrix sizes ( $N > 1024$ ), the situation changes. PAR1 and PAR2 become less efficient, close to SEQ (this effect holds for the multiples of 256 only; for the data related to other values, see Section 5.3). However, MEM becomes much more efficient and demonstrates the speedup of more than 10X. Therefore both high-level transformations of algorithms and taking care of low-level implementation details are necessary to obtain the efficient parallel programs. Measurement results also demonstrate the complex dependency of the execution time on real parallel systems, as compared to ideal theoretical models that suggest a simple  $O(N^3)$  dependency.

## 5.2 Electron Density Program

For the electron density program, we have compared the execution time of the initial sequential program (SEQ) and the parallelized program (PAR) for grid dimensions  $N$  from 200 to 800 (see Fig. 3).

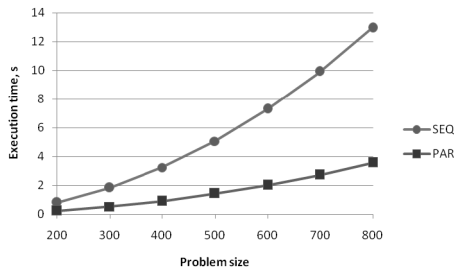


Fig. 3. Comparison of initial and transformed program (electron density)

Applying transformations has resulted in quite significant speedup – from 3.3X to 3.6X (depending on the problem size) on a 4-core system.

## 5.3 Comparison with Auto-Parallelizing Tools

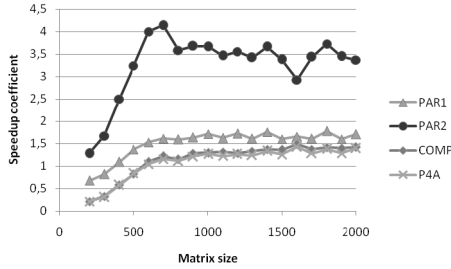
In order to compare our approach with other parallelization solutions for Fortran, we have used two popular tools for automatic parallelization: Intel Fortran Compiler [7] and Par4All source-to-source transformation tool [51]. Both tools were used to parallelize the initial sequential version of the Gauss elimination program.

In case of Intel Fortran Compiler, we used the following parameters related to parallelization: enable parallelization (**/Qparallel**); threshold for auto-parallelization (**/Qpar-threshold:70**); interprocedural optimization (**/Qipo**); optimization level (**/O3**); complete auto-parallelizer report (**/Qpar-report3**). By running the compiler with these parameters, we obtained a parallelized executable file and also a parallelization report, describing the loops that were parallelized.

Par4All tool was used without additional parameters and produced a new version of Fortran code with OpenMP directives inserted for some of the loops. However, we had to slightly modify the source code: in particular, the Fortran 95 variable declarations and comments were changed to the Fortran 77 format. The obtained OpenMP code was compiled using Intel Fortran Compiler.

We have measured the performance of 5 versions of Gauss elimination program: SEQ – an initial sequential version; PAR1 – a parallel program with the UpdateElements operator parallelized; PAR2 – a parallel program with the both UpdateElements and FindMaxElement operators parallelized; COMP – a program parallelized using Intel Fortran Compiler (same source code as for the SEQ version, but compiled with different parameters); P4A – a program parallelized with a Par4All tool. We have excluded from this comparison the MEM version (with memory access optimizations), as such optimization is not supported by parallelizing tools, and also it only works for selected values of matrix size. The speedup of the parallel versions (compared to the initial SEQ version is shown on the Fig. 4.





**Fig. 4.** Comparison with auto-parallelizing tools

As can be seen from the experimental results, the automatic parallelization approach provides a speedup of about 1,5X for large matrix sizes ( $N > 1000$ ). Both Intel Fortran Compiler and Par4All tool provide very close results (although Intel Fortran Compiler is slightly better for large matrix sizes). Indeed, looking at the tools output (transformed source code with OpenMP directives in case of Par4All and parallelization report in case of Intel Fortran Compiler), we can see that they have parallelized the same loops. When comparing the results of the auto-parallelization tools with our transformations (implemented in the PAR1 and PAR2 programs), the following key differences can be identified:

1. The auto-parallelizing tools have parallelized many small loops, such as those in the `SwapMaxRowColumn` and `CalculateRow(K)` operators, and also outside of the code fragment that we considered. Our approach was to select only the most computationally intensive,  $O(N^3)$  loops, because they provide noticeable speedup. We have ignored  $O(N^2)$  and  $O(N)$  loops in order to simplify transformed code, while the automatic tools have processed them (this difference between fully automated and semi-automated parallelizing tools is noted also in [49]). However, the difference in performance from these loops is negligible, at least for the range of matrix sizes that we have measured. It may become significant for small matrix sizes ( $N < 100$ ), but in such cases parallelization is not efficient and the sequential program performs better anyway.
2. In the `UpdateElements` operator, the auto-parallelizing tools have parallelized the inner loop, but not the outer loop. Parallelizing inner loop is less efficient because of the synchronization overhead, as already discussed in Section 4.2. In fact, this decision accounts for the difference between the performance of the auto-parallelized COMP and P4A versions, and the PAR1 version obtained using our approach (about 1,2X performance improvement). This difference illustrates the tendency of the fully automated tool to make more conservative decisions, e.g. to ignore the loop if the dependency analysis hints that there may be a dependency between loop iterations, rather than try to use more sophisticated analysis to prove that there is in fact no dependency.
3. The `FindMaxElement` operator was not parallelized by the auto-parallelizing tools. This accounts for the difference between the PAR1 and the PAR2 versions,

which is quite significant (about 2X performance improvement). Such difference illustrates that the selected auto-parallelizing tools were unable to perform more complex transformations, changing not only loop indices, but also loop bodies.

The biggest advantage of the auto-parallelizing tools compared to our approach is the full automation of the transformation process. The only input required from the user (apart from the sequential source code) is the choice of the parameters. In our approach, the user has to provide the following additional input: what loops should be parallelized, what transformations should be used, and if the desired transformation is not available, it should be described using Termware system.

However, as was demonstrated in Section 4, the loop selection can be sufficiently automated using the combination of static analysis with rewriting rules and dynamic analysis with a profiler. As for implementing transformations, the high-level nature of the algebraic program models and the declarative style of the Termware language simplify creation of new transformations. Also, as our research progresses, we plan to build a library of reusable transformations, so that the user can either apply an existing transformation “as is”, or use it as a starting point to create a new transformation.

## 6 Related Work

There has been an extensive research in the area of parallelizing existing sequential Fortran code, in particular for multicore architectures. Historically, the Fortran language was used to solve complex computational problems in scientific and engineering domain. As such problems benefit the most from the parallelization, it is natural that emerging parallel systems supported Fortran and provided the means to express parallelism in this language [6]. As the technology matured, a standardization of different vendor-specific extensions became an important topic [27]. Also with the increase in program size and complexity, the means for automating parallel programming were developed [26]. Very important for the Fortran programmers is dealing with the legacy code [48], as a huge amount of code has been created during more than 50 years of Fortran history, most of it no longer supported by original developers, but still valuable to the research community. Recently, with the appearance of the new parallel platforms, such as multicore processors [9], GPUs [18] and Cell processors [24], programming such devices using Fortran has been studied [5].

The Fortran parallelization solutions can be divided, according to their level of automation, into the following groups:

1. Manual parallelization. These approaches provide the means for the user to explicitly describe parallelism.
2. Semi-automated parallelization. Some parallelization steps are automated, still non-trivial input from the user is required. Our approach belongs to this group.
3. Fully automated parallelization. The tool requires only source code and possibly some configuration as an input, and produces a parallel program without user intervention.

Manual parallelization approaches provide the programming model to divide a program into parallel tasks, coordinate execution of these tasks and data exchange between them. Such model can be implemented in form of a parallel library, such as the widely used for distributed memory systems MPI library [47]. For the multicore platform, there are parallel libraries such as Intel TBB [44] for C++ or TPL [31] for .NET languages. Domain-specific and highly optimized libraries, e.g. LAPACK [2] and ATLAS [52] for linear algebra, are widely used to implement specific functions.

More popular in Fortran community is describing parallelism using language extensions. For shared memory systems, the most used programming model is provided by OpenMP [11]. Recently, similar language extensions have been proposed for distributed memory systems – XcalableMP [35] and for GPU programming – HMPP [14]. Previously developed Fortran parallel extensions include HPF [27], Co-Array Fortran [36] and others. Using language extensions simplifies transforming a sequential source code into a parallel code: only small regions of code have to be modified. Also they preserve familiar language style, therefore simplifying the understanding of a parallel code.

Yet another approach is using new, specially constructed parallel languages, such as X10 [46], Chapel [10], Fortress [1]. Such languages were specifically designed for parallel programming; therefore they provide powerful features for creating new programs. However, using such languages with legacy code requires rewriting large portions of code, which is usually not possible.

In general, manual parallelization approaches allow obtaining the most efficient code, but require significant effort from the programmer in order to learn new programming model and implement it correctly and efficiently.

Semi-automated parallelization approaches try to reduce the effort required from the programmer by automating some steps of the parallelization, yet retain the high efficiency of the obtained code by delegating the most significant decisions to the programmer. An important research direction is interactive parallelization, represented by such tools as SUIF Explorer [33], PIPS [22], iPat/OMP [23], ParaScope editor [28], HPFIT [8], Paralax [50] and others. These tools include the capabilities of auto-parallelizing compilers, such as dependency analysis, detection of loops for parallelization, code transformations. They also provide a way for the programmer to influence the parallelization process, using graphical user interface (SUIF Explorer and others), annotations (Paralax) or scripting (PIPS).

An interesting approach proposed by Tournavitis et al. [49] combines static analysis of data dependencies, dynamic analysis of runtime profiles to detect the missed parallelization opportunities, and machine learning to specialize the parallel program for concrete execution platforms. Our approach is similar, although more limited in scope – we do not (yet) use the machine learning techniques, and our system currently supports fewer transformations. The difference of our approach from [49] and other interactive parallelizing solutions lies in the usage of rewriting rules system as an implementation language for the transformations, which significantly simplifies the addition of the new transformations.

Another research area related to semi-automated parallelization is refactoring of Fortran code [38], i.e. code transformations that do not change the behavior of the

program, but improve the code structure. The tool implementing source-to-source transformations can be used both for refactoring and for parallelization [32], [42]. The popular refactoring tools for the Fortran language include Photran [39] and ROSE [43]. As reported in [48], using refactoring tools before parallelization improves the efficiency of the parallelizers, as static analysis benefits from more clear and understandable code structure. Notice that such tools operate on AST level, while our approach uses higher-level and more concise algebraic models of source code. The capabilities of our tools should be sufficient to implement refactoring transformations, although we haven't attempted it yet. Refactoring could be useful for our legacy code program (electron density) – we haven't performed it as we tried to stay as close as possible to the original source code that is well known and understood by existing users of the program.

The third group of approaches is fully automated parallelization, as exemplified by the parallelizing compilers [26]. These approaches try to relieve the programmer from parallelizing sequential code by detecting and parallelizing the suitable code fragments (mostly the loops without data dependencies between iterations). The popular recent parallelizing compilers for the Fortran language include the commercial Intel Fortran Compiler [7] and Portland Group (PGI) compiler [41], and the open-source OpenUH [21] and Omni OpenMP [30] compilers. Significant previously developed compilers include Rice Fortran D compiler [19] and Vienna Fortran Compiler [12].

Some source-to-source transformation tools also work in a fully automated manner. The recent Par4All tool [51] provides fully automated interface to PIPS [22] and transforms sequential C and Fortran code by inserting OpenMP directives. By using this tool, the programmer can quickly transform existing code without learning PIPS transformations and scripting. Many research parallelizing compilers are also implemented as source-to-source transformations, including Omni OpenMP [30] and Rice Fortran D compilers [19]. Source-to-source transformations allow studying (and possibly modifying) the compiler output in a more convenient way, compared to the parallelization reports.

While the fully automated tools greatly improve the programmer's productivity, they often produce inefficient code [48], [49]. Therefore, using semi-automated approaches is more practical, at least for the current state of research.

There is also an extensive research of various methods and techniques that can be used in parallelization tools, either fully or semi-automated. This research area includes static analysis [25], [40], dynamic analysis [20], [37], [45], performance analysis [34], auto-tuning [13], [52], machine learning [49] etc. Our system currently does not use such advanced approaches, which limits its applicability, especially when dealing with complex legacy code. In future, we plan to include such approaches, both to improve the capabilities of our tool and to validate the possibility of implementing advanced techniques using rewriting rules system.

## 7 Conclusion

In this chapter we have described our approach for parallelizing Fortran programs by applying formalized program transformations to existing sequential Fortran code.

Using rewriting rules technique automates application of the transformations and prevents mistakes that can appear when applying changes to source code manually. High-level algebraic models simplify understanding of legacy programs and their transformations, and enable transformation on different levels of abstraction. We have applied our approach both to the simple program implementing Gauss elimination algorithm and to the real-world quantum chemistry problem (calculating electron density). Performance measurements demonstrate significant speedup for both programs. We also compare our approach to automated parallelizing solutions, demonstrating the benefits of semi-automated parallelization.

Further research directions include development of additional transformations for the shared memory systems, as well as using the same approach for transforming legacy Fortran applications to target distributed memory systems and GPUs. Our future plans also include extension to Grid and cloud-based platforms. Also we are planning to improve support for large and complex Fortran programs, in particular automate selection of most suitable place of application for transformations.

## References

1. Allen, E., et al.: The Fortress language specification version 1.0. Sun Microsystems (2006)
2. Anderson, E., et al.: LAPACK Users' guide, 3rd edn. Society for Industrial and Applied Mathematics, Philadelphia (1999)
3. Andon, P., Doroshenko, A., Zhereb, K.: Programming high-performance parallel computations: formal models and graphics processing units. *Cybernetics and Systems Analysis* 47(4), 659–668 (2011)
4. Andon P.I., Doroshenko, A.Y., Tseitlin, G.O., Yatsenko O.A.: Algebra-algorithmic models and methods of parallel programming. *Academperiodika, Kiev* (2007) (in Russian)
5. Asanovic, K., et al.: A view of the parallel computing landscape. *Communications of the ACM* 52(10), 56–67 (2009)
6. Backus, J.: The history of FORTRAN I, II, and III. *SIGPLAN Not.* 13(8), 165–180 (1978)
7. Blair-Chappell, S., Stokes, A.: *Parallel Programming with Intel Parallel Studio XE*. John Wiley & Sons, Hoboken (2012)
8. Brandes, T., et al.: HPFIT: a set of integrated tools for the parallelization of applications using High Performance Fortran. PART I: HPFIT and the TransTOOL environment. *Parallel Comput.* 23(1-2), 71–87 (1997)
9. Buttari, A., Dongarra, J., Kurzak, J., Langou, J., Luszczek, P., Tomov, S.: The Impact of Multicore on Math Software. In: Kågström, B., Elmroth, E., Dongarra, J., Waśniewski, J. (eds.) *PARA 2006*. LNCS, vol. 4699, pp. 1–10. Springer, Heidelberg (2007)
10. Chamberlain, B.L., Callahan, D., Zima, H.P.: Parallel programmability and the chapel language. *International Journal of High Performance Computing Applications* 21(3), 291–312 (2007)
11. Chapman, B., Jost, G., Van Der Pas, R.: *Using OpenMP: portable shared memory parallel programming*. The MIT Press, Cambridge (2007)
12. Chapman, B., Mehrotra, P., Zima, H.: Programming in Vienna Fortran. *Sci. Program.* 1(1), 31–50 (1992)
13. Datta, K., et al.: Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In: *ACM/IEEE Conference on Supercomputing, SC 2008*, pp. 1–12. IEEE Press, Piscataway (2008)

14. Dolbeau, R., Bihan, S., Bodin, F.: HMPP: A hybrid multi-core parallel programming environment. Technical report, CAPS Enterprise (2007)
15. Doroshenko, A., Shevchenko, R.: A Rewriting Framework for Rule-Based Programming Dynamic Applications. *Fundamenta Informaticae* 72(1), 95–108 (2006)
16. Doroshenko, A., Zhreb, K., Yatsenko, O.: Formal Facilities for Designing Efficient GPU Programs. In: *International Conference on Concurrency Specification and Programming (CS&P 2010)*, Bornicke, pp. 142–153 (2010)
17. Doroshenko, A.Y., Zhreb, K.A., Tyrchak, Y.M., Khatniuk, A.O.: Creating Efficient Parallel Programs in Fortran Using Rewriting Rules Technique. In: *International Conference on High-Performance Computations (HPC-UA 2011)*, Kyiv, pp. 76–83 (2011) (in Russian)
18. Henderson, T., et al.: Experience Applying Fortran GPU Compilers to Numerical Weather Prediction. In: *2011 Symposium on Application Accelerators in High-Performance Computing*, pp. 34–41. IEEE Computer Society, Washington (2011)
19. Hiranandani, S., Kennedy, K., Tseng, C.-W.: Compiling Fortran D for MIMD distributed-memory machines. *Commun. ACM.* 35(8), 66–80 (1992)
20. Hoefler, T., Schneider, T.: Communication-centric optimizations by dynamically detecting collective operations. In: *17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 305–306. ACM, New York (2012)
21. Huang, L., Eachempati, D., Hervey, M.W., Chapman, B.: Exploiting global optimizations for OpenMP programs in the OpenUH compiler. *SIGPLAN Not.* 44(4), 289–290 (2009)
22. Irigoin, F., Jouvelot, P., Triolet, R.: Semantical interprocedural parallelization: an overview of the PIPS project. In: *5th International Conference on Supercomputing*, pp. 244–251. ACM, New York (1991)
23. Ishihara, M., Honda, H.: Development and implementation of an interactive parallelization assistance tool for OpenMP: iPat/OMP. *IEICE Transactions on Information and Systems* 89(2), 399–407 (2006)
24. Keir, P., Cockshott, P.W., Richards, A.: Mainstream Parallel Array Programming on Cell. In: Alexander, M., D’Ambra, P., Belloum, A., Bosilca, G., Cannataro, M., Danelutto, M., Di Martino, B., Gerndt, M., Jeannot, E., Namyst, R., Roman, J., Scott, S.L., Traff, J.L., Vallée, G., Weidendorfer, J. (eds.) *Euro-Par 2011, Part I. LNCS*, vol. 7155, pp. 260–269. Springer, Heidelberg (2012)
25. Kejariwal, A., et al.: Cache-aware partitioning of multi-dimensional iteration spaces. In: *SYSTOR 2009: The Israeli Experimental Systems Conference*, pp. 15:1–15:12. ACM, New York (2009)
26. Kennedy, K., Allen, J.R.: *Optimizing compilers for modern architectures: a dependence-based approach*. Morgan Kaufmann, San Francisco (2002)
27. Kennedy, K., Koebel, C., Zima, H.: The rise and fall of High Performance Fortran. In: *3rd ACM SIGPLAN Conference on History of Programming Languages (HOPL III)*, pp. 7:1–7:22. ACM, New York (2007)
28. Kennedy, K., McKinley, K.S., Tseng, C.W.: *Interactive parallel programming using the ParaScope Editor*. *IEEE Transactions on Parallel and Distributed Systems* 2(3), 329–341 (1991)
29. Khavryuchenko, V.D., et al.: Quantum chemical study of polyaromatic hydrocarbons in high multiplicity states. *International Journal of Modern Physics B* 21(26), 4507–4515 (2007)
30. Kusano, K., Satoh, S., Sato, M.: Performance Evaluation of the Omni OpenMP Compiler. In: Valero, M., Joe, K., Kitsuregawa, M., Tanaka, H. (eds.) *ISHPC 2000. LNCS*, vol. 1940, pp. 403–414. Springer, Heidelberg (2000)

31. Leijen, D., Schulte, W., Burckhardt, S.: The design of a task parallel library. In: 24th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA 2009), pp. 227–242. ACM, New York (2009)
32. Liao, C., Quinlan, D.J., Panas, T., de Supinski, B.R.: A ROSE-Based OpenMP 3.0 Research Compiler Supporting Multiple Runtime Libraries. In: Sato, M., Hanawa, T., Müller, M.S., Chapman, B.M., de Supinski, B.R. (eds.) IWOMP 2010. LNCS, vol. 6132, pp. 15–28. Springer, Heidelberg (2010)
33. Liao, S.-W., et al.: SUIF Explorer: an interactive and interprocedural parallelizer. In: 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 37–48. ACM, New York (1999)
34. Liu, X., et al.: Automatic performance debugging of SPMD-style parallel programs. *Journal of Parallel and Distributed Computing* 71(7), 925–937 (2011)
35. Nakao, M., Lee, J., Boku, T., Sato, M.: Productivity and Performance of Global-View Programming with XcalableMP PGAS Language. In: 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012), pp. 402–409. IEEE Computer Society, Washington (2012)
36. Numrich, R.W., Reid, J.: Co-array Fortran for parallel programming. *ACM SIGPLAN Fortran Forum* 17(2), 1–31 (1998)
37. Oancea, C.E., Rauchwerger, L.: Logical inference techniques for loop parallelization. In: 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 509–520. ACM, New York (2012)
38. Overbey, J., Xanthos, S., Johnson, R., Foote, B.: Refactorings for Fortran and high-performance computing. In: Proceedings of the Second International Workshop on Software Engineering for High Performance Computing System Applications, pp. 37–39. ACM, New York (2005)
39. Overbey, J.L., Fotzler, M.J., Kasza, A.J., Johnson, R.E.: A collection of refactoring specifications for Fortran 95. *SIGPLAN Fortran Forum* 29(3), 11–25 (2010)
40. Paek, Y., Hoeflinger, J., Padua, D.: Efficient and precise array access analysis. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 24(1), 65–109 (2002)
41. Portland Group PGI Compiler, <http://www.pgroup.com/>
42. Preissl, R., et al.: Transforming MPI source code based on communication patterns. *Future Gener. Comput. Syst.* 26(1), 147–154 (2010)
43. Quinlan, D.J.: ROSE compiler infrastructure, <http://rosecompiler.org/>
44. Reinders, J.: Intel Threading Building Blocks: outfitting C++ for multi-core processor parallelism. O’Reilly Media, Inc., Sebastopol (2007)
45. Rus, S., Pennings, M., Rauchwerger, L.: Sensitivity analysis for automatic parallelization on multi-cores. In: 21st Annual International Conference on Supercomputing (ICS 2007), pp. 263–273. ACM, New York (2007)
46. Saraswat, V.A., Sarkar, V., von Praun, C.: X10: concurrent programming for modern architectures. In: 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, p. 271. ACM, New York (2007)
47. Snir, M., et al.: MPI: The complete reference. MIT Press, Cambridge (1995)
48. Tinetti, F.G., Méndez, M.: Fortran Legacy software: source code update and possible parallelisation issues. *SIGPLAN Fortran Forum* 31(1), 5–22 (2012)
49. Tournavitis, G., Wang, Z., Franke, B., O’Boyle, M.F.P.: Towards a holistic approach to auto-parallelization: integrating profile-driven parallelism detection and machine-learning based mapping. *ACM SIGPLAN Notices* 44(6), 177–187 (2009)

50. Vandierendonck, H., Rul, S., De Bosschere, K.: The Parallax infrastructure: automatic parallelization with a helping hand. In: 19th International Conference on Parallel Architectures and Compilation Techniques, pp. 389–400. ACM, New York (2010)
51. Ventroux, N., et al.: SESAM/Par4All: a tool for joint exploration of MPSoC architectures and dynamic dataflow code generation. In: Proceedings of the 2012 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools, pp. 9–16. ACM, New York (2012)
52. Whaley, R.C., Dongarra, J.J.: Automatically tuned linear algebra software. In: 1998 ACM/IEEE Conference on Supercomputing, pp. 1–27. IEEE Computer Society, Washington (1998)



# University as a Corporation Which Serves Educational Interests

Alexander Spivakovsky, Lyudmila Alferova, and Eugene Alferov

Kherson State University, 27, 40 Rokhiv Zhovtnya St., Kherson, 73000, Ukraine  
{Spivakovsky, alferov\_jk}@ksu.ks.ua, kuznetsova.mila@gmail.com

**Abstract.** This chapter focuses on analyzing the structure of a university as a corporation – a complex organizational and functional mechanism which serves educational interests. It suggests that looking at a university as a specific business corporation helps better understand, more accurately plan, and seamlessly implement the transformation of a university towards better meeting the requirements of academic reform and the society at large. It is demonstrated that managing a university as a corporation is more complex compared to conventional business because of bigger diversity of different aspects and facets to be regarded in a balanced and harmonized way. A conclusion is drawn that an Information Technology and Analytics infrastructure is imperative as an instrument for managing a university at all levels. Finally the evidence is given of a successful trial of this management model and approach in real world settings at Kherson State University during the last several years – in particular for training professionals in our Computer Science and IT programme.

**Keywords:** university, corporation, IT infrastructure, management, labor market, academic reform competence, education process.

## 1 Introduction

University management today faces several difficult issues in transforming their organizations towards being more effective, efficient, and competitive. One of important instruments in optimizing the performance of a university is Information Technologies (IT). Indeed, an IT infrastructure is a backbone which shapes the way several important questions in strategic and operational management are answered. Therefore the result of a proper deployment of IT components, implementation and proper configuration of an integrated IT infrastructure for a university needs to be regarded as one of the top priority assets influencing the performance of the whole organization. Under this facet university management needs to make several important decisions on: the configuration of investments into strategic development, IT in particular; specification, unification, harmonization, and deployment of business processes in management at all levels; a proper superimposition of those business processes on to the IT infrastructure; use of IT for making the processes and results of management activity more transparent to the employees, students, and society; and,

last but not least, introducing appropriate incentives to managers at all levels for lowering their acceptance barriers with respect to new IT, working patterns, and style.

Resolving all these important issues in a harmonized and balanced way allows expecting a considerable increase in management performance – with positive implications to improvement in university competitiveness. For that, regarding a university as a corporation with specific inputs, objectives, and outputs may be helpful. If so, the methods, patterns, and best practices of corporate management may be tested at a university, hence adopting at academia better flexibility and adaptability of business organizations to market and economy changes.

The remainder of this chapter is structured as follows. Section 2 presents our motivation for the reported research in more detail and defines the focus – IT management. The related work is briefly reviewed in Section 3. Section 4 presents our main contribution – the model of a university as a corporation which serves educational interests. Section 5 presents our results in validating this model by applying it to IT management at Kherson State University (KSU) in a real world case study. Our conclusions are drawn and plans for the future work outlined in Section 6.

## **2 Motivation**

In this chapter we explore the idea of adopting corporate management style and pattern at a University and elaborate it in a form of a model. A university in our approach to shape out its management is regarded as a corporation which serves educational interests. The focus on this specificity straightforwardly allows us to choose appropriate inputs, objectives, and outputs for management processes.

A university, very similarly to a corporation, is a big and complex organization with many important facets in management: human resources; finance; materials and procurement, teaching; research, development, and intellectual property; student government; public relations; etc. Successful and productive management of all these diverse aspects of university life is heavily based on collecting, processing, analyzing, and communicating information. Therefore, IT and information management needs to be regarded as a first priority activity in this broad spectrum. This premise motivated us to choose IT management as the focus of our research.

One of important strategic level corporate management tasks is to develop a pathway for a university to maintain or even increase competitiveness on the market. The same is fairly true for universities today – the environment is the market of educational services. The complications at this market are its volatility and a tendency of toughening the requirements to university graduates in terms of the quality of their professional capabilities. These specificities imply that a well-managed university behaves like a business corporation – is adaptive to emerging loci of influence, redistribution of those influences. Adaptation mechanisms are incorporation of external units or bodies in the university structure, inclusion of these new units in decision making processes, adoption and adaption of new approaches, mechanisms, workflows, and policies for better decision making. For example a tendency today is involving students and staff in decision making processes as first class citizen with capability and authority proportional to university management. These changes

factually trigger the transformation of the behaviors of different groups of people contracted to a university to a corporate behavior.

In the settings circumscribed above the role of IT managements receives substantial importance. Effective and efficient IT management allows quickly and adequately implement required changes – both in management strategies and operations – with substantially less effort and in shorter time. So the overall performance of university management increases and the organization becomes more responsive and adaptable to environmental changes reflecting the evolution of societal needs.

Consequently, perceiving a university as a corporation which serves educational interests of its students allows to shift the emphasis of university educational service from traditional communication of knowledge from a professor to his students to establishing an educational environment or infrastructure. This infrastructure provides much richer opportunities both for a tutor and a student. For a tutor it facilitates creating all the necessary educational information resources and substantially extended interface for communication. For a student it guarantees right-on-time delivery of personalized educational material in the contexts of different disciplines and in different modalities. Straightforwardly, Information and Communication Technology (ICT) infrastructure and tools play a very important role in that as an instrumental basis of the outlined educational infrastructure that facilitates better satisfying intellectual requirements of all involved categories of knowledge workers.

Our extensive analysis of the published best practices of university management unfortunately did not reveal reliable and robust methods for measuring effectiveness and efficiency (i.e. performance) of IT management in academia. To the best of our knowledge, the influence of introducing performance measurement practice on the improvement of IT support to the abovementioned facets of academic activities – in particular organizational and process optimization, strategy development, decision making, and ultimately, the professional capabilities of the graduates. Therefore we hypothesized that adopting a corporate management model (where all the required metrics, mechanisms, and best practices are in place) to university IT management will provide a desired outcome. We believed that such a hypothesis may be formulated also because universities cooperate with industry in many aspects and adapt this interface to industrial standard. One of relevant examples is involvement of business leaders in creating and developing new organizational capabilities at academia for better integration into the labor market and prompter reaction to market requirement to the major academic product – their graduates looking for employment and carrier building in industries.

Based on that, our grand objective in this research was to investigate how the introduction of an appropriate and comprehensive IT infrastructure at a university will increase the performance of university management and, consequently, the quality of the academic product provided to the labor market. Methodologically we:

- Regarded a university as a specific corporation which serves educational interests of students. We also supposed that their interests are coherent to the requirements of the labor market and evolve in line with these requirements. Hence, educational interests are in fact the reflection of what is expected from university graduates in terms of their professional competencies. For further implementation of this view the model of a university as a corporation has been elaborated.

- Used the corporation model for proof-testing it in a real world case study focused on analyzing the performance of IT management at Kherson State University. The infrastructure and preconditions for this case study have been developed in our previous research and development activities [1].

### 3 Related Work

Our research reported in this chapter is based on several interrelated aspects relevant to managing and productively using IT in business. The first building block has been taken from the review of the best practices in business by the Center for Information Systems Research of the Sloan School of Management at Massachusetts Institute of Technology. Their review is based on the analysis of using information technology (IT) practices in business by more than 250 companies [2]. Apart of very useful and valid factual material on the use of IT, one valid side effect of this report is the confirmation of the importance of the role of IT and IT infrastructures in modern enterprises. Notably, it has been shown in [2] effective use of IT management for implementing strategies leads to success of institution. In particular companies that were successful in their IT management had more than 20% higher profits than similar companies with the same strategy but with ineffective IT Governance [2-4]. Another confirmation of the importance of IT in performance management is given by Melchert and Winter [5] who also define performance “*as valued contribution to reach the goals of an organization.*” We follow this definition of performance in our research as it straightforwardly connects means to ends.

In the vast body of relevant publications in business performance management domain, the papers focusing on modeling performance were of particular relevance for our work as we were seeking for proper metrics and methods of measuring performance of IT management. A good review of performance management practices, perspectives, and measurement models is given in [6, 7] resulting in the proposal of a Performance Ontology [7].

According to the performance management literature (e.g., [8]), the most popular frameworks for performance measurement and management (PMM) are Balanced Scorecards, Business Excellence Model, Performance Prism. Those frameworks are based on the introduction of a set of interrelated performance indicators.

Based on the analysis of the related publications, it may be stated in a summary that IT management serves as a strategic driver for each successful organization today. The competitive advantage of an organization is determined not just by the presence of effective mechanisms for collecting and processing various information resources and providing them to their customers (students and their employers in case of a university), but also the ability to transform these resources into their corporate actionable knowledge.

### 4 University as a Corporation

A business corporation (further – a corporation) is a legal business entity, formed as the union of individual founders and functioning independently of them. A

corporation has economic objectives guiding its business activity [9]. A big corporation normally generates influences on the society, hence needs to elaborate its social orientation and responsibility [10]. As a legal entity, a corporation has to act according to the normative environment – i.e. comply with relevant normative rules and laws. It has to respect consumer rights and interests, take care of its employees. All these aspects have to be treated harmonically and lay the basics of the corporate culture with implications to specifying the code of employee behavior, corporate traditions, etc.

Corporations sell products or services. In a market oriented economy corporate income strongly depends on the quality of product which is one of the major indicators of competitiveness. For a university, regarded as a corporation with a specific focus on education, the markets are: (i) labor market; and (ii) the market of educational and related services. With respect to the labor market the product of a university is their graduate. Product quality is therefore the quality of the professional capabilities of university graduates. Hence, a university as a corporation has to set the assurance of providing quality professional capabilities to their graduates as one of its primary objectives. At the market of educational services, a university sells not their graduates but teaching and learning services to their students and, perhaps, to other academic entities. So, one more primary goal for a university is to assure that the quality of their teaching materials is competitively high.

Perhaps because of the outlined similarities between modern academic organizations and business companies an opinion that a university and, at a higher level, an academic system in their structure and action resemble a corporate system very much, also in public opinion. Moreover, as a university is an organization providing substantial societal impact and in a large proportion use public funding, taxpayers identify themselves as, so to say, university stockholders with an influence on forming management structure. In fact a university president, though elected internally by a university council, is in fact assigned to this position based also on the opinion of the regional community – the stockholders. If so, the stakeholders will be more willing to become also the customers of the university corporation – as students and their parents who pay for educational services in some way. Hence, market relationships and their specificity needs to be taken into account in all aspects of university management.

As in any other type of business corporations, business information at a university is a specific type of information tightly connected with and influencing decision making at all levels of management structure. Business information covers the internals of a University (human resources, student contingent, educational programmes, budget, funding sources, etc.) and external environment of a university (e.g. government order, situation on labor market, economic indicators like an average salary in industry, tariffs for communal services, state tax policies and so on). Hence, collection and timely processing of all the relevant facets of business information is of a high priority for university management as it influences noticeably the pragmatic decisions they make and further implement in the corporation. It is also important that only valid or reliable information is taken into account that depicts the internals and externals in a fair and unbiased manner. Basing management practices on objective facts will substantially increase the effectiveness and efficiency of

management, compared to a widely spread practice in which strategic decisions are often made based on personal subjective attitudes and perceptions, traditional business patterns, or intuition of senior university management. Possible inconsistencies between objective business information and subjective business knowledge have therefore be resolved using known methodologies and best practices of knowledge elicitation and acquisition (e.g. [11]).

A well-known fact is that knowledge based businesses and corporations face information overload (e.g. [12]) which implies knowledge acquisition bottleneck (e.g. [13]). The trend is that these problems will be sharpened in time as the volumes and velocities of incoming information increase. Therefore the importance is growing of using an IT infrastructure as a corporate instrument that supports timely, reliable, and effective information processing, knowledge acquisition, communication, decision making. Introducing IT instruments in all relevant business processes and ground them in a coherent IT infrastructure increases the performance of business information and knowledge processing and, consequently, university management – improving timeliness, objectivity, and completeness of knowledge acquisition and use for decision making which now is able to account all the related information facets and aspects. For example the corrective actions in a student enrollment company may be proposed and approved based on monitoring the dynamics of enrollment applications. Statistical information about paying education fees by students may trigger the expulsion of those students who break their contractual obligations, but also help predict the financial balance of a university.

Our experience in implementing and deploying such an IT infrastructure at Kherson State University (KSU) convinces that a properly configured IT toolset provides qualitatively better possibilities to have access to and process information sources inside a university. Using IT in our Information Analytical Systems (IAS) allows effectively collect up-to-date information about the key aspects of organization activities. For example at our university we have developed and deployed the following modules of the university IAS [14] that help significantly increase management performance:

- Planning and financial accounting [14] comprising human resources and students;
- Financial bookkeeping [14, 15] comprising debits, credits, and transfers for personnel, students and contractual obligations;
- Materials accounting [14, 15];
- University entrants [14, 15];
- Academic accounting [14, 15].

The IAS as an infrastructure facilitates providing reliable and relevant information in particular by maintaining the system of access rights and personified services for all involved management roles and serves as an information integrator. Integration of information provided by IAS means that all authorized users acquire all available and relevant business information in the form of automatically generated reports. These reports are further used to fulfill functional duties. IAS also helps generating reports for external bodies like governmental agencies: the Pension Fund, Tax Administration, National Statistical Office, Ministry of Education and Science.

Universities as corporations active on the market of education services use IT as an instrument for obtaining the following impacts:

- Extending the contingent of students as customers, expanding the spectrum of educational services
- Reducing uncertainty and lowering risk while implementing strategic management decisions at operational level. For example, opening of new educational programmes, creation of new business units, expansion of material base that requires constant renewal, etc.
- Positively influencing certain aspects of society and regional community
- Improving university corporate performance through monitoring and evaluation of internal performance indicators such as quality of teaching and learning, quality of personnel, availability of computers and information resources, library funds, etc.
- Improving university corporate competitiveness through monitoring and evaluation of external performance indicators such as the proportion of successfully employed graduates, business testimonials with respect to the professional capabilities of graduates, participation in international programmes, etc.

It has to be noted that solutions of university management and education process issues are not only be sought within economic and legislative facets. The mentality of stakeholders is also a very important aspect that may be a source of serious blocking factors for academic reform. For ensuring that the reform is performed in a proper and coordinated way the code of desirable behavior [2, 15] has to be changed in a way to stimulate rational acceptance of IT in management practice. Both material and immaterial incentives may be used for that. .

In the context of globalization university is gaining new features. Providing, supporting and dissemination of culture – the main mission of a university in the XIX - XX centuries – fades into the background. Other roles come to the forefront of university management activity: adaptation to the current socio-economic and political changes and cooperation between a university and the society. At the National level cooperation partners comprise the State and its governmental bodies, while at the Global level the partners are academic peers and international organizations. Anyhow, these changes in management priorities still have to be properly focused on satisfying societal educational interests.

A Business company usually focuses on profitability as a major indicator of competitiveness. In contrast to business in general, the major focus of university business is on continuous commitment to provide quality education to its students – which subsumes profitability indirectly and in a broader, not only material sense. Due to this broadness the business model of a university corporation is naturally richer and comprises more aspects. The most important are: attractiveness for students; richness of the portfolio of educational programmes supported by the appropriate institute and departmental structure; the disciplines and their curricula; teaching and didactical materials; diversification of the funding resources; and, last but not least, the IT infrastructure and resources required to manage university business. These factors must be always considered in a harmonic balance to achieve the goals of the university.

For achieving a proper balance on all the scales an effective and efficient management system needs to be implemented with a focused use of relevant IT

instruments. In the context of the major objective – improving teaching and learning quality – the performance of educational process needs to be addressed in a close cooperation between teaching departments who manage the process and IT management who provides and maintains the tooling for teaching and learning.

**Table 1.** The specificity of increasing performance at university corporations

<b>Factors relevant to improving performance in a business corporation</b>	<b>The specificity of related indicators in a university corporation</b>
Increase of income	Diversifying the portfolio of funds by acquiring scientific and technological capabilities and resources for National and International funded projects. Harmonically combining research and teaching.
Increase of product quality	With respect to education services, v Publishing and carefully maintaining teaching and learning resources digitally using university IT infrastructure. Establishing digital communication culture among all stakeholders, comprising academic staff and students. With respect to producing graduates: ensure that the education service infrastructure is effectively and efficiently used for constantly improving professional capabilities of students throughout the whole period of study.
Improvement of staff competencies	Using integrated, corporate, personalized information and analytical system to support business processes at a University.
Modernization of equipment. Priority is given to the core production facilities.	Very similarly to a business corporation, except the core production facility is the complex of educational services based on the IT infrastructure. Hence, investment in the IT infrastructure has to be a priority one.
Modernization and optimization of technology	Again, very similarly to a business corporation, except the technology is different – didactics. For making didactics effective and efficient a proper IT support has to be at hand.



To summarize the analysis of the common features and differences of a business and university corporation several analogies could be drawn as shown in Table 1.

## **5 KSU as a Corporation Serving Computer Science and IT Students**

In this section we provide the evidence of how managing KSU as a corporation and introduction of the university IT infrastructure helped us increasing performance and better serving the interests of Computer Science and IT students.

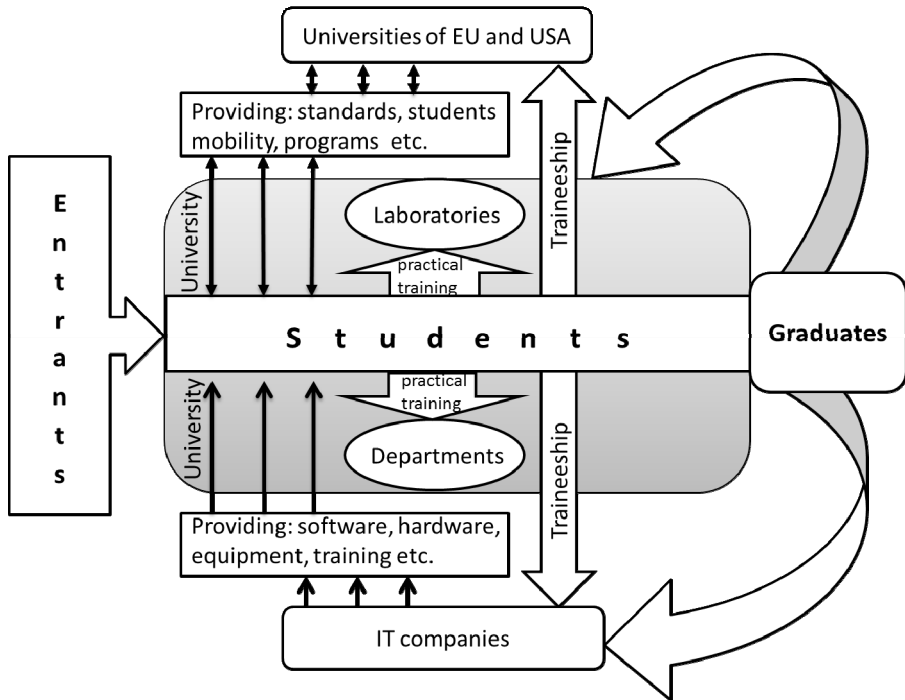
For creating the preconditions for developing highly professional specialists at KSU many resources and web services are designed, implemented, deployed, and widely used in teaching and learning processes. All these help academic staff create information and communication environment where they can share their learning materials, communicate and work together with students on developed courses. Starting from the first year of studies at the university our students have an opportunity to observe the changes and innovations in information technologies, take part in software development projects which extend and refine the IT infrastructure and IAS. So, they combine active participation in a traditional education process with obtaining practical experience of software development in an environment that is very similar to software industry.


In addition to that our senior students explore the benefits of several partnerships of KSU with several external business companies working in software production: DataArt ([www.dataart.com](http://www.dataart.com)), PostIndustria ([www.postindustria.com](http://www.postindustria.com)), Aricent ([www.aricent.com](http://www.aricent.com)) and many others. Well-formed and mutually beneficial cooperation between competent IT professionals from these industrial companies and students having their internships at the companies helps identify, reveal, and further develop the creative traits of future specialists. Student internship projects therefore yield results that are seamlessly used in industry and produce business value. The situatedness of our students in the industrial environment is also very helpful in validating the quality of the professional capabilities they have obtained at the university. Furthermore the environment motivates students well to continue their professional development.

Our training cycle that involves industry as a stakeholder is sketched out in Fig. 2. Starting from the 1st year the students in a Computer Science and IT programme are taught several fundamental and applied disciplines. Didactics for these disciplines involves the use of many teaching software tools for illustrative purposes, as simulators, for providing teaching and learning materials. The tools are managed and configured and the feedback from students is collected using the university IT infrastructure. Leading specialists from the partner IT companies are involved in the educational process as tutors for the applied disciplines.

IT management at KSU comprises several IT departments for providing software and IT services, supporting business and academic processes, technical support. Starting from 2nd year of study our Computer Science and IT students are given an opportunity to apply for a part time job in these departments. Successful applicants become a part of our IT management and development team and work in the projects

of these departments that develop, maintain, and manage information resources and services. In addition to industrial internships mentioned above our senior students are directed for traineeships or scholarships to study and practice at the universities in France, Germany, Great Britain and USA.



Legend:  – university with its information and communication infrastructure

**Fig. 1.** Computer Science and IT training cycle at KSU

The result of implementing this training cycle is very positive. Those graduates who passed it have much better chances to be employed either at industry or by a university department because: (i) their professional capabilities have been formed under direct influence of and with full account to the requirements of these employers; and (ii) they are already known to the employers.

Our academic infrastructure also creates good opportunities for those students who choose to develop an academic carrier. Students involved in research and development projects become well known to our internal and sometimes external leading scientists. So, it is much easier for them to select a scientific adviser for post-graduate studies. For fostering exchanges and further cooperation between our senior students and scientific communities KSU is active in organizing and hosting academic conferences. In ICT domain for example, KSU host ICTERI conference series (icteri.org). Our academics together with students are active in submitting their research and development results to these conferences.

Finally it may be noted that such an organization of teaching and learning in our Computer Science and IT programme not only increases the performance and refines the capabilities of knowledge workers considerably, but also creates an attractive and comfortable professional environment for students and academics.

## 6 Conclusions

In this chapter we formulated a hypothesis that looking at a university as to a business corporation may be a correct approach to improve performance in management and academic activities. We exploited many analogies between a business corporation and a university which, put together in a coherent management model, allowed us to adopt industrial best practices for performance management and improvement in academic environment. We also figured out the specificities of a university as a particular kind of a business corporation. Important ones were the objectives, the impacts on the society at large, and the specific markets where a university sells its product – educational services and graduates. Last but not least, we found out that managing a university as a corporation is more complex compared to conventional business because of bigger diversity of different aspects and facets to be regarded in a balanced and harmonized way. We drew a conclusion that an IT (and IAS) infrastructure is imperative as an instrument for managing a university at all levels.

We have also provided the evidence of a successful trial of this management model and approach at Kherson State University during the last several years – in particular for training professionals in our Computer Science and IT programme. Practical results testify that the major performance indicator for an academic programme – the proportion of graduates who are employed and continue building their professional carriers became much higher than it was before. As a conclusion we may now state as practically proven that the investment in implementing and further developing an integrated university IT infrastructure is one of the most effective and efficient kinds of investment. It increases the competitiveness of a university corporation and makes it firmly established on the markets. It facilitates impacting the stakeholders and the society in broad in a proper and constructive way.

## References

1. Spivakovsky, A., Alferova, L., Alferov, E.: Conceptualization of University Structure as a Complex Mechanism Serving Educational Interests. In: Proc. 8-th Int. Conf. ICTERI 2012, Kherson, Ukraine, June 6-10, CEUR-WS.org, vol. 848, pp. 121–127 (2012) (in Ukrainian)
2. Weill, P., Ross, J.W.: IT Governance: How Top Performers Manage IT Decision Rights for Superior Results. Harvard Business School Press (2004)
3. Albrecht, R., Pirani, J.A.: Using an IT governance structure to achieve alignment at the university of Cincinnati. ECAR Case Studies, Educause Center for Applied Research, ECAR (2004)
4. Clark, A.J.: IT governance: determining who decides. Educause Center for Applied Research (ECAR), Research Bulletins (2005)

5. Melchert, F., Winter, R.: The Enabling Role of Information Technology for Business Performance Management. In: Meredith, B., Shanks, G., Arnott, D., Carlsson, S. (eds.) *Decision Support in an Uncertain and Complex World*, Proc. 2004 IFIP Int. Conf. on Decision Support Systems (DSS 2004), Prato, Italy, pp. 535–546 (2002)
6. Matzke, W.-E.: Engineering Design Performance Management - from Alchemy to Science through ISTa (Invited Talk). In: *ISTA 2005*. LNI, vol. 63, pp. 154–179. GI (2005)
7. Ermolayev, V., Matzke, W.-E.: Towards Industrial Strength Business Performance Management. In: Mařík, V., Vyatkin, V., Colombo, A.W. (eds.) *HoloMAS 2007*. LNCS (LNAI), vol. 4659, pp. 387–400. Springer, Heidelberg (2007)
8. Collett, R.: Benchmarking IC Development Capability – What to Measure? *Fabless Forum*. *Fabless Semiconductor Association* 11(2) (2004)
9. Investopedia, <http://www.investopedia.com/terms/c/corporation.asp>
10. What is corporation, <http://ehow.in.ua/78315-shho-take-korporaciya.html> (in Ukrainian)
11. Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., van de Velde, W., Wielinga, B.: *Knowledge engineering and management. The Common KADS Methodology*. MIT Press (1999)
12. Edmunds, A., Morris, A.: *International Journal of Information Management* 20(1), 17–28 (2000), doi:S0268-4012(99)00051-1
13. Cullen, J., Bryman, A.: The Knowledge Acquisition Bottleneck: Time for Reassessment? *Expert Systems* 5(3), 216–225 (1988)
14. Belyaev, Y., Spivakovsky, A., Shedrolosev, D.: *Information analytical system of university management “University”*: Practical Aspects. Kherson State University Publishing, Kherson (2006) (in Ukrainian)
15. Spivakovsky, O.V., Fedorova, Y.B., Glushenko, O.O., Kudas, N.A.: *IT management of higher education establishments*, 3rd additional edn., Kherson (2010) (in Ukrainian)

# A Multi-channel Communication Framework

Michal Nagy

University of Jyväskylä, P.O. Box 35 (Agora), FI-40014, Jyväskylä, Finland  
michal.nagy@jyu.fi

**Abstract.** We present a modular framework for a business-to-customer communication service integrating several communication channels. Using such a service it is possible to find hidden relationships between messages and thus collect more customer-related data. The framework core is a message-conversion engine capable of converting channel-independent abstract messages into concrete messages and vice-versa. The conversion process is context-dependent. The context consists of formally described communication channel characteristics and user preferences. The framework is based on semantic technologies due to a balance between their expressive power, reasoning properties, and existence of production-quality tools. This chapter describes the multi-channel communication framework with relation to its components. Among others we discuss message conversion and channel selection.

**Keywords:** communication, ontology, context-awareness, B2C.

## 1 Introduction

The introduction of electronic media into businesses changed the way businesses communicate with their customers and between each other. A typical business has nowadays several options how it can reach its customers and how the customers can reach the business. We call these means of communication communication channels. With the increasing number of communication channels the problem of consolidation and integration becomes more complicated.

We will illustrate this problem on an example of a consumer electronics seller trying to communicate with a potential customer. The seller has a web page, Facebook page, Twitter account, phone number and an email address. These are the communication channels. The seller may start a campaign selling a new model of a TV. The seller sends a new message to all followers on Twitter and Facebook. The next day a customer asks a question about the new product through the seller's web page. The seller replies by sending an email to the email address that the customer provided in the web form. Then the customer buys the product using a form on the seller's web page. The example illustrates communication through several communication channels – social networks, web forms and email. It would be useful for the seller to understand that the customer asking the question through the web form is the same person that later bought the product. This way the seller may build up a database of all communication

happening between the business and a particular customer no matter what communication channel is being used. The seller may for example find out that the customer prefers email communication in 90% of cases and that he/she prefers to buy the products in person in the brick-and-mortar shop instead of buying them online. This may differ from another customer that prefers communication through Facebook and always buys product electronically through the electronic shop.

In order to make such a system we have to overcome several issues. In [7] we present five issues related to multi-channel communication – message conversion, communication channel selection, information element extraction, goal/purpose modeling and context modeling. In this publication we propose a framework that tries to solve these issues. The framework is mostly based on semantic technologies and utility functions. The emphasis was given to the real-world implementation.

Section 2 talks about the motivation behind this effort. Section 3 describes the framework and briefly discusses the main framework components. Section 4 defines ontologies that are used by various components of the framework. Section 5 describes engine algorithms used for message conversions and channel selection. Lastly, section 6 concludes the chapter.

## 2 Motivation

A multi-channel communication system brings several advantages to businesses. Firstly, it allows them to collect more meaningful information about their customers. Nowadays, information systems can collect a significant amount of data related to B2C communication. However, more data does not necessarily mean more information. Isolated facts become more useful once they are linked with each other given a certain context. This is the motivation of the Linked Data initiative as well [1]. This way hidden links between data can be detected.

Every communication effort initiated by the business should be rational and should be done with a goal and purpose (e.g. promoting a product with the goal of selling it). Better aimed communication increases the chances of goals being fulfilled. In order to make these rational decisions, one must be well informed. There should be a system capable of collecting information about every customer – past actions, preferences, financial profile, etc. Instead of sending a message to concrete customers, a business may decide to send a message to a certain type of customer. For example if the business tries to advertise a TV, it should advertise it one way to home cinema enthusiast, but other way to customers interested in products with low energy consumption.

There is a significant amount of business-relevant information on the Internet. The problem is however that the information is difficult or even impossible to understand by a machine [4]. The situation is changing. Technologies such as RDF (Resource Description Framework), OWL (Web Ontology Language) allow people and businesses to publish this information in a machine-readable form on the Internet. There are many publicly available RDF storages that contain useful

information (e.g. DBpedia [3], data.gov.uk portal [2], etc.). It could be beneficial to utilize this information in business' decision making process.

### 3 Multi-channel Communication Framework

#### 3.1 Framework Overview

The framework consists of two main components – knowledge base and message conversion engine. The goal of the knowledge base is to store information about five main areas – messages, communication channels, customers, commodities and business actions. All these partial knowledge bases are linked. The framework user is free to extend the knowledge base and corresponding knowledge schemas (ontologies) to fit the business needs. The message conversion engine is responsible for message conversion. It converts message received from customers to abstract message trying to extract information elements that are relevant to the business. Then it interprets them in form of actions. The framework user may specify several message templates to send message to the customers. The engine converts message templates into concrete messages that can be sent to the customer. It is also responsible for proper channel selection that may depend on customer's personal preferences and other circumstances. The framework overview is available in Figure 1.

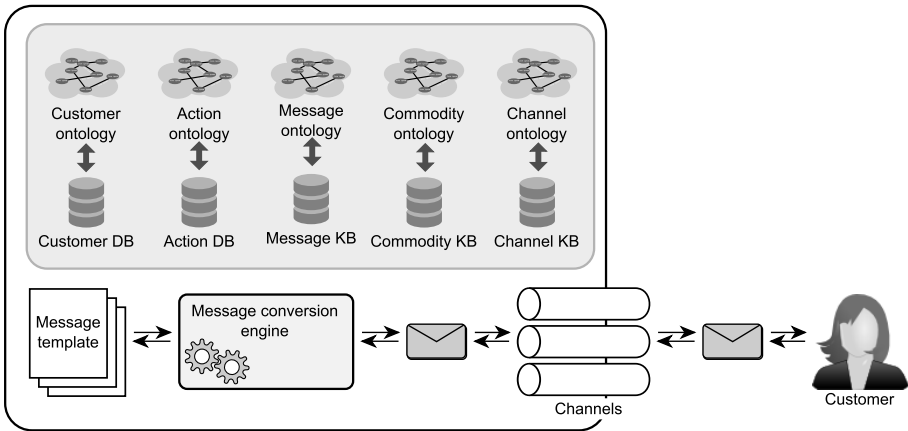


Fig. 1. Multi-channel communication framework overview

#### 3.2 Knowledge Base

The system should be capable of storing different types of information about customers, channels and business. Some of this data is relatively static (e.g. product data) and some data is dynamic (e.g. customer information). However,

we believe that a single way of representation is favorable due to the need to integrate and reason about all these kinds of data together. Reasoning will take place mostly in two cases – when a message is received and when a message is sent.

We need a way of knowledge modeling that would be expressive enough to describe relationships between heterogeneous entities such as customers, products, services, channels, etc. Moreover, the language must allow reasoning about the facts. This reasoning must be sound, so that it infers only valid facts. It should be complete as well, so that no valid fact is missing. If a valid missing was missing, it would result in a failure to send an outbound message or to properly annotate an inbound message. Lastly, this modeling framework must not be computationally too complex. As mentioned earlier, the discussed system should be usable in real-world production environment. Therefore the existence of a mature tool with abovementioned properties is important.

Based on these requirements we believe that Resource Description Framework (RDF) is the most reasonable way to model our data. It is expressive enough, proven and widely supported by a great amount of production-quality tools for modeling (e.g. Protégé [8]), storing and querying data (e.g. Jena framework [6], Sesame [12]). Also, there are many reasoners that are sound, complete and still computationally inexpensive (e.g. RacerPro [9], HerMiT [5], etc.). RDF is built on top of wide-spread standards such as XML, XML Schema, etc. It is closely related to Web Ontology Language (OWL) which is used to formally define knowledge schemas called ontologies. Thanks to ontologies and reasoners it is possible not only to conclude new facts from existing ones, but to check data consistency as well.

## 4 Ontologies

### 4.1 Overview

In this section we present several ontologies that are needed for the framework's operation. Each ontology belongs to a different namespace due to logical division. Later in the text we use prefix names to refer to specific namespaces. They are described in Table I.

Prefixes `xsd`, `owl`, `rdfs` and `rdf` represent well-known namespaces and ontologies. All the other prefixes represent framework-specific namespaces and ontologies. The commodity ontology represents all the products and services that the business is trying to sell. The message ontology represents abstract and concrete messages. The action ontology is about actions that the business or a customer can perform. The channel ontology is used to describe communication channels. Lastly, the customer ontology represents a customer schema including customer personal information, contact addresses, preferences, etc. In order to conserve space the ontologies are not described directly in OWL. For each ontology the class hierarchy is graphically depicted in a figure and the list of preferences is described in a table.

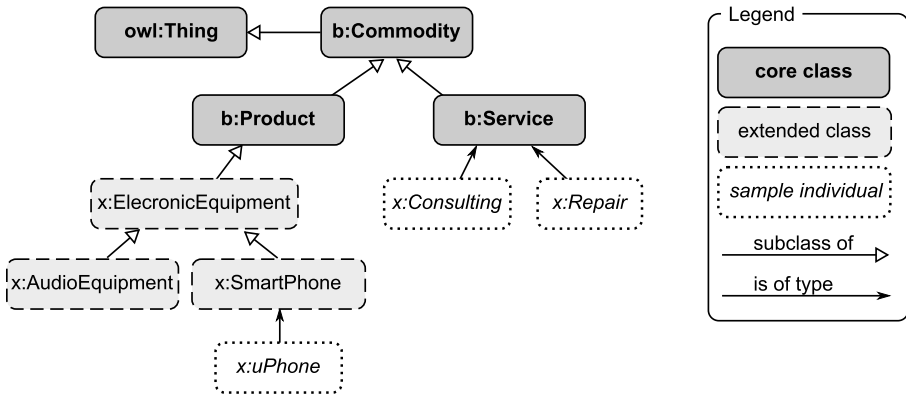


**Table 1.** Namespaces and prefixes used in this chapter

Prefix	Namespace	Ontology
b	<a href="http://cs.jyu.fi/ai/OntoGroup/mmcc/commodity#">http://cs.jyu.fi/ai/OntoGroup/mmcc/commodity#</a>	Commodity ontology
a	<a href="http://cs.jyu.fi/ai/OntoGroup/mmcc/action#">http://cs.jyu.fi/ai/OntoGroup/mmcc/action#</a>	Action ontology
m	<a href="http://cs.jyu.fi/ai/OntoGroup/mmcc/message#">http://cs.jyu.fi/ai/OntoGroup/mmcc/message#</a>	Message ontology
ch	<a href="http://cs.jyu.fi/ai/OntoGroup/mmcc/channel#">http://cs.jyu.fi/ai/OntoGroup/mmcc/channel#</a>	Channel ontology
cu	<a href="http://cs.jyu.fi/ai/OntoGroup/mmcc/customer#">http://cs.jyu.fi/ai/OntoGroup/mmcc/customer#</a>	Customer ontology
xsd	<a href="http://www.w3.org/2001/XMLSchema#">http://www.w3.org/2001/XMLSchema#</a>	XML Schema [15]
owl	<a href="http://www.w3.org/2002/07/owl#">http://www.w3.org/2002/07/owl#</a>	Web Ontology Language [14]
rdf	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#">http://www.w3.org/1999/02/22-rdf-syntax-ns#</a>	Resource Description Framework [10]
rdfs	<a href="http://www.w3.org/2000/01/rdf-schema#">http://www.w3.org/2000/01/rdf-schema#</a>	RDF Schema [11]

## 4.2 Commodity Ontology

In general the commodity ontology depends on the business domain and can comprise of a variety of goods and services. However, there is a single primitive generic ontology that can be extended based on the needs of the business. The generic commodity ontology together with an example of an extension can be seen in Figure 2. The extension is based on a scenario where the user of the framework is a small consumer electronics seller.

**Fig. 2.** Class hierarchy of a generic commodity ontology and its sample extension

The central point of the ontology is the class `b:Commodity`. A commodity represents either a product (`b:Product`) or a service (`b:Service`). Class `b:Commodity` also acts as an integration point to other ontologies, that means other framework ontologies refer to it.

### 4.3 Communication Channel Ontology

The communication channel ontology is relatively small. The user of the framework needs to extend it with communication channels he/she wants to use in his/her business. The central point of the communication channel ontology is the class `ch:ComChannel`. The class hierarchy is depicted in Figure 3.

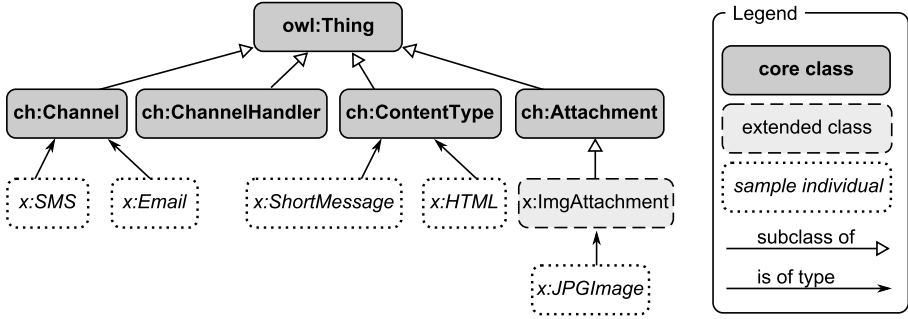


Fig. 3. Class hierarchy of a generic commodity ontology and its sample extension

Each communication channel (or just channel) is characterized by several object and datatype properties. The speed and reliability of the channel is determined by the property `ch:speed` and `ch:reliability`. Higher value represents higher speed/reliability. Property `ch:cost` represents any other considerations – e.g. financial cost. The capability to transfer attachments is expressed through `ch:attachmentFeature` object property. In general a channel might have no possibility to transfer attachments (e.g. SMS) up to the ability transfer several types of attachments (e.g. electronic mail). For the sake of simplicity we describe communication channel ontology properties in Table 2.

Table 2. Object and datatype properties of the channel ontology

URI	Min. card.	Max. card.	Domain	Range
<code>ch:speed</code>	1	1	<code>ch:ComChannel</code>	<code>xsd:float</code>
<code>ch:reliability</code>	1	1	<code>ch:ComChannel</code>	<code>xsd:float</code>
<code>ch:cost</code>	1	1	<code>ch:ComChannel</code>	<code>xsd:float</code>
<code>ch:attachmentFeature</code>	0	n	<code>ch:ComChannel</code>	<code>ch:Attachment</code>
<code>ch:registeredHandler</code>	1	1	<code>ch:ComChannel</code>	<code>ch:ChannelHandler</code>
<code>ch:acceptsContent</code>	1	n	<code>ch:ChannelHandler</code>	<code>ch:Content</code>
<code>ch:conversionFunction</code>	1	1	<code>ch:ChannelHandler</code>	<code>xsd:string</code>

In general, each channel uses different types of messages and handles them in a different way. Channel handler is a special class that represents the way messages are constructed and manipulated. It stores the list of content types that can be

transferred through the particular communication channel. This is expressed by linking the channel handler to a content instance by `ch:acceptsContent` property. The list of content types is configurable and fully depends on the framework user. There is a content conversion function associated with each handler. This function takes the content definition from the message template and converts it to a particular content suitable for this channel. Since each channel can deal with different types of content, each channel has its own handler.

#### 4.4 Action Ontology

The main goal of the multi-channel communication framework is to allow message integration among various communication channels. However, messages have no business value unless they are properly interpreted. As mentioned earlier we assume that each message is sent for a reason and that it has a purpose. Therefore we believe that each message is associated with one or more actions. A message is a piece of information that needs to be interpreted under a certain context. Usually a message has a recipient, sender, subject and body. An action is something that a message represents. In our case we are interested in business actions. Some of these actions are related to certain commodities (products or services) such as a purchase, question about a product, ordering a service or a product return. Other actions are not related to any product – e.g. contact information change, change of preferences, etc.

We use class `m:Message` for representation of messages, class `a:Action` for representation of actions and class `b:Commodity` for the representation of commodities. The relationship between these classes is depicted in Figure 4. Object property `m:represents` represents a connection between a message and action(s). There are two subclasses of `a:Action`. Subclass `a:GeneralAction` represents an action that is not related to any commodity. Subclass `a:RelatedAction` represents an action related to a concrete product or service. This connection is expressed by object property `a:regardingCommodity`. The specific actions that businesses are interested in can vary and each business should have the ability to define their own types of actions. The ontology in Figure 4 is a generic action ontology that can easily be extended by businesses

#### 4.5 Customer Ontology

The customer ontology models information related to personal data of the customers together with their business preferences. Each customer can be reached through several channels. This is expressed through the class `cu>Contact`. A customer may have several contacts. Each contact is linked to a channel and has a datatype property representing customer's address in that particular channel. Also, the ontology defines a datatype property `cu:preference` for each contact. The value of the preference property is a real number between 0 and 1. This number expresses the customer's willingness to be informed using this contact. If the value is 0, the user never wants to be contacted. If the value is 1, the user always wants to be contacted. The list of customer ontology properties is

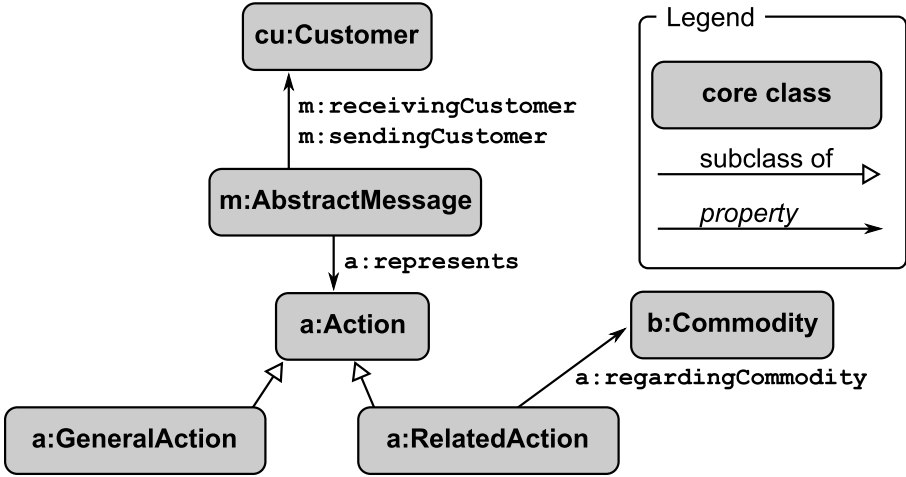


Fig. 4. Relationship between messages, actions and commodities

depicted in Table 3. We do not provide any class hierarchy figure, because both `ch:Customer` and `ch:Contact` are direct descendants of OWL class `owl:Thing`. The customer ontology contains only entities that are important for the proper framework operation. Personal information such as name or birth date are not included in the ontology. The framework user can provide them by extending the ontology.

Table 3. Object and datatype properties of the customer ontology

URI	Min. card.	Max. card.	Domain	Range
<code>cu:hasContact</code>	0	n	<code>cu:Customer</code>	<code>cu:Contact</code>
<code>cu:correspondingChannel</code>	1	1	<code>cu:Contact</code>	<code>ch:Channel</code>
<code>cu:contactAddress</code>	1	1	<code>cu:Contact</code>	<code>xsd:string</code>
<code>cu:preference</code>	1	1	<code>cu:Contact</code>	<code>xsd:float</code>

#### 4.6 Message Ontology

There are two basic types of messages – inbound and outbound. An inbound message is a message sent by a customer to the business. An outbound message is a message originating from the business and sent to one or more customers. According to another criterion a message can either be abstract or concrete. A concrete message is a message sent through a concrete channel from/to a concrete address with a concrete content. In case of a concrete message no information about the action or customer is included. An abstract message is a message on

a higher thought level. It is linked to a concrete customer from the customer database in case the customer is the receiver or sender of the message. It is also linked to the proper action that was supposed to be achieved through it. Both abstract and concrete messages can be either inbound or outbound. That makes it four types of messages. Apart from abstract and concrete messages the framework distinguishes a term called message template. A message template is a prescription for an outbound abstract message. In the message template the recipients, the content and the communication channel are known in a form of a query. Such a template can be converted into an outbound abstract message by resolving these queries. This concept is similar to the idea of executable knowledge by Terziyan [13] and will be described in detail in the next section.

The message ontology consists of four main classes – `m:ConcreteMessage`, `m:AbstractMessage`, `m:InboundMessage` and `m:OutboundMessage`. The class hierarchy is depicted in Figure 5. The property descriptions are in Table 4.



Fig. 5. Class hierarchy of the message ontology

A concrete message has datatype properties annotating the sender, receiver, subject and content. All these properties expect string as their values. As mentioned earlier, a concrete message is not mapped to a concrete customer URI. It only stores the information about the concrete address (string) it was sent to or received from. It is the job of the engine to do the conversion between an abstract and a concrete message. A concrete message also has two object properties. The one connects it to the communication channel that was used to send/receive the message. The other object property points to attachments that were sent or received together with the message. The user can extend the message ontology and specify subclasses of concrete messages such as text message (SMS), email, tweet, etc. Some of these subclasses do not specify certain properties. For example tweets and SMS messages do not have subjects. Some concrete messages may contain attachments and some may not.

**Table 4.** Object and datatype properties of the message ontology

URI	Min. card.	Max. card.	Domain	Range
m:to	1	n	m:ConcreteMessage	xsd:string
m:from	1	1	m:ConcreteMessage	xsd:string
m:dateReceived	1	1	m:Message	xsd:dateTime
m:subject	0	1	m:Message	xsd:string
m:content	0	1	m:Message	xsd:string
m:channel	1	1	m:Message	c:Channel
m:hasAttachment	0	n	m:Message	m:Attachment
a:represents	0	n	m:AbstractMessage	a:Action
m:receivingCustomer	0	1	m:AbstractMessage	cu:Customer
m:sendingCustomer	0	1	m:AbstractMessage	cu:Customer

An abstract message describes the sender and receiver in form of a URI identifying a concrete individual from the knowledge base. As mentioned in the action ontology description, an abstract message includes the information about an action that the message represents.

## 5 Message Conversion Engine

### 5.1 Message Template Description

A message template is a prescription for an abstract message. Naturally, the abstract message generation process requires data that the template should be filled with. The data needed for the generation process is expressed in a form of a query. During the conversion the query is executed by the engine and the result of the query represents that working data set that is used to fill the template with data. A message template is a prescriptive element, not a descriptive one. Therefore we believe that RDF is not the most suitable way of expressing it. In Figure 6 we present a simple language to describe a message template.

The template description consists of four main parts – query, recipient specification, message content and channel specification. The recipient specification defines who should receive this message. It can be a single person or several people. The message content is a piece of information that will become the content of the concrete message after the conversion. Use of variable is permitted in the message content specification. Lastly, channel specification is a description of a channel that should be used to send the concrete message.

The query defines the working dataset that the conversion engine will work with during the conversion process. This query is written in SPARQL (SPARQL Protocol and RDF Query Language), which is a standard query language for RDF data. The other elements describe the other three main parts of the message by referring to the data in the working dataset. Recipients of the message are defined as an enumeration of concrete customers through their URIs or

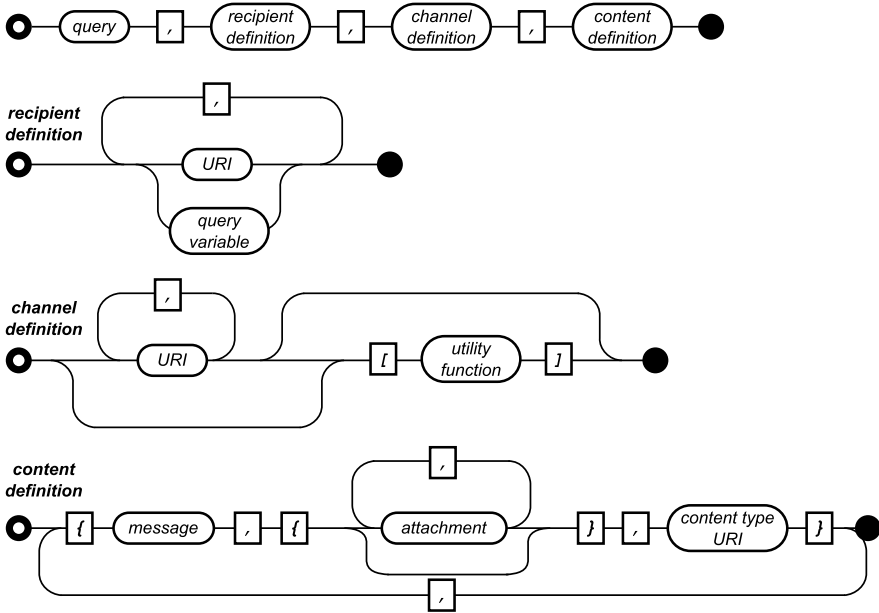


Fig. 6. Message template description language

variable names from the query. Message content can be a combination of textual messages, images, attachments, etc. Each message content specification is accompanied with a content type specification. These are used to determine which communication channel can be used to transfer them. For example if the message is a simple short text, it can be used as an email, SMS or tweet. However, if it is a text with images and some PDF document as an attachment, it can be specified as an email message only. Each abstract message can have several message content specifications with accompanying content type definitions. An example of a message template can be seen in Figure 7.

The query portion of the abstract message above describes a dataset that contains information about customers and products. The query is looking for customers that have bought a consumer electronics product after March 1st 2012. It also looks for the product name, product service file URL and product service file physical location. The query is used later in the message conversion process. The recipients of the message will be customers specified by a variable. The content of the message are two triples. The first triple contains some text with variables, attached service file and content type description specifying that it is an HTML text with an attachment. The second triple consists of some text, empty attachment set and content type description indicating that it is a short text. We will use this abstract message later to describe the message conversion process.

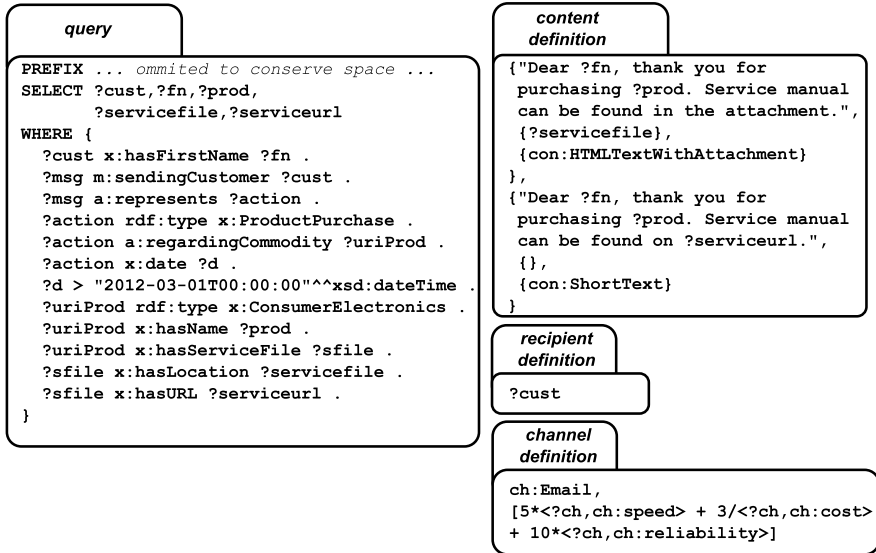


Fig. 7. An example of a message template

## 5.2 Message Conversion Process

The goal of the message conversion process is to produce a set of concrete messages by applying a message template on top of the contextual data. The message conversion process is described in a function called `ConvertMessage` (see Algorithm 1). It takes a message template and applies it on top of the data by performing the SPARQL query described in the query portion of the template.

The message conversion algorithm first performs a SPARQL query defined by the query portion of the abstract message definition. The result of the query is a binding table where each binding is represented by a row. A sample binding table is shown in Table 5. The sample binding table contains five variables representing the customer (`?cust`), customer's first name (`?fn`), product name (`?prod`), service file location (`?servicefile`) and URL of the service file on the producer's web page (`?serviceurl`). Then the algorithm iterates through all lines and for each line it determines the recipient and content. The channel definition is taken as it is provided in the message template. The channel selection process is described later in the chapter. The recipient is relatively straightforward to determine, since it is represented by a single variable. In the example from Figure 7 the recipient is represented by variable `?cust`. The content is determined by substituting the variables in the template text with the values from the binding table. In the example from Figure 7 these variables are `?fn`, `?prod`, `?servicefile` and `?serviceurl`.

Recipient, channel, and content are three parameters that define an abstract message. Please note that in case of an abstract message the recipient is determined by a list of user URIs. Similarly, channel is determined by a list of



**Algorithm 1.** ConvertMessage

---

```

input : MsgTemplate msgtemp
TableRow[] rows = performSPARQLQuery (msgtemp.query);
ConcreteMessage[] concrMsgs;
foreach row in rows do
    | URI recipient = row.getColumn (msgtemp.recipientVariable);
    | ChannelDefinition[] channel = row.getColumn (msgtemp.channelVariable);
    | ContentDescription[] content = msgtemp.contentDescription;
    | concrMsgs.add(ConvertAbstractToConcrete (recipient, channel, content));
end
foreach cMsg in concrMsgs do
    | SendConcreteMessage(cMsg);
end

```

---

**Table 5.** Sample binding table based on a fictional customer and product database

?cust	?fn	?prod	?servicefile	?serviceurl
u:id5	John	Sonic X52	/mans/X52.pdf	http://sonic.com/X52.pdf
u:id3	Jane	Jogman 2000	/mans/Jog2k.pdf	http://tony.com/J2000.pdf
u:id2	Bill	Pear uPhone	/mans/PuP.pdf	http://pear.com/uPhone.pdf

channel URIs or utility functions with resolved variables. The content is represented as a list of triples (message content, attachments, content type) with resolved variables. Thanks to the use of URIs, an abstract message can be linked to concrete users and channels. Figure 8 contains an example of an abstract message based on the first row of the binding table. In the example we can see that the first member of the content list is a triple with a text for customer named John, attachment containing one PDF file and marked as a content type `con:HTMLTextWithAttachment`. The second member of the content list contains just text and content type, but it does not contain any attachment. In the channel section we can see the same content as in the channel section of the message template. One channel was given statically and one is described using a utility function (see Figure 7). The channel selection process is explained later in the chapter.

However, an abstract message does not contain enough information for a message to be sent to a concrete user. Firstly, it is still lacking a concrete address (e.g. email address, phone number, social network account, etc.). Secondly, in case utility functions are used to specify the communication channels, it is also lacking information about the concrete channel. Lastly, the content is specified as a list of content triples describing various content types. Therefore it has to be converted it into a concrete message. A concrete message then contains a concrete address, concrete message text with concrete attachments and concrete channel that it should be sent to. The conversion from an abstract message to a concrete message is done by calling `ConvertAbstractToConcrete` function (see Algorithm 2).

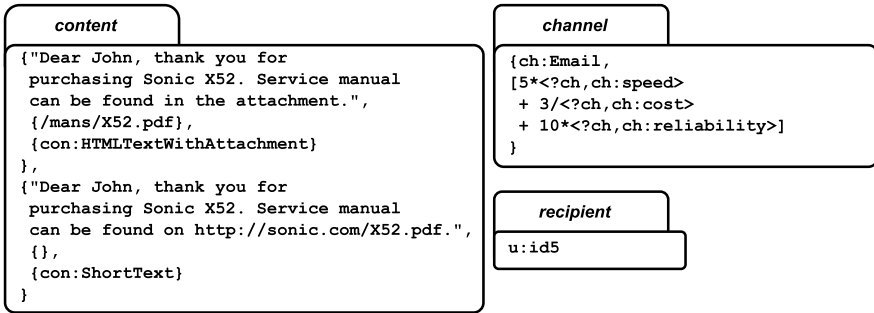


Fig. 8. An example of an abstract message

---

**Algorithm 2.** ConvertAbstractToConcrete
 

---

```

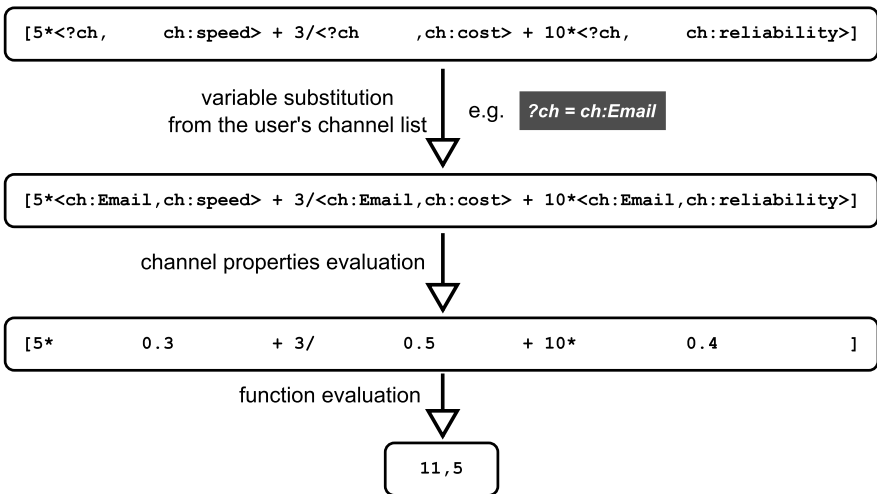
input  : URI customer,
        ChannelDefinition[] channelDefs,
        ContentDefinition[] contentDefs
output: ConcreteMessage[] concrMsgs
ConcreteMessage[] concrMsgs;
URI[] channels;
foreach chDef in channelDefs do
  switch typeOf (chDef) do
  | case URI
  |   channels.add(chDef);
  |   break;
  | case UtilityFunction
  |   channels.add(determineChannel(chDef, customer));
  |   break;
  | endsw
end
foreach ch in channels do
  ChannelHandler chh = getChannelHandler(ch);
  foreach content in contentDefs do
    if channelAccepts (channel, content.contentType) then
      Message msg = chh.convertMessage(content.msg,
      content.attachmentList, content.contentType);
      Address addr = chh.extractAddress(content.user);
      concrMsgs.add(channel, addr, msg);
    end
  end
end
return concrMsgs;
  
```

---

ConvertAbstractToConcrete function takes an abstract message as the input and produces a set of concrete messages as the output. Please note that the three input parameters of ConvertAbstractToConcrete function correspond to recipient, channel, and content parameters of the abstract message mentioned above. The first step of the conversion is to obtain the channel handler based on the channel URI. The handler is capable of message content generation and extracting addresses from user descriptions. Then the function iterates through the list of content triples and for each triple it checks if the particular channel accepts the particular content type from the triples. For example channel `ch:SMS` can accept content type `con:ShortText`, but it cannot accept content type `con:HTMLwithAttachment`. If the channel can accept the content type defined in the abstract message description, then a concrete message is created with the help of a channel handler. If the channel cannot accept the content type, the algorithm proceeds to the next content description triple. As a result a set of concrete message is created.

### 5.3 Channel Selection

The channel selection depends on the context data and the message template used to generate the message. From ConvertAbstractToConcrete function it is clear that the channel definition section of the message template may contain either a channel URI or a utility function. If a channel URI is found, it is simply added to the list channels that will be used to send the concrete message. However, if a utility function is found, the process is more complicated. In ConvertAbstractToConcrete function the process of utility function evaluation is hidden behind the determineChannel call.



**Fig. 9.** An example of a utility function substitution and evaluation

The `determineChannel` function takes the recipient's URI and finds all channels that the recipient may use. Then, in an iteration each of them is used to substitute the `?ch` variable in the utility function definition and the function is evaluated. The substitution and evaluation process is depicted in Figure 9. The `<channel URI, property>` pairs are replaced with values and the utility function is evaluated. The result is a real number representing the score of a given channel. Higher value represents more desirable channel. The channel with the highest value is considered the most suitable one and it is selected.

## 6 Conclusion

We presented a multi-channel communication framework capable of message integration from various communication channels. The framework consists of two main parts – knowledge base and message conversion engine. The framework's knowledge base is built on top of semantic technologies such as RDF, OWL, SPARQL, etc. The data is stored in form of RDF triples. The data schema is based on five main OWL ontologies. The commodity ontology describes products and services that the business is dealing with. The communication channel ontology is model for communication channels, channel handler and content types. The action ontology represents a variety of actions that the business or a customer can perform. The fourth ontology is the customer ontology dealing with customer's personal information, contact addresses and channel preferences. Lastly, the message ontology defines abstract and concrete messages and their relationships to other components. All the ontologies are extendable by the framework user in order to reflect the business needs.

The message conversion engine works with message templates that are being transformed into concrete messages. The message template is a prescription for a concrete message and it is defined by four parts – SPARQL query, recipient definition, channel definition and content definition. The engine performs the query and retrieves a working data set in form of a binding table. Then it chooses the best communication channel to send the message to. Lastly, it generates the message content based on the content specifications and communication channel's capability of accepting various content types.

**Acknowledgements.** The author would like to thank the industrial partner IPSS (Intelligent Precision Solutions and Services), members of IOG (Industrial Ontologies Group) and the TIVIT Cloud Software Program for supporting this work. Moreover, many thanks go to the reviewers for their helpful comments.

## References

1. Berners-Lee, T.: Linked Data (2006), <http://www.w3.org/DesignIssues/LinkedData.html>
2. data.gov.uk, <http://data.gov.uk/>
3. DBpedia, <http://dbpedia.org>

4. Hendler, J.: Agents and the Semantic Web. *IEEE Intelligent Systems* 2, 30–37 (2001)
5. HermiT reasoner, <http://hermit-reasoner.com/>
6. Jena framework, <http://jena.apache.org/>
7. Nagy, M.: On the Problem of Multi-Channel Communication. In: *Proceedings of ICTERI 2012*, Kherson, Ukraine, pp. 128–133 (2012)
8. Protégé ontology editor, <http://protege.stanford.edu/>
9. RacerPro reasoner, <http://www.racer-systems.com/>
10. Resource Description Framework, <http://www.w3.org/RDF/>
11. Resource Description Framework Schema, <http://www.w3.org/TR/rdf-schema/>
12. Sesame framework, <http://www.openrdf.org/>
13. Terziyan, V., Kaykova, O.: From Linked Data and Business Intelligence to Executable Reality. *International Journal on Advances in Intelligent Systems* 5, 194–208 (2012)
14. Web Ontology Language, <http://www.w3.org/TR/owl-features/>
15. XML Schema, <http://www.w3.org/XML/Schema>

# Satisfiability and Validity Problems in Many-Sorted Composition-Nominative Pure Predicate Logics

Mykola S. Nikitchenko and Valentyn G. Tymofieiev

Department of Theory and Technology of Programming  
Taras Shevchenko National University of Kyiv  
64, Volodymyrska Street, 01601 Kyiv, Ukraine  
nikitchenko@unicyb.kiev.ua,  
tvalentyn@univ.kiev.ua

**Abstract.** We propose methods for solving the satisfiability and validity problems in many-sorted composition-nominative pure predicate logics (without functions and with equality). These logics are algebra-based logics of many-sorted partial predicates constructed in a semantic-syntactic style on the methodological basis that is common with programming; they can be considered as generalizations of traditional many-sorted logics on classes of partial predicates that do not have fixed arity. We show the reduction of the satisfiability problem to the same problem for many-sorted classical first-order pure predicate logic with equality. As validity is dual to satisfiability, the method proposed can be adopted to the validity problem. This enables us to use existent satisfiability and validity checking procedures developed for classical logic also for solving these problems in composition-nominative pure predicate logics with equality.

**Keywords:** many-sorted logic, composition-nominative logic, partial predicate, quasiary predicate, partial logic, first-order logic, satisfiability, validity.

## 1 Introduction

Satisfiability and validity problems [1] are important for such areas as program verification, synthesis, analysis, testing, etc. [2–5]. In this chapter we address the satisfiability and validity problems in the context of the *composition-nominative approach* [6], which aims to construct a hierarchy of logics of various abstraction and generality levels on the methodological basis that is common with programming. The main principles of the approach are principles of *development from abstract to concrete*, *priority of semantics*, *compositionality*, and *nominativity*.

These principles specify a hierarchy of new logics that are semantically based on algebras of predicates, functions, and programs, which are considered as partial mappings. Operations over such mappings are called *compositions*. Data classes are considered on various abstraction levels, but the main attention is paid to the class of *nominative data*. Such data consist of pairs *name–value*. Nominative data can represent various data structures such as records, arrays, lists, relations, etc. [6, 7]; this explains the importance of the notion of nominative data. In the simplest case

nominative data can be treated as partial mappings from a certain set  $V$  of (possibly typed) names (or variables) into a set of basic (atomic) values. Such data are called *nominative sets*. Nominative sets represent program states for simple programming languages (see, for example, [6–8]). From this follows that semantic models of programs and logics are mathematically based on the notion of nominative set (nominative data in general case). Partial mappings over nominative sets are called *quasiary*. This fact permits to integrate models of programs and logics, and represent them as a hierarchy of composition-nominative models [9, 10]. Logics developed within such approach are called *composition-nominative logics* (CNL) because such logics are determined 1) by operations (*compositions*) in algebras of predicates, functions, and programs, and 2) by *nominative* structures of data on which these predicates, functions and programs are defined. In this connection we should admit that, in particular, logical connectives and quantifiers are also formalized as compositions because compositions represent general semantic constructs used in logic.

CNL can be considered as generalization of classical predicate logic but many methods developed within classical logic can also be applied to CNL. Here we confirm this statement for the satisfiability and validity problems in CNL. The investigations on the topic in hand were initiated in [11] and developed in [12]. The latter work provides a brief discussion of related approaches and presents hierarchy of CNL along with reduction methods for solving satisfiability problems in abstract CNL: propositional, quantifier-free, and pure first-order predicate logics without equality. In [13] similar result was announced for pure predicate logics with equality.

In this chapter we introduce the notion of many-sorted CNL and consider satisfiability and validity problems for many-sorted pure CNL (a predicate logic with equality but without functions). We construct algorithms that reduce the satisfiability and validity problems in this logic to the same problems in classical first-order predicate logic with equality.

The chapter is structured in the following way. In section 2 we give a motivating example; then in section 3 we give formal definitions of the many-sorted logics studied in this chapter, and define the satisfiability and validity problems. In section 4 we describe reduction methods for solving these problems and present a simple example. In section 5 we summarize our results and indicate directions for future investigations.

## 2 Motivating Example

Let us consider a simple programming language ELT (Example Language with Types) which is used here to demonstrate how many-sorted program logics can be constructed. ELT is similar to such languages as WHILE [8], IMP [14], etc.

The grammar of the language is defined as follows:

```

prg ::= begin var dcl ; stm end
dcl ::= i: integer | x: real | dcl ; dcl
stm ::= i:=ie | x:=re | stm1 ; stm2 | if b then stm1 else stm2 |
       while b do stm | begin stm end
ie ::= n | i | ie1 + ie2 | ie1 - ie2 | ie1 * ie2 | (ie)

```

$$re ::= c \mid x \mid re_1 + re_2 \mid re_1 - re_2 \mid re_1 * re_2 \mid re_1 / re_2 \mid (re)$$

$$b ::= ie_1 = ie_2 \mid ie_1 \neq ie_2 \mid ie_1 > ie_2 \mid re_1 = re_2 \mid re_1 \neq re_2 \mid re_1 > re_2 \mid b_1 \vee b_2 \mid \neg b \mid (b)$$

where:

- $n$  ranges over integer numbers  $Int = \{\dots, -2, -1, 0, 1, 2, \dots\}$
- $c$  ranges over real numbers  $Real = \{\dots, -0.1, \dots, 0.0, \dots, 0.1, \dots\}$
- $i$  ranges over variables (names) of integer type  $V_I = \{I, M, N, \dots\}$
- $x$  ranges over variables (names) of real type  $V_R = \{R, X, Y, \dots\}$
- $ie$  ranges over integer expressions  $Iexpr$
- $re$  ranges over real expressions  $Rexpr$
- $b$  ranges over Boolean expressions  $Bexpr$
- $stm$  ranges over statements (programs)  $Stm$
- $dcl$  ranges over variable declarations  $Dcl$
- $prg$  ranges over programs  $Prg$

As an example consider an ELT program  $RF$  for calculating a rational function  $r = 1/x^n$ . In this program we use variables  $N$ ,  $X$ , and  $R$  for denoting values  $n$ ,  $x$ , and  $r$  respectively:

```
begin
  var N: integer; X: real; R: real;
  begin
    R:=1.0;
    while N≠0 do begin N:=N-1; R:=R/X end
  end
end
```

Starting from this example we construct program algebras of two forms:

- First, we define semantics of  $RF$  in the style of denotational semantics; as a result we obtain an algebra oriented on the program  $RF$ . This algebra is a *many-sorted program algebra* with partial quasiary predicates, ordinary functions, and program functions.
- Then, we define a *class of many-sorted program algebras*. This class of algebras is a semantic base for program logics. It captures main program properties that are invariant of such programs' specifics as variable typing, interpreted functions, etc.

To emphasize mapping's partiality/totality we use the sign  $\xrightarrow{p}$  for partial mappings and the sign  $\xrightarrow{t}$  for total mappings; therefore the terms 'partial' and 'total' are often omitted.

To formalize program semantics various approaches were proposed. Here we will follow denotational semantics (see, for example, [8, 14]). Semantic mapping is represented as  $\llbracket \cdot \rrbracket$ . On the abstract level semantics of  $RF$  can be treated as a certain function. Given a state  $d$  of the form  $\llbracket N \mapsto n, X \mapsto x \rrbracket$  where  $n$  is an integer number and  $x$  is a real number, this function evaluates (if terminates) a new state of the form  $\llbracket N \mapsto 0, X \mapsto x, R \mapsto r \rrbracket$  in which  $r$  – a value of the variable  $R$  – is equal to  $1/x^n$ .



For our program  $RF$  examples of states are  $[X \mapsto 5.3, N \mapsto 4]$ ,  $[X \mapsto 5.3, N \mapsto 4, R \mapsto 8.2]$ ,  $[X \mapsto 5.3]$ . In a state  $d$  a variable  $v$  can have a value (this is denoted  $d(v)\downarrow$ ) or be undefined (denoted  $d(v)\uparrow$ ); thus,

$$[X \mapsto 5.3, N \mapsto 4](X)\downarrow \text{ and } [X \mapsto 5.3, N \mapsto 4](R)\uparrow.$$

Let  $V = \{N, X, R\}$ ,  $T = \{Int, Real\}$ ,  $A = Int \cup Real$ . To give a definition of states with typed variables we first introduce type valuation (type assignment) mapping  $\tau_{RF} : V \xrightarrow{t} T$  in the following way:  $\tau_{RF} = [N \mapsto Int, X \mapsto Real, R \mapsto Real]$ . Note that this notation can be considered as another form of variable declarations: “ $N$ : integer;  $X$ : real;  $R$ : real”. Now, having  $\tau_{RF}$  we can define the set of states  $State_{RF}$  as the set of all partial mappings  $d : V \xrightarrow{p} A$  such that the value of  $N$  in  $d$  belongs to  $Int$  if it is defined and the values of  $X$  and  $R$  belong to  $Real$  if they are defined.

Having described states we are able to represent formal semantics of arithmetic and Boolean expressions. Integer expressions denote functions (called *many-sorted quasiary functions of integer type*) of the set  $Fn^{Int}(\tau_{RF}) = State_{RF} \xrightarrow{p} Int$ ; thus, for  $ie \in Iexpr$  we have  $\llbracket ie \rrbracket \in Fn^{Int}(\tau_{RF})$ . Real expressions denote functions (called *many-sorted quasiary functions of real type*) of the set  $Fn^{Real}(\tau_{RF}) = State_{RF} \xrightarrow{p} Real$ ; thus, for  $re \in Rexpr$  we have  $\llbracket re \rrbracket \in Fn^{Real}(\tau_{RF})$ . Boolean expressions denote predicates (called *many-sorted quasiary predicates*) of the set  $Pr(\tau_{RF}) = State_{RF} \xrightarrow{p} Bool$ ; thus for  $b \in Bexpr$  we have  $\llbracket b \rrbracket \in Pr(\tau_{RF})$ .

As states are constructed with the help of naming (nominative) relation, they are also called *typed nominative sets* and their class is denoted by  $NST(\tau_{RF})$ . Therefore functions of the classes  $Fn^{Int}(\tau_{RF})$ ,  $Fn^{Real}(\tau_{RF})$ , and predicates of the class  $Pr(\tau_{RF})$  are also called *integer nominative functions*, *real nominative functions*, and *nominative predicates* respectively since they are defined on classes of nominative sets. As to classification related to the ranges of functions from  $Fn^{Int}(\tau_{RF})$  and  $Fn^{Real}(\tau_{RF})$  we call them *ordinary functions* since their ranges are sets of atomic (non-structured) values.

In  $RF$  the following expressions occur: Boolean expression  $N \neq 0$ , integer expressions  $1$  and  $N-1$ , real expressions  $1.0$  and  $R/X$ . For simplicity's sake we use bold font to represent semantics of these expressions, thus,

$$\llbracket N \neq 0 \rrbracket = N \neq \mathbf{0}, \llbracket 1 \rrbracket = \mathbf{1}, \llbracket N-1 \rrbracket = N-\mathbf{1}, \llbracket 1.0 \rrbracket = \mathbf{1.0}, \text{ and } \llbracket R/X \rrbracket = \mathbf{R/X}.$$

These mappings have the following types:  $N \neq \mathbf{0} \in Pr(\tau_{RF})$ ;  $\mathbf{1}$ ,  $N-\mathbf{1} \in Fn^{Int}(\tau_{RF})$ ;  $\mathbf{1.0}$ ,  $\mathbf{R/X} \in Fn^{Real}(\tau_{RF})$ . Functions  $\mathbf{1}$  and  $\mathbf{1.0}$  are constant functions. Note that  $\mathbf{R/X}$  is partial.

Analyzing the structure of the program we see that it is constructed from the above considered expressions and from structuring constructs such as assignment, sequencing, and loop.

The semantics of structuring constructs is defined with the help of special operators (compositions).

Programs from  $Prg$  and statements from  $Stm$  denote program functions of the class  $FPrG(\tau_{RF}) = State_{RF} \xrightarrow{P} State_{RF} = NST(\tau_{RF}) \xrightarrow{P} NST(\tau_{RF})$ . Such functions are also called *biquasiary functions*; they belong to the class of *binominative functions*.

Semantics of structured statements is defined by total  $n$ -ary compositions. Each composition has a type (also called *type arity* of composition). The following compositions with conventional meaning are used for formalizing semantics of  $RF$ :

1. Integer assignment composition  $AS_I^i: Fn^{Int}(\tau_{RF}) \xrightarrow{t} FPrG(\tau_{RF})$  and real assignment composition  $AS_R^x: Fn^{Real}(\tau_{RF}) \xrightarrow{t} FPrG(\tau_{RF})$  (parameter  $i$  denotes a variable of integer type and parameter  $x$  denotes a variable of real type);
2. Composition of sequential execution  $\bullet: FPrG(\tau_{RF}) \times FPrG(\tau_{RF}) \xrightarrow{t} FPrG(\tau_{RF})$ ;
3. Loop composition  $WH: Pr(\tau_{RF}) \times FPrG(\tau_{RF}) \xrightarrow{t} FPrG(\tau_{RF})$ .

Thus,

$$\llbracket N := N - 1 \rrbracket = AS_I^N(N-1), \llbracket N := N - 1; R := R/X \rrbracket = AS_I^N(N-1) \bullet AS_R^R(R/X).$$

Note, that we define  $\bullet$  by commuting arguments of conventional functional composition:  $f \bullet g = g \circ f$ .

The definitions introduced permit to conclude that the following *many-sorted program algebra* with many-sorted quasiary predicates, ordinary functions, and program functions has been constructed:

$$A^{RF}(\tau_{RF}) = \langle Pr(\tau_{RF}), Fn^{Int}(\tau_{RF}), Fn^{Real}(\tau_{RF}), FPrG(\tau_{RF}); \\ N \neq 0, 1, N-1, 1.0, R/X, AS_I^i, AS_R^x, \bullet, WH \rangle.$$

This algebra has four carriers of quasiary mappings ( $Pr(\tau_{RF}), Fn^{Int}(\tau_{RF}), Fn^{Real}(\tau_{RF}), FPrG(\tau_{RF})$ ), one predicate ( $N \neq 0$ ), four ordinary functions ( $1, N-1, 1.0, R/X$ ), and four compositions ( $AS_I^i, AS_R^x, \bullet, WH$ ). The predicate and ordinary functions are considered as algebra constants.

We would like to emphasize the fact that semantics of  $RF$  program or its subexpressions can be represented as terms of this algebra. In particular, the term for  $RF$  program is as follows:

$$\llbracket RF \rrbracket = AS_R^R(1.0) \bullet WH(N \neq 0, AS_I^N(N-1) \bullet AS_R^R(R/X)).$$

Note that this term and its subterms can denote partial mappings, as the division can be undefined; also  $WH$  composition can be a source of undefinedness.

Many-sorted algebra  $A^{RF}(\tau_{RF})$  has been constructed directly oriented on  $RF$  program, therefore it is very restricted, but still it can be used to prove some properties of this program or its components. For example, we can prove the following equality (commutativity of assignment operators):

$$AS_I^N(N-1) \bullet AS_R^R(R/X) = AS_R^R(R/X) \bullet AS_I^N(N-1).$$

Please note that this equality holds for any typing of variables, say,  $N$  may have type of natural or real numbers,  $R$  and  $X$  can be typed as rational numbers, etc. This observation puts a question: what program properties are invariant under type valuations, or under predicate/function variations, etc? In other words, what properties are general program properties that are independent of program's specifics, or, more precisely, what program properties hold for a certain class of program algebras? This question has a logical nature related to general laws of programs. To answer such questions we construct special program logics called *composition-nominative program logics*.

The idea of such construction can be explained on example of algebra  $A^{RF}(\tau_{RF})$ . In this algebra we have carriers of three kinds: 1) quasiary predicates, 2) integer and real quasiary functions (ordinary functions), and 3) biquasiary functions (program functions). This will lead to many-sorted logics of three levels: 1) pure predicate logics based on algebras with one carrier (predicates), 2) predicate-function logics based on algebras with carriers of two kinds (predicates and ordinary functions), and 3) predicate-function-program logics based on algebras with carriers of three kinds (predicates, ordinary functions, and programs functions). Obtained logics are called composition-nominative pure predicate logics, composition-nominative predicate-function logics, and composition-nominative program logics respectively.

The above-mentioned classes of algebras constitute *semantic components* of corresponding logics. As to *syntactic component*, it is specified by classes of algebra terms. In logic, terms of algebras related to predicates are usually called *formulas*, terms related to classes of ordinary functions are called by the same word – *terms*, and terms related to program functions are called here *program texts*. Logics under construction should usually be more expressive than program algebras they are built upon. In particular, the set of compositions over predicates additionally includes such compositions as renomination [10] and quantification; program logics should also include compositions that relate program functions with predicates. As an example of such composition we can mention ternary (Floyd–Hoare) composition [12], which given two predicates (precondition and postcondition) and a program function produces new predicate that specifies the validity of corresponding Floyd–Hoare assertion.

To construct a composition-nominative program logic we choose the following *semantic-syntactic scheme*. First, we should describe a class of algebras that forms a semantic base of corresponding logic (*semantic component of a logic*). Then, we should describe a signature of the logic and a class of formulas (language expressions in general) constructed over the signature (*syntactic component of a logic*). At last, we should define methods of interpreting language expressions in each algebra of the constructed class (*interpretational component of a logic*).

A class of algebras (*semantic component*) is specified by the following components:

- A set of types denoted by a capital Greek letter  $T$
- A set of variables denoted by  $V$
- A class of all type valuations of the form  $\tau: V \xrightarrow{t} T$ ; for each  $\tau$  we define:

- A class of typed nominative sets  $NST(\tau)$
- A class of predicates  $Pr(\tau) = NST(\tau) \xrightarrow{P} Bool$
- Classes of ordinary functions of the form  $Fn^A(\tau) = NST(\tau) \xrightarrow{P} A$  (for each  $A \in T$ )
- A class of program (binominative) functions  $FPr_g(\tau) = NST(\tau) \xrightarrow{P} NST(\tau)$
- A class  $C(\tau)$  of compositions with fixed types (type arities) which specify their domains and ranges (this class is defined in a uniform way for all type valuation mappings)
- An algebra  $A(\tau) = \langle Pr(\tau), \{Fn^A(\tau) \mid A \in T\}, FPr_g(\tau); C(\tau) \rangle$

A many-sorted program logic language (*syntactic component*) consists of logic expressions partitioned into classes of formulas, terms, and program texts. Informally speaking, it is specified by the following components:

- A set  $S$  of sorts (sorts may be considered as type variables)
- A *signature*  $\Sigma^S = (V, S, \xi)$  of sort valuation (here  $\xi : V \xrightarrow{t} S$  is a sort valuation mapping)
- A set of composition symbols  $Cs(\Sigma^S)$  with their sort arities which specify sorts for their domains and ranges. Here a sort arity of a composition symbol is a formal expression of the form  $E_1 \times \dots \times E_n \xrightarrow{t} E$  where  $E_1, \dots, E_n, E \in \{Pr(\xi)\} \cup \{Fn^s(\xi) \mid s \in S\} \cup \{FPr_g(\xi)\}$ ,  $n \geq 0$
- Sets  $Ps$ ,  $Fs$ , and  $Pr_g$ s of predicate symbols, ordinary function symbols with their sorts, and program symbols respectively
- A *language signature* specified by sets  $\Sigma^S$ ,  $Cs(\Sigma^S)$ ,  $Ps$ ,  $Fs$ ,  $Pr_g$ s and by (implicit) arity mappings for symbols of compositions and ordinary functions
- Sets of atomic expressions (atomic formulas, terms, and program texts) with their sort arities; for composition-nominative logics these sets usually coincide with  $Ps$ ,  $Fs$ , and  $Pr_g$ s respectively; symbols of null-ary compositions are not considered as atomic formulas because they have fixed interpretation
- Inductive rules for constructing new logic expressions from existing ones (for each composition symbol  $cs \in Cs(\Sigma^S)$ ). In such constructions sort arities of expressions should be taken into account; thus for each logic expression its sort arity can be evaluated

An interpretational component is defined by:

- A class of sort interpretation mappings of the form  $I^S : S \xrightarrow{t} T$ . In combination with  $\xi$  a mapping  $I^S$  specifies type evaluation mapping  $\tau = I^S \circ \xi$ , and consequently, a certain algebra  $A(\tau)$
- A uniform interpretation of composition symbols from  $Cs(\xi)$  as compositions in the algebra  $A(\tau)$ ; this interpretation is fixed for the logic, therefore it is not included in the list of interpretation mappings

- Classes of interpretation mappings of predicate, function, and program symbols of the form

$$I^{Ps} : Ps \xrightarrow{t} Pr(\tau), \quad I^{Fs} : Fs \xrightarrow{t} \bigcup_{A \in T} Fn^A(\tau), \quad \text{and} \quad I^{Prgs} : Prgs \xrightarrow{t} FPrg(\tau)$$

- Classes of interpretation mappings for atomic expressions (if they differ from  $Ps$ ,  $Fs$ , and  $Prgs$ )

A tuple  $(A(\tau), I^{Ps}, I^{Fs}, I^{Prgs})$  is called a *model of a composition-nominative program logic*. Models can be represented by tuples of the form  $(\Sigma^S, I^{Ps}, I^{Fs}, I^{Prgs})$  called *language interpretation* (or *logic interpretation*) usually denoted here by  $J$  (possibly with subscripts). Given such an interpretation, each logic expression can be interpreted in a traditional way as predicate, ordinary function, or program function in algebra  $A(\tau)$ . Thus, interpretational component describes classes of language models represented by language interpretations.

The proposed scheme permits to define various kinds of composition-nominative program logics varying from simple ones to more complex ones. Still, taking into consideration the fact that this chapter is practically a first step in studying many-sorted composition-nominative program logics, we restrict ourselves to considering satisfiability and validity problems only for *composition-nominative pure predicate logics with equality*, i.e. logics which are based on a class of algebras of the form  $A(\tau) = \langle Pr(\tau), C(\tau) \rangle$  where the set  $C(\tau)$  consists of compositions of disjunction  $\vee$ , negation  $\neg$ , renomination  $R_{\bar{x}}$ , existential quantification  $\exists x$ , and equality predicate  $=_{xy}$  (see the next section for formal definitions).

### 3 Formal Definitions of Many-Sorted Composition-Nominative Pure Predicate Logics

To define the logics we have to specify their semantic, syntactic, and interpretational components. Semantic component is formed by special predicate algebras considered below.

#### 3.1 Algebras of Quasiary Predicates over Typed Nominative Sets

Let  $V$  be a *set of names*. According to tradition, names from  $V$  are also called *variables*. Let  $T$  be a *class of types* and  $\tau : V \xrightarrow{t} T$  be a total mapping called *type valuation*.

Given  $V$ ,  $T$ , and  $\tau$ , a class  $NST(V, T, \tau)$  (shortly:  $NST(\tau)$ ) of typed nominative sets is defined by the following formula:

$$NST(\tau) = \left\{ d : V \xrightarrow{p} \bigcup_{A \in T} A \mid \forall v \in V (d(v) \downarrow \Rightarrow d(v) \in \tau(v)) \right\}.$$

Informally speaking, typed nominative sets represent states of typed variables. Though nominative sets are defined as mappings, we follow mathematical traditions and also use set-like notation for these objects. In particular, the notation  $d = [v_i \mapsto a_i \mid i \in I]$  describes a nominative set  $d$ . Ternary membership relation  $v_i \mapsto a_i \in_n d$  means that  $d(v_i)$  is defined and its value is  $a_i$  ( $d(v_i) \downarrow = a_i$ ).

Let  $Bool = \{F, T\}$  be a set of Boolean truth values. Let  $Pr(V, T, \tau) = NST(\tau) \xrightarrow{P} Bool$  be a set of all partial predicates (this set is shortly denoted by  $Pr(\tau)$ ). Predicates from  $Pr(\tau)$  are called *many-sorted partial quasiary predicates*.

For  $p \in Pr(\tau)$ ,  $d \in NST(\tau)$ ,  $v \in V$ ,  $a \in \tau(v)$  we write:

- $p(d) \downarrow$  to denote that  $p$  is defined on a nominative set  $d$
- $p(d) \downarrow = b$  to denote that  $p$  is defined on  $d$  with a Boolean value  $b$
- $p(d) \uparrow$  to denote that  $p$  is undefined on  $d$
- $d(v) \downarrow$  to denote that a component with a name  $v$  is present in  $d$
- $d(v) \downarrow = a$  to denote that  $v \mapsto a \in_n d$
- $d(v) \uparrow$  to denote that the value of the name  $v$  is undefined in  $d$

Operations over  $Pr(\tau)$  are called *compositions*. For pure predicate logic the set of compositions  $C(V, T, \tau)$  (shortly:  $C(\tau)$ ) is  $\{\vee, \neg, R_{\bar{x}}, \exists x, =_{xy}\}$ . These compositions are defined as follows ( $p, q \in Pr(\tau)$ ,  $d \in NST(\tau)$ ).

- Binary composition of *disjunction*  $\vee : Pr(\tau) \times Pr(\tau) \xrightarrow{t} Pr(\tau)$  :

$$(p \vee q)(d) = \begin{cases} T, & \text{if } p(d) \downarrow = T \text{ or } q(d) \downarrow = T, \\ F, & \text{if } p(d) \downarrow = F \text{ and } q(d) \downarrow = F, \\ \text{undefined} & \text{in other cases.} \end{cases}$$

- Unary composition of *negation*  $\neg : Pr(\tau) \xrightarrow{t} Pr(\tau)$  :

$$(\neg p)(d) = \begin{cases} T, & \text{if } p(d) \downarrow = F, \\ F, & \text{if } p(d) \downarrow = T, \\ \text{undefined} & \text{if } p(d) \uparrow. \end{cases}$$

- Unary parametric composition of *renomination*  $R_{x_1, \dots, x_n}^{v_1, \dots, v_n} : Pr(\tau) \xrightarrow{t} Pr(\tau)$  :

$$(R_{x_1, \dots, x_n}^{v_1, \dots, v_n} p)(d) = p([v \mapsto a \in_n d \mid v \notin \{v_1, \dots, v_n\}] \nabla [v_i \mapsto d(x_i) \mid d(x_i) \downarrow, i \in \{1, \dots, n\}]),$$

where:  $v_1, \dots, v_n, x_1, \dots, x_n \in V$ ,  $v_1, \dots, v_n$  are distinct *upper names*,  $\tau(v_1) = \tau(x_1), \dots, \tau(v_n) = \tau(x_n)$ ,  $n \geq 0$ . The  $\nabla$  operation is defined as follows: if  $d_1$  and  $d_2$  are two nominative sets then  $d = d_1 \nabla d_2$  consists of all named pairs of  $d_2$  and only those pairs of  $d_1$ , whose names are not defined in  $d_2$ .

- Unary parametric composition of existential quantification  $\exists x: Pr(\tau) \xrightarrow{t} Pr(\tau)$  with the parameter  $x \in V$ :

$$(\exists x p)(d) = \begin{cases} T, & \text{if there exists } a \in \tau(x): p(d \nabla x \mapsto a) \downarrow = T, \\ F, & \text{if for each } a \in \tau(x): p(d \nabla x \mapsto a) \downarrow = F, \\ \text{undefined} & \text{in other cases.} \end{cases}$$

Here  $d \nabla x \mapsto a$  means  $d \nabla [x \mapsto a]$ .

- Null-ary parametric composition of *equality*  $=_{xy}: Pr(\tau)$  with parameters  $x, y \in V$ ,  $\tau(x) = \tau(y)$ :

$$=_{xy}(d) = \begin{cases} T, & \text{if } d(x) \downarrow, d(y) \downarrow, \text{ and } d(x) = d(y), \\ T, & \text{if } d(x) \uparrow \text{ and } d(y) \uparrow, \\ F & \text{otherwise.} \end{cases}$$

Note that parametric compositions of existential quantification, renomination, and equality can also represent classes of compositions. Thus, notation  $\exists x$  can represent one composition, when  $x$  is fixed, or a class  $\{\exists x \mid x \in V\}$  of such compositions for various names.

A pair  $A^S(V, T, \tau) = \langle Pr(V, T, \tau), C(V, T, \tau) \rangle$  is called a *many-sorted algebra of quasiary predicates*. Such algebras form semantic base for a many-sorted composition-nominative pure predicate logic (here referred to as  $L$ ) and a many-sorted first-order classical pure predicate logic (here referred to as  $L_C$ ). Let us now proceed with syntactic and interpretational components of respective logics.

### 3.2 Many-Sorted Composition-Nominative Pure Predicate Logic $L$

*Syntactic component.* Let  $S$  be a set of sorts. Let  $\xi$  be a total mapping  $\xi: V \xrightarrow{t} S$  called sort valuation (sort assignment). A triple  $\Sigma^S = (V, S, \xi)$  is called a signature of sort valuation.

Let  $Cs(\Sigma^S)$  be a set of composition symbols that represent compositions in algebras defined above,  $Cs(\Sigma^S) = \{\vee, \neg, R_{\bar{x}}^{\bar{v}}, \exists x, =_{xy}\}$ . For simplicity, here and afterwards we use the same notation for symbols of compositions and compositions themselves. We also assume that each composition symbol is (implicitly) decorated with its sort arity.

Let  $Ps$  be a set of predicate symbols.

A triple  $\Sigma^L = (\Sigma^S, Cs(\Sigma^S), Ps)$  is a signature of a language of  $L$ .

Given a language signature  $\Sigma^L = (\Sigma^S, Cs(\Sigma^S), Ps)$ , we inductively define the language of  $L$  – the set of formulas  $Fr(\Sigma^L)$ :

1. If  $P \in Ps$  then  $P \in Fr(\Sigma^L)$
2. If  $\Phi, \Psi \in Fr(\Sigma^L)$  then  $(\Phi \vee \Psi) \in Fr(\Sigma^L)$
3. If  $\Phi \in Fr(\Sigma^L)$  then  $\neg\Phi \in Fr(\Sigma^L)$
4. If  $\Phi \in Fr(\Sigma^L)$ ,  $v_i \in V$ ,  $x_i \in V$ ,  $v_i$  are distinct,  $\xi(v_i) = \xi(x_i)$ ,  $i = 1, \dots, n$ ,  $n \geq 0$   
then  $R_{x_1, \dots, x_n}^{v_1, \dots, v_n} \Phi \in Fr(\Sigma^L)$
5. If  $\Phi \in Fr(\Sigma^L)$ ,  $x \in V$  then  $\exists x \Phi \in Fr(\Sigma^L)$
6. If  $x, y \in V$ ,  $\xi(x) = \xi(y)$  then  $=_{xy} \in Fr(\Sigma^L)$

Formulas of the form  $P$  are atomic formulas ( $P \in Ps$ ). Following conventional notation, we write  $x = y$  instead of  $=_{xy}$ .

*Interpretational component.* Let  $I^S : S \xrightarrow{t} T$  be a sort interpretation mapping. Given  $\xi$ , this mapping specifies a type valuation mapping  $\tau = I^S \circ \xi$ , and consequently an algebra of quasiary predicates over typed nominative sets  $A^S(V, T, \tau) = \langle Pr(V, T, \tau), C(V, T, \tau) \rangle$ .

Composition symbols have fixed interpretation, but we additionally need interpretation  $I^{Ps} : Ps \xrightarrow{t} Pr(\tau)$  of predicate symbols to obtain a language interpretation. A corresponding tuple  $J = (\Sigma^S, I^S, I^{Ps})$  is called an *L-interpretation*.

Given a formula  $\Phi$  and an interpretation  $J$  we can speak of an interpretation of  $\Phi$  in  $J$ . It is denoted by  $\Phi_J$  or  $J(\Phi)$ .

For different equivalent transformations of  $L$ -formulas we need unessential variables (analogs of fresh variables in classical logic) that do not affect the formula meanings [10]. Therefore we assume that there is a set  $U$  of unessential typed variables,  $U \subseteq V$  such that  $U$  contains an infinite number of variables of each sort. By calling a variable  $u \in U$  unessential we restrict considered interpretations of predicate symbols to such that are neither sensitive to the value of the component with the name  $u$  in nominative sets, nor to presence of such a component. Formally, a variable  $u \in V$  is *unessential in an interpretation*  $I^{Ps}$  if  $I^{Ps}(P)(d) = I^{Ps}(P)(d \nabla u \mapsto a)$  for all  $P \in Ps$ ,  $d \in NST(\tau)$ ,  $a \in \tau(u)$ .



### 3.3 Many-Sorted First-Order Classical Pure Predicate Logic $L_C$

*Syntactic component.* Let  $\Sigma^S = (V, S, \xi)$  be a signature of sort valuation. Let  $Cs_C(\Sigma^S)$  be a set of composition symbols,  $Cs_C(\Sigma^S) = \{\vee, \neg, \exists x, =_{xy}\}$ . Note, that renomination composition is not explicitly used in classical logic, thus  $Cs_C(\Sigma^S) \subset Cs(\Sigma^S)$ .

Let  $Ps$  be a set of predicate symbols and  $arity: Ps \xrightarrow{t} \bigcup_{n \geq 0} S^n$  be a sort arity mapping. It maps each predicate symbol  $P$  to a tuple  $(s_1, \dots, s_n)$ ,  $s_i \in S, i = 1, \dots, n, n \geq 0$  that represents sorts of its arguments.

A tuple  $\Sigma^{LC} = (\Sigma^S, Cs_C(\Sigma^S), Ps, arity)$  is a signature of a language of  $L_C$ . Given this signature we define the language of  $L_C$  – the set of formulas  $Fr(\Sigma^{LC})$  – inductively:

1. If  $P \in Ps$ ,  $arity(P) = (s_1, \dots, s_n)$ ,  $x_1, \dots, x_n \in V$ ,  $\xi(x_i) = s_i$ ,  $i = 1, \dots, n$ ,  $n \geq 0$  then  $P(x_1, \dots, x_n) \in Fr(\Sigma^{LC})$
2. If  $\Phi, \Psi \in Fr(\Sigma^{LC})$  then  $(\Phi \vee \Psi) \in Fr(\Sigma^{LC})$
3. If  $\Phi \in Fr(\Sigma^{LC})$  then  $\neg \Phi \in Fr(\Sigma^{LC})$
4. If  $\Phi \in Fr(\Sigma^{LC})$ ,  $x \in V$  then  $\exists x \Phi \in Fr(\Sigma^{LC})$
5. If  $x, y \in V$ ,  $\xi(x) = \xi(y)$  then  $(x = y) \in Fr(\Sigma^{LC})$

Formulas defined according to rules 1 and 5 are called *atomic*.

*Interpretational component.* Let  $I^S: S \xrightarrow{t} T$  be a sort interpretation mapping. Define  $\tau = I^S \circ \xi$  and consider an algebra of quasiary predicates over typed nominative sets  $A^S(V, T, \tau) = \langle Pr(V, T, \tau), C(V, T, \tau) \rangle$ . We assume that types are not empty. Formulas  $Fr(\Sigma^{LC})$  are interpreted as predicates in such algebras. Atomic formulas of the form  $x=y$  are interpreted as predicates  $=_{xy}$ . To give an interpretation of atomic formulas of the form  $P(x_1, \dots, x_n)$  we need to specify an interpretational mapping for predicate symbols. In case of classical logic it is specified by a mapping  $I_C^{Ps}$  that associates each predicate symbol  $P \in Ps$  of arity  $(s_1, \dots, s_n)$  with a total  $n$ -ary predicate  $I_C^{Ps}(P): I^S(s_1) \times \dots \times I^S(s_n) \xrightarrow{t} Bool$ . Thus,  $L_C$ -interpretations have the form  $J_C = (\Sigma^S, I^S, I_C^{Ps})$ . Such interpretation  $J_C$  for every atomic formula  $P(x_1, \dots, x_n)$  defines its meaning in  $Pr(\tau)$  as a predicate  $P(x_1, \dots, x_n)_J$  such that  $P(x_1, \dots, x_n)_J(d) = I_C^{Ps}(P)(d(x_1), \dots, d(x_n))$  for every  $d \in NST(\tau)$ ; if one of the values  $d(x_1), \dots, d(x_n)$  is not defined then  $P(x_1, \dots, x_n)_J$  is undefined on  $d$ . Let us note that in classical logic  $d$  is

called *variable valuation* or *variable assignment*. The meaning  $\Phi_J \in Pr(\tau)$  of a complex formula  $\Phi \in Fr(\Sigma^{LC})$  is defined in a usual way.

For the logics  $L$  and  $L_C$  derived compositions (such as conjunction  $\&$ , universal quantification  $\forall x$ , negated equality  $\neq_{xy}$ , etc.) are defined in a traditional way. In the sequel we consider formulas in their traditional form using infix operations and brackets; brackets can be omitted according to conventional rules for the priorities of operations (priority of the binary disjunction is weaker than priority of unary operations).

Formulas and interpretations in logics  $L$  and  $L_C$  are called *L-formulas* / *L<sub>C</sub>-formulas* and *L-interpretations* / *L<sub>C</sub>-interpretations* respectively.

Notions of satisfiability and validity can be defined in a uniform way for both logics, therefore their definitions are given only for the logic  $L$ .

A formula  $\Phi$  is called *satisfiable in an L-interpretation J*, if there is a nominative set  $d \in NST(V, T, \tau)$  such that  $\Phi_J(d) \downarrow = T$ . We shall denote this by  $J \approx \Phi$ . A formula  $\Phi$  is called *satisfiable in a logic L* if there is an  $L$ -interpretation  $J$  such that  $J \approx \Phi$ . We shall denote this  $\models_L \Phi$ , or  $\models \Phi$  if  $L$  is understood from the context.

A formula  $\Phi$  is called *valid in an L-interpretation J* if there is no  $d \in NST(V, T, \tau)$  such that  $\Phi_J(d) \downarrow = F$ . This is denoted  $J \models \Phi$ , which means that  $\Phi$  is not refutable in  $J$ . A formula  $\Phi$  is called *valid in logic L* if  $J \models \Phi$  for every  $L$ -interpretation  $J$ . We shall denote this  $\models_L \Phi$ , or just  $\models \Phi$  if  $L$  is understood from the context.

Formulas  $\Phi$  and  $\Psi$  are called *equisatisfiable* in a logic  $L$  if they are either both satisfiable or both not satisfiable (i.e. unsatisfiable) in  $L$ .

We call formulas  $\Phi$  and  $\Psi$  *equivalent* if  $\Phi_J = \Psi_J$  for every interpretation  $J$ .

In classical first-order logics we have the property that  $\Phi$  is satisfiable if  $\Phi$  is valid. This is not the case in CNL due to possible presence of a nowhere defined predicate (which is valid). Still satisfiability and validity are related notions in CNL: formula  $\Phi$  is satisfiable in an  $L$ -interpretation  $J$  iff  $\neg\Phi$  is not valid in  $J$ . This property permits to consider satisfiability and validity as dual properties.

*Satisfiability problem* for composition nominative logics considered in the chapter consists in checking whether or not given  $L$ -formula is satisfiable. *Validity problem* consists in checking whether or not given  $L$ -formula is valid.

## 4 Reduction of Satisfiability and Validity Problems

In this section we aim to reduce the satisfiability and validity problems for many-sorted composition-nominative pure predicate logic to the same problems for many-sorted first-order classical pure predicate logic.

Consider an arbitrary formula  $\Phi \in Fr(\Sigma^L)$ . We transform it to a formula  $\Phi_C \in Fr(\Sigma^{LC})$ , which is satisfiable in  $L_C$  if and only if the formula  $\Phi$  is satisfiable in  $L$ .

Given a formula  $\Phi$ , we first construct its unified renominative normal form denoted  $urnf[\Phi]$ . This form has several properties outlined below.

An  $L$ -formula  $\Phi$  is said to be in *unified renominative normal form* (URNF) if the following requirements are met:

- Renomination compositions are only applied in  $\Phi$  to predicate symbols. That is, for every subformula of the form  $R_{\bar{x}}^{\bar{v}} \Psi$  we have that  $\Psi \in Ps$
- Every occurrence of a predicate symbol  $P \in Ps$  is a part of some subformula of the form  $R_{\bar{q}}^{\bar{u}} P$  (possibly with empty lists of names in renomination)
- For every two subformulas  $R_{\bar{q}}^{\bar{u}} P$  and  $R_{\bar{y}}^{\bar{w}} Q$  we have that vectors  $\bar{u}$  and  $\bar{w}$  coincide; it means that for all instances of renomination composition the lists of upper names are the same
- For every subformula  $R_{\bar{x}}^{\bar{v}} P$  and every quantifier  $\exists y$  that occur in the formula  $\Phi$  we have that  $y \in \bar{v}$

We use notation  $y \in \bar{v}$  ( $y \in \{ \bar{v}, \bar{x} \}$ ) to indicate that  $y$  is among the variables of the list  $\bar{v}$  (lists  $\bar{v}$  and  $\bar{x}$ ).

When a formula is in URNF we call every its subformula of the form  $R_{\bar{x}}^{\bar{v}} P$  a *renominative atom*. Note that if a formula is in URNF then every its subformula is also in URNF.

**Lemma 1.** There is a (non-deterministic) algorithm that given an arbitrary formula  $\Phi \in Fr(\Sigma^L)$  constructs its equisatisfiable unified renominative normal form  $urnf[\Phi] \in Fr(\Sigma^L)$ .

*Proof.* Consider the following transformation rules (T1–T9) of the form  $\Phi_l \mapsto \Phi_r$ , where  $\Phi_l, \Phi_r \in Fr(\Sigma^L)$ :

T1)  $R_{\bar{x}}^{\bar{v}} =_{xy} \mapsto =_{\tilde{x}\tilde{y}}$  where  $\tilde{x} = x(\bar{v}/\bar{x})$ ,  $\tilde{y} = y(\bar{v}/\bar{x})$  (see T4 for definitions);

T2)  $R_{\bar{x}}^{\bar{v}} (\Phi_1 \vee \Phi_2) \mapsto (R_{\bar{x}}^{\bar{v}} \Phi_1 \vee R_{\bar{x}}^{\bar{v}} \Phi_2)$ ;

T3)  $R_{\bar{x}}^{\bar{v}} \neg \Phi \mapsto \neg R_{\bar{x}}^{\bar{v}} \Phi$ ;

T4)  $R_{x_1, \dots, x_n, y_1, \dots, y_m}^{v_1, \dots, v_n, w_1, \dots, w_m} R_{s_1, \dots, s_n, z_1, \dots, z_k}^{v_1, \dots, v_n, u_1, \dots, u_k} \Phi \mapsto R_{\alpha_1, \dots, \alpha_n, \gamma_1, \dots, \gamma_m, \beta_1, \dots, \beta_k}^{v_1, \dots, v_n, w_1, \dots, w_m, u_1, \dots, u_k} \Phi$  where

$w_i \neq u_j$  ( $i = 1, \dots, m; j = 1, \dots, k$ ),  $\alpha_i = s_i(v_1, \dots, v_n, w_1, \dots, w_m / x_1, \dots, x_n, y_1, \dots, y_m)$ ,  $\beta_j = z_j(v_1, \dots, v_n, w_1, \dots, w_m / x_1, \dots, x_n, y_1, \dots, y_m)$ . Here  $r(b_1, \dots, b_q / c_1, \dots, c_q) = r$  if  $r \notin \{b_1, \dots, b_q\}$ ,  $r(b_1, \dots, b_q / c_1, \dots, c_q) = c_i$  if  $r = b_i$  for some  $i$ ;

T5)  $R_{\bar{x}}^{\bar{v}} \exists y \Phi \mapsto \exists y R_{\bar{x}}^{\bar{v}} \Phi$ , when  $y \notin \{ \bar{v}, \bar{x} \}$ ;

T6)  $R_{z, \bar{x}}^{y, \bar{v}} \exists y \Phi \mapsto \exists y R_{\bar{x}}^{\bar{v}} \Phi$ ;

T7)  $R_{y, \bar{x}}^{z, \bar{v}} \exists y \Phi \mapsto \exists u R_{y, \bar{x}}^{z, \bar{v}} R_u^y \Phi$ ,  $u \in U$ ,  $\zeta(u) = \zeta(y)$ ,  $u$  does not occur in the formula on the left hand side of the rule;

T8)  $R_{\bar{q}}^{\bar{u}} P \mapsto R_{z, \bar{q}}^{z, \bar{u}} P$  (in case when vectors  $\bar{u}, \bar{v}$  are empty this rule is represented as  $P \mapsto R_z^z P$ );

T9)  $R_{q_1, \dots, q_i, \dots, q_j, \dots, q_n}^{u_1, \dots, u_i, \dots, u_j, \dots, u_n} P = R_{q_1, \dots, q_j, \dots, q_i, \dots, q_n}^{u_1, \dots, u_j, \dots, u_i, \dots, u_n} P$ .

The rule T4 represents explicitly the result of functional composition of parameters of two successive renominations. The rule T7 permits to assume without loss of generality that all quantified variables in initial formula are different.

The rules T1–T9 are equivalent transformations in  $L$  [10]. Using these rules the formula can be transformed to renominative normal form. First, using rules T1–T7 we can push renomination composition down to atomic formulas. Then, using rules T8, T9 we unify all renominative atoms so that the lists of upper names (variables) coincide. Obtained formula will be equivalent (and therefore equisatisfiable) with initial formula.  $\square$

Once  $L$ -formulas are in unified renominative normal form, we can apply a syntactical reduction  $mclf$ , which transforms them into  $L_C$ -formulas,  $mclf : Fr(\Sigma^L) \xrightarrow{t} Fr(\Sigma^{L_C})$ . This reduction is formalized inductively as follows:

1.  $mclf[=_{xy}] = (x = y)$
2.  $mclf[R_{x_1, \dots, x_n}^{v_1, \dots, v_n} P] = P(x_1, \dots, x_n)$
3.  $mclf[\neg \Phi] = \neg mclf[\Phi]$
4.  $mclf[(\Phi_1 \vee \Phi_2)] = (mclf[\Phi_1] \vee mclf[\Phi_2])$
5.  $mclf[\exists x \Phi] = \exists x(x \neq e_s \ \& \ mclf[\Phi]), \xi(x) = s$

For the rule 5 we specify that for each sort  $s \in S$  there is a predefined unessential variable of the same sort  $e_s \in U$ ,  $\xi(e_s) = s$  that does not occur in original formula. All applications of the 5-th rule introduce the same variable  $e_s$  when processing quantifier instances over variables of the same sort  $s$ .

Due to properties of URNF, applications of the rule 2 create instances of predicates of the same arity.

Now we can proceed with reducing the satisfiability problem in logic  $L$  to the satisfiability problem in classical logic  $L_C$ .

**Lemma 2.** Let  $\Phi$  be an  $L$ -formula in unified renominative normal form,  $\Phi \in Fr(\Sigma^L)$ ,  $J$  be an  $L$ -interpretation such that  $J \models \Phi$ . Then there is an  $L_C$ -interpretation  $J_C$  such that  $J_C \models mclf[\Phi]$ .

*Proof.* Let  $\Sigma^L = (\Sigma^S, Cs(\Sigma^S), Ps)$ ,  $\Sigma^S = (V, S, \xi)$ ,  $J = (\Sigma^S, I^S, I^{Ps})$ ,  $I^{Ps} : Ps \xrightarrow{t} Pr(\tau)$ ,  $\tau = I^S \circ \xi$ ,  $\xi : V \xrightarrow{t} S$ ,  $I^S : S \xrightarrow{t} T$ . Due to monotonicity of compositions in considered CNL, if a formula is satisfiable in  $L$  then it is also satisfiable

on some total  $L$ -interpretation [12]. Total interpretations mean that predicate symbols are interpreted in the class  $NST(V, T, \tau) \xrightarrow{t} Bool$  of total predicates. Therefore without loss of generality we assume that  $I^{Ps}$  maps predicate symbols  $Ps$  to total predicates.

To construct an interpretation  $J_C$  we consider algebras of quasiary predicates over typed nominative sets defined over extended types. Such an algebra, denoted by  $A^S(V, T_\varepsilon, \tau_\varepsilon)$ , is constructed from the algebra  $A^S(V, T, \tau)$ .

Let  $\varepsilon$  be a new element such that  $\varepsilon \notin \bigcup_{A \in T} A$ . For  $A \in T$  define  $A_\varepsilon = A \cup \{\varepsilon\}$ . Let

$T_\varepsilon = \{A_\varepsilon \mid A \in T\}$  and  $I_\varepsilon^S : S \xrightarrow{t} T_\varepsilon$  be such a sort interpretation that  $I_\varepsilon^S(s) = I^S(s) \cup \{\varepsilon\}$  for all  $s \in S$ . Let  $\tau_\varepsilon = I_\varepsilon^S \circ \xi$ .

Now we define the class of total nominative sets by the following formula:

$$NSTT(V, T_\varepsilon, \tau_\varepsilon) = \left\{ d : V \xrightarrow{t} \bigcup_{A_\varepsilon \in T_\varepsilon} A_\varepsilon \mid \forall v \in V (d(v) \in \tau_\varepsilon(v)) \right\}.$$

Denote by  $V_\Phi \subseteq V$  the set of all variables that occur as upper names in renominative atoms of  $\Phi$ .

From  $J \models \Phi$  follows that  $\Phi_J(d_0) \downarrow = T$  for some  $d_0 \in NST(V, T, \tau)$ . As  $e_s \in U$  for all  $s \in S$  we can assume without loss of generality that for all  $s \in S$   $d_0(e_s) \uparrow$ . Let us denote  $\Phi_C = mclf[\Phi]$ ,  $\Phi_C \in Fr(\Sigma^{LC})$ ,  $\Sigma^{LC} = (\Sigma^S, C_{SC}(\Sigma^S), Ps, arity)$ . We will now construct an interpretation  $J_C$  of the form  $J_C = (\Sigma^S, I_\varepsilon^S, I_C^{Ps})$ , where  $I_C^{Ps}$  is described below, such that  $J_C \models \Phi_C$ .

Let us first define the mapping  $\varepsilon^+ : NST(V, T, \tau) \xrightarrow{t} NSTT(V, T_\varepsilon, \tau_\varepsilon)$  by the formula  $\varepsilon^+(d) = [v \mapsto \varepsilon \mid v \in V] \nabla d$ . Informally, this mapping adds a pair  $v \mapsto \varepsilon$  to the nominative set  $d$  for every name  $v$  that is undefined in  $d$ .

It is clear that  $\varepsilon^+$  is bijective. We denote the inverse mapping  $\varepsilon^- : NSTT(V, T_\varepsilon, \tau_\varepsilon) \xrightarrow{t} NST(V, T, \tau)$ ,  $\varepsilon^-(d) = [v \mapsto d(v) \mid v \in V, d(v) \neq \varepsilon]$ . Informally, this mapping deletes all pairs  $v \mapsto \varepsilon$  from the nominative set  $d$ .

Let  $(v_1, \dots, v_n)$  be a list of upper names of renominative atoms occurring in  $\Phi$ . Then for every  $P \in Ps$  of arity  $(\xi(v_1), \dots, \xi(v_n))$  we construct a total predicate  $I_C^{Ps}(P) = p, p : \tau_\varepsilon(v_1) \times \dots \times \tau_\varepsilon(v_n) \xrightarrow{t} Bool$  in the following way:

$$p(a_1, \dots, a_n) = I^{Ps}(P) ([v \mapsto a \mid v \mapsto a \in_n d_0, v \notin \{v_1, \dots, v_n\}] \nabla [v_i \mapsto a_i \mid i = 1, \dots, n, a_i \neq \varepsilon]).$$

This completes definition of  $L_C$ -interpretation  $J_C$ .

Let  $D = \{\varepsilon^-(\varepsilon^+(d_0)\nabla d) \mid d \in NSTT(V_\Phi, T_\varepsilon, \tau_\varepsilon)\}$ . The set  $D$  contains nominative sets that are variations of  $d_0$  for only those names that occur in upper names of renominative atoms in  $\Phi$ . The set of such names is finite and their values determine satisfiability of  $\Phi$ ; this fact permits to prove that satisfiability of constructed classical formula is preserved.

Now we prove by induction that for every subformula  $\Psi$  of  $\Phi$  and its counterpart  $\Psi_C = mclf[\Psi]$  we have that  $\Psi_J(d) = (\Psi_C)_{J_C}(\varepsilon^+(d))$  for each  $d \in D$ .

Base of induction: we have that  $(R_{x_1, \dots, x_n}^{v_1, \dots, v_n} P)_J(d) = (P(x_1, \dots, x_n))_{J_C}(\varepsilon^+(d))$  by construction of  $I_C^{Ps}$  and  $(=_{xy})_J(d) = (x = y)_{J_C}(\varepsilon^+(d))$  by definition of  $L_C$ .

We prove the induction statement only for the non-trivial case of existential quantifier composition.

Let  $\Psi = \exists x \Theta$ , then  $\Psi_C = mclf[\Psi] = \exists x (x \neq e_{\xi(x)} \& \Theta_C)$ , where  $\Theta_C = mclf[\Theta]$ . By induction base we have that for every  $d \in D$   $\Theta_J(d) = (\Theta_C)_{J_C}(\varepsilon^+(d))$ . Let us prove that for every  $d \in D$   $\Psi_J(d) = (\Psi_C)_{J_C}(\varepsilon^+(d))$ .

Suppose  $\Psi_J(d) = T$ . That is:  $(\exists x \Theta)_J(d) = T$ . It means there is some  $a \in \tau(x)$  such that  $\Theta_J(d\nabla x \mapsto a) = T$ . As  $x \in V_\Phi$  we have that  $d \in D \Rightarrow d\nabla x \mapsto a \in D$ . That means, according to induction assumption, that  $(\Theta_C)_{J_C}(\varepsilon^+(d\nabla x \mapsto a)) = T$ .

Note that  $\varepsilon^+(d\nabla x \mapsto a) = \varepsilon^+(d)\nabla x \mapsto a$ . That is,  $(\Theta_C)_{J_C}(\varepsilon^+(d)\nabla x \mapsto a) = T$ . For every  $d \in D$  we have  $\varepsilon^+(d)(e_{\xi(x)}) = \varepsilon$ ; we also have by construction that  $e_{\xi(x)}$  does not occur in  $\Psi$  and  $a \neq \varepsilon$ . Hence  $\varepsilon^+(d\nabla x \mapsto a)(x) \neq \varepsilon^+(d\nabla x \mapsto a)(e_{\xi(x)})$  and  $(x \neq e_{\xi(x)})_{J_C}(\varepsilon^+(d)\nabla x \mapsto a) = T$ . Therefore,  $(\exists x (x \neq e_{\xi(x)} \& \Theta_C))_{J_C}(\varepsilon^+(d)) = T$ . So we have obtained that the implication  $\Psi_J(d) = T \Rightarrow (\Psi_C)_{J_C}(\varepsilon^+(d)) = T$  holds.

Suppose that  $\Psi_J(d) = F$ . That is:  $(\exists x \Theta)_J(d) = F$ . Then we have that for all  $a \in \tau(x)$   $\Theta_J(d\nabla x \mapsto a) = F$ . That means, according to induction assumption, that for all  $a \in \tau(x)$   $(\Theta_C)_{J_C}(\varepsilon^+(d\nabla x \mapsto a)) = F$ . Let us assume that  $(\Theta_C)_{J_C}(\varepsilon^+(d)\nabla x \mapsto b) = T$  for some  $b \in \tau_\varepsilon(x)$ . It then follows immediately that  $b = \varepsilon$ , which means that  $(x \neq e_{\xi(x)} \& \Theta_C)_{J_C}(\varepsilon^+(d)\nabla x \mapsto b) = F$  for all  $b \in \tau_\varepsilon(x)$ , which means that  $(\exists x (x \neq e_{\xi(x)} \& \Theta_C))_{J_C}(\varepsilon^+(d)) = F$ . Thus the implication  $\Psi_J(d) = F \Rightarrow (\Psi_C)_{J_C}(\varepsilon^+(d)) = F$  holds.

Hence, from  $\Phi_J(d_0) \downarrow = T$  we obtain that  $(\Phi_C)_{J_C}(\varepsilon^+(d_0)) \downarrow = T$ . □

Now we can prove the inverse of lemma 2: satisfiability of  $mclf[urnf[\Phi]]$  in classical logic  $L_C$  implies satisfiability of  $\Phi$  in  $L$ . The idea is the following: having an interpretation in  $L_C$  we construct a new interpretation in  $L$  deleting values of variables  $e_s$  for all  $s \in S$ . But it may happen that from one type we delete more than one value. This can violate satisfiability of  $\Phi$  in the obtained interpretation. Therefore we first should construct an interpretation in which types are disjoint. This can be formalized by the following notion: a sort interpretation mapping  $I^S : S \xrightarrow{t} T$  is called *type-disjoint* if for every  $s_1, s_2 \in S$ ,  $s_1 \neq s_2$  we have that  $I^S(s_1) \cap I^S(s_2) = \emptyset$ . Then having a type disjoint interpretation we can delete from types values of  $e_s$  for all  $s \in S$ . For such interpretations we will delete one value only from each type. The both constructions described preserve satisfiability (lemmas 3 and 4).

**Lemma 3.** Let  $\Phi \in Fr(\Sigma^L)$ ,  $J = (\Sigma^S, I^S, I^{Ps})$  be an  $L$ -interpretation such that  $J \models \Phi$ . Then there is an  $L$ -interpretation  $J_{\#} = (\Sigma^S, I_{\#}^S, I_{\#}^{Ps})$  such that  $J_{\#} \models \Phi$  and  $I_{\#}^S$  is type-disjoint.

*Proof.* Let  $\Sigma^L = (\Sigma^S, Cs(\Sigma^S), Ps)$ ,  $\Sigma^S = (V, S, \xi)$ ,  $I^{Ps} : Ps \xrightarrow{t} Pr(\tau)$ ,  $\tau = I^S \circ \xi$ ,  $\xi : V \xrightarrow{t} S$ ,  $I^S : S \xrightarrow{t} T$ . Given  $s \in S, A \in T$  define  $A^s = \{(s, a) \mid a \in A\}$  and  $T_{\#} = \{A^s \mid A \in T, s \in S\}$ . Then define  $I_{\#}^S : S \xrightarrow{t} T_{\#}$  by the formula  $I_{\#}^S(s) = A^s$  where  $A = I^S(s)$ . It is easy to see that  $I_{\#}^S$  is type-disjoint. Let  $\tau_{\#} = I_{\#}^S \circ \xi$ . Consider mapping  $\# : NST(V, T, \tau) \xrightarrow{t} NST(V, T_{\#}, \tau_{\#})$  defined by the formula  $\#(d) = [\nu \mapsto (\xi(\nu), a) \mid \nu \mapsto a \in_n d]$ ,  $d \in NST(V, T, \tau)$ . Obviously,  $\#$  is a bijection. Define  $I_{\#}^{Ps} : Ps \xrightarrow{t} Pr(\tau_{\#})$  by the formula  $I_{\#}^{Ps}(P)(\#(d)) = I^{Ps}(P)(d)$ ,  $P \in Ps$ ,  $d \in NST(V, T, \tau)$ . The equality in the formula is a strong equality. Let  $J_{\#} = (\Sigma^S, I_{\#}^S, I_{\#}^{Ps})$ . By induction on the structure of  $\Phi$  we can prove that  $J(\Phi)(d) = J_{\#}(\Phi)(\#(d))$ ,  $d \in NST(V, T, \tau)$ . From this follows the statement of the lemma.  $\square$

It is easy to check that an equivalent lemma also holds for  $L_C$ -formulas and  $L_C$ -interpretations.

**Lemma 4.** Let  $\Phi$  be an  $L$ -formula in unified renominative normal form,  $\Phi \in Fr(\Sigma^L)$ ,  $J_C$  be an  $L_C$ -interpretation such that  $J_C \models mclf[\Phi]$ . Then there is an  $L$ -interpretation  $J$  such that  $J \models \Phi$ .

*Proof.* Let  $\Sigma^{LC} = (\Sigma^S, Cs_C(\Sigma^S), Ps, arity)$ ,  $J_C = (\Sigma^S, I_C^S, I_C^{Ps})$ ,  $\Sigma^S = (V, S, \xi)$ ,  $\xi : V \xrightarrow{t} S$ ,  $I_C^S : S \xrightarrow{t} T_C$ ,  $\tau_C = I_C^S \circ \xi$ ,  $I_C^{Ps}$  be an interpretation of predicate

symbols to total  $n$ -ary predicates. Due to lemma 3 we can assume that  $I_C^S : S \xrightarrow{t} T_C$  is type-disjoint.

Let  $\Phi_C = mclf[\Phi]$ ,  $(\Phi_C)_{J_C}(d_C) = T$  for some  $d_C \in NSTT(V, T_C, \tau_C)$ .

According to  $mclf$  transformation, for each sort  $s$  in  $S$  we have a predefined variable  $e_s$ . Let  $T$  be the class of sets that consists of the sets  $I_C^S(s) \setminus \{d_C(e_s)\}$  for all sorts  $s \in S$ . Now we construct a sort interpretation  $I^S : S \xrightarrow{t} T$  such that for all  $s \in S$   $I^S(s) = I_C^S(s) \setminus \{d_C(e_s)\}$ . Let  $\tau : V \xrightarrow{t} T$ ,  $\tau = I^S \circ \xi$ . This definition is correct since  $I_C^S$  is type-disjoint and therefore, injective.

Let us define the mappings  $\delta^+ : NST(V, T, \tau) \xrightarrow{t} NSTT(V, T_C, \tau_C)$  and

$\delta^- : NSTT(V, T_C, \tau_C) \xrightarrow{t} NST(V, T, \tau)$  by the following formulas:

$$\delta^+(d) = [v \mapsto d_C(e_{\xi(v)}) \mid v \in V] \nabla d, \quad \delta^-(d) = [v \mapsto d(v) \mid v \in V, d(v) \neq d_C(e_{\xi(v)})].$$

Obviously, these mappings are bijections.

Finally, let us construct the interpretation of predicate symbols  $I^{Ps} : Ps \xrightarrow{t} Pr(\tau)$  and a nominative set  $d_0 \in NST(V, T, \tau)$  such that  $\Phi_J(d_0) \downarrow = T$  where  $J = (\Sigma^S, I^S, I^{Ps})$ .

Let  $D_C = \{d_C \nabla d \mid d \in NSTT(V_\Phi, T_C, \tau_C)\}$ . For each predicate symbol  $P$  occurring in  $\Phi$  within renominative atoms with  $(v_1, \dots, v_n)$  as upper names and for every  $d \in NST(V, T, \tau)$  we assign  $I^{Ps}(P)(d) = (P(v_1, \dots, v_n))_{J_C}(\delta^+(d))$ . Now we can prove that for every subformula  $\Psi$  of  $\Phi$  and its counterpart  $\Psi_C = mclf[\Psi]$  we have that  $\Psi_J(\delta^-(d)) = (\Psi_C)_{J_C}(d)$  for each  $d \in D_C$ . The proof scheme is the same as that of lemma 2.

Assign  $d_0 = \delta^-(d_C)$ . Then we obtain that  $\Phi_J(d_0) = T$ . □

**Theorem 1.** Let  $\Phi \in Fr(\Sigma^L)$ . Then  $\models_L \Phi$  if and only if  $\models_{L_C} mclf[urnf[\Phi]]$ .

Proof of the theorem follows from lemmas 2–4.

**Theorem 2.** Let  $\Phi \in Fr(\Sigma^L)$ . Then  $\models_L \Phi$  if and only if  $\models_{L_C} mclf[urnf[\Phi]]$ .

*Proof.* Duality of satisfiability and validity means that  $\models_L \Phi$  iff  $\neg\Phi$  is not satisfiable in  $L$ . From the theorem 1 follows that  $\neg\Phi$  is not satisfiable in  $L$  iff  $mclf[urnf[\neg\Phi]]$  is not satisfiable in  $L_C$ . This means that  $\models_{L_C} \neg mclf[urnf[\neg\Phi]]$ . By inspections of rules for  $urnf$  and  $mclf$  we can conclude that negation is distributive with respect to these transformations. Thus,

$$\models_{L_C} \neg mclf[urnf[\neg\Phi]] \Leftrightarrow \models_{L_C} \neg \neg mclf[urnf[\Phi]] \Leftrightarrow \models_{L_C} mclf[urnf[\Phi]]. \quad \square$$



The results obtained state the reduction of satisfiability and validity problems in many-sorted composition-nominative pure predicate logic to the same problems in many-sorted first-order classical pure predicate logic.

Let us illustrate the proposed reduction methods on a simple example.

**Example.** Let us check satisfiability and validity of the formula  $\Phi$ ,

$$\Phi = \forall x \forall y \forall z ((\neq_{xy} \& \neq_{yz} \rightarrow =_{xz}) \& (R_c^b P \& \neg R_{d,c}^{a,b} \exists a P)),$$

in the logic  $L$  such that  $\Sigma^L = (\Sigma^S, Cs(\Sigma^S), Ps)$ ,  $\Sigma^S = (V, S, \xi)$ ,  $V = \{x, y, z, a, b, c, d\}$ ,  $S = \{s_1, s_2\}$ ,  $\xi(a) = \xi(b) = \xi(c) = \xi(d) = s_1$ ,  $\xi(x) = \xi(y) = \xi(z) = s_2$ .

Let us construct its unified renominative normal form  $\Phi_{UR}$ . First we push the renomination down according to the rule T6 and obtain the following formula:

$$\forall x \forall y \forall z ((\neq_{xy} \& \neq_{yz} \rightarrow =_{xz}) \& (R_c^b P \& \neg \exists a R_c^b P)).$$

Then we change the form of the renominative atoms due to quantifier occurrences, and get the following formula  $\Phi_{UR} = urnf[\Phi]$ ,

$$\Phi_{UR} = \forall x \forall y \forall z ((\neq_{xy} \& \neq_{yz} \rightarrow =_{xz}) \& (R_{a,c,x,y,z}^{a,b,x,y,z} P \& \neg \exists a R_{a,c,x,y,z}^{a,b,x,y,z} P)).$$

Note that we use derived transformation rules that handle compositions  $\&$  and  $\forall$ .

Now we can apply the *mclf* transformation and obtain:

$$\Phi_C = mclf[\Phi_{UR}] = \forall x \forall y \forall z ((x \neq e_{s_2} \rightarrow (y \neq e_{s_2} \rightarrow (z \neq e_{s_2} \rightarrow (x \neq y \& y \neq z \rightarrow x = z)))) \& P(a, c, x, y, z) \& \neg \exists a (a \neq e_{s_1} \& P(a, c, x, y, z))).$$

Formula  $\Phi_C$  is satisfiable in two-sorted logic  $L_C$ . Therefore  $\Phi$  is satisfiable in  $L$ .

Formula  $\Phi_C$  is not valid in two-sorted logic  $L_C$ . Therefore  $\Phi$  is not valid in  $L$ .

Indeed, let  $J = (\Sigma^S, I^S, I^{Ps})$  be an  $L$ -interpretation such that  $I^S(s_1) = \{1, 2, 3\}$ ,  $I^S(s_2) = \{\alpha, \beta\}$ . Let  $I^{Ps}(P)(d) \downarrow = F$  if a pair  $a \mapsto r \in_n d$  for some  $r \in I^S(s_1)$  and  $T$  in all other cases. In other words, the predicate  $P$  takes the value  $T$  on a nominative set  $d$  if the variable  $a$  is undefined in  $d$ . Now, for instance we have that  $\Phi_J([b \mapsto 1, c \mapsto 1]) \downarrow = T$  and  $\Phi_J([a \mapsto 1, b \mapsto 1, c \mapsto 1]) \downarrow = F$ .

## 5 Conclusions

Composition-nominative logics are grounded on the same methodological and mathematical basis as program models. Therefore these logics should reflect such program features as partiality, compositionality, nominativity, elaborated type system, etc. These new features of CNL complicate their investigation; therefore it seems reasonable to transfer results obtained in classical logic to CNL. In the chapter we have demonstrated this idea for satisfiability and validity problems for many-sorted

composition-nominative pure predicate logic. As a main result we have shown that the problems under consideration can be reduced to the same problems for many-sorted classical predicate logic with equality. Thus, existent state-of-the-art methods and techniques for checking satisfiability and validity in classical logics can also be applied to composition-nominative logics.

Future work on the topic will include investigation of satisfiability and validity problems for richer CNL, in particular, for predicate-function logics and for program logics of Floyd-Hoare style. Another direction is related to logics over hierarchic nominative data. Hierarchic data permit to represent such complex structures as lists, stacks, arrays etc; thus, such logics will be closer to program models with more rich data types. We also plan to inspect what methods and techniques for studying partiality, compositionality, nominativity, and satisfiability modulo theory [15–20] can be applied in composition-nominative logics.

## References

1. Mendelson, E.: *Introduction to Mathematical Logic*, 4th edn. Chapman & Hall, London (1997)
2. Kroening, D., Strichman, O.: *Decision Procedures – an Algorithmic Point of View*. Springer, Heidelberg (2008)
3. Marques-Silva, J.: *Practical Applications of Boolean Satisfiability*. In: *Workshop on Discrete Event Systems*, Goteborg, Sweden, May 28–30, pp. 74–80 (2008)
4. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: *Solving SAT and SAT modulo theories: from an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T)*. *J. ACM* 53, 937–977 (2006)
5. de Moura, L., Bjørner, N.: *Satisfiability Modulo Theories: Introduction and Applications*. *Comm. ACM* 54(9), 69–77 (2011)
6. Nikitchenko, N.S.: *A Composition Nominative Approach to Program Semantics*. Technical Report IT, TR 1998-020, Technical University of Denmark (1998)
7. Basarab, I.A., Gubsky, B.V., Nikitchenko, N.S., Red’ko, V.N.: *Composition Models of Databases*. In: Eder, J., Kalinichenko, L.A. (eds.) *East-West Database Workshop. Workshops in Computing Series*, pp. 221–231. Springer, London (1995)
8. Nielson, H.R., Nielson, F.: *Semantics with Applications: A Formal Introduction*. John Wiley & Sons Inc. (1992)
9. Nikitchenko, M.S.: *Composition-Nominative Aspects of Address Programming*. *Kibernetika I Sistemnyi Analiz* 6, 24–35 (2009) (in Russian)
10. Nikitchenko, M.S., Shkilnyak, S.S.: *Mathematical Logic and Theory of Algorithms*. Publishing House of Taras Shevchenko National University of Kyiv, Kyiv (2008) (in Ukrainian)
11. Nikitchenko, M.S., Tymofieiev, V.G.: *Satisfiability Problem in Composition-Nominative Logics*. In: *Proceedings of the Eleventh International Conference on Informatics INFORMATICS 2011*, Roznava, Slovakia, November 16–18, pp. 75–80 (2011)
12. Nikitchenko, M.S., Tymofieiev, V.G.: *Satisfiability in Composition-Nominative Logics*. *Central European Journal of Computer Science* (to appear)

13. Nikitchenko, M.S., Tymofieiev, V.G.: Satisfiability Problem in Composition-Nominative Logics of Quantifier-Equational Level. In: Proc. 8th Int. Conf. ICTERI 2012, Kherson, Ukraine, June 6-10, vol. 848. CEUR-WS.org (2012), <http://ceur-ws.org/Vol-848/ICTERI-2012-CEUR-WS-paper-38-p-56-70.pdf>
14. Winskel, G.: *The Formal Semantics of Programming Languages*. MIT Press (1993)
15. Blamey, S.: *Partial Logic*. In: Gabbay, D., Guentner, F. (eds.) *Handbook of Philosophical Logic*, vol. III. D. Reidel Publishing Company (1986)
16. Jones, C.B.: Reasoning About Partial Functions in the Formal Development of Programs. *ENTCS* 145, 3–25 (2006)
17. Owe, O.: Partial Logics Reconsidered: A Conservative Approach. *Form. Asp. Comput.* 5, 208–223 (1997)
18. Janssen, T.M.V.: Compositionality. In: van Benthem, J., ter Meulen, A. (eds.) *Handbook of Logic and Language*, pp. 417–473. Elsevier and MIT Press (1997)
19. Pitts, A.M.: Nominal Logic, A First Order Theory of Names and Binding. *Inform. Comput.* 186, 165–193 (2003)
20. Barrett, C., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability Modulo Theories. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) *Handbook of Satisfiability*. IOS Press (2009)

# A Criterion for Existence of Global-in-Time Trajectories of Non-deterministic Markovian Systems

Ievgen Ivanov<sup>1,2</sup>

<sup>1</sup> Taras Shevchenko National University of Kyiv, Ukraine

<sup>2</sup> Paul Sabatier University, Toulouse, France

ivanov.eugen@gmail.com

**Abstract.** We consider the following question: given a continuous-time non-deterministic (not necessarily time-invariant) dynamical system, is it true that for each initial condition there exists a global-in-time trajectory. We study this question for a large class of systems, namely the class of complete non-deterministic Markovian systems. We show that for this class our question can be answered using analysis of existence of locally defined trajectories in a neighborhood of each time moment.

**Keywords:** dynamical systems, non-deterministic systems, Markovian systems, global-in-time trajectories.

## 1 Introduction

In this chapter we consider the following question: given a continuous-time non-deterministic (not necessarily time-invariant) dynamical system, is it true that for each time moment  $t_0$  and initial state  $x_0$  there exists a global-in-time trajectory  $t \mapsto s(t)$  such that  $s(t_0) = x_0$ . Our goal is to study how this question is related to existence of locally defined trajectories.

For deterministic systems the existence of a global trajectory for each initial condition implies that each locally defined trajectory (e.g., on a finite interval of the real time scale) can be extended to a globally defined trajectory. This is not necessary for non-deterministic systems. For example, for each initial condition  $x(t_0) = x_0 > 0$  the differential inclusion  $\frac{dx}{dt} \in [0, x^2]$  has both a globally defined constant trajectory  $x(t) = x_0$  and a trajectory of the equation  $\frac{dx}{dt} = x^2$  which escapes to infinity in finite time and cannot be extended infinitely into future.

Some related problems, e.g., global existence of solutions of initial value problems for various classes of differential equations [1,2,3] and inclusions [4,5,6], existence of non-Zeno global-in-time executions of hybrid automata [7,8,9] are well known. However, most studies either consider a more strong property of non-deterministic systems than we do (e.g., whether for each initial condition every trajectory exists into future [4,5,6]), or are limited to deterministic systems (differential equations with unique solutions, deterministic hybrid automata, etc.).

We will study our existence question for a large class of systems, namely the class of complete non-deterministic Markovian systems. A formal definition of this class will be given in the next section, but note that the term Markovian here refers to purely non-deterministic (i.e., non-stochastic) systems. This class includes systems defined by continuous and discrete-continuous formalisms (differential inclusions, switched systems) which are used for modeling and analysis in natural sciences and technology.

The article is organized in the following way. In Section 2 we give a definition and consider examples and basic properties of non-deterministic complete Markovian systems. In Section 3 we consider our existence question for systems of this class and formulate a theorem which shows that it can be answered using independent analysis of existence of locally defined trajectories in a neighborhood of each time moment. In Section 4 we give a proof of this theorem.

## 2 Non-deterministic Complete Markovian Systems

The notions of a Markov process or system [10] are usually defined and studied in the context of probability theory. However, they also make sense in a purely non-deterministic setting, where no quantitative information is attached to events, trajectories, transitions, etc. General definitions of continuous-time Markovian systems of such kind have appeared in the literature [11]. They describe a large class of non-deterministic systems which can have both continuous and discontinuous (jump-like) trajectories. The idea is that the current state of a system contains all the information that is needed to characterize its possible future behaviours.

Below we define the notion of a non-deterministic (complete) Markovian system in spirit of, but not exactly as in [11]. The main reasons for this are that we would like to take into account non-time-invariant systems and focus on partial trajectories, i.e., trajectories defined on a subset of the time scale.

We will use the following notation:  $\mathbb{N} = \{1, 2, 3, \dots\}$ ,  $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ ,  $f : A \rightarrow B$  is a total function from  $A$  to  $B$ ,  $f : A \dashrightarrow B$  is a partial function from  $A$  to  $B$ ,  $f|_X$  is the restriction of a function  $f$  to a set  $X$ ,  $2^A$  is the power set of a set  $A$ . The notation  $f(x) \downarrow$  ( $f(x) \uparrow$ ) means that  $f$  is defined (resp. undefined) on the argument  $x$ ,  $dom(f) = \{x \mid f(x) \downarrow\}$ . Also,  $\neg$ ,  $\vee$ ,  $\wedge$ ,  $\Rightarrow$ ,  $\Leftrightarrow$  denote the logical operations of negation, disjunction, conjunction, implication and equivalence correspondingly. Let us denote:

- $T = [0, +\infty)$  is the (real) time scale. We assume that  $T$  is equipped with a topology induced by the standard topology on  $\mathbb{R}$ . We will use the same positive real time scale  $T$  throughout the chapter.
- $\mathfrak{T}$  is the set of all connected subsets of  $T$  with cardinality greater than one.

For the purpose of this chapter we will use the following definition of dynamical system on the time scale  $T$ .

**Definition 1.** *A dynamical system on  $T$  is an abstract object  $M$  (a mathematical model; in applications this may be an equation, inclusion, etc.) together with*

an associated time scale  $T$ , a set of states  $Q$ , and a set of (partial) trajectories  $Tr$ . A trajectory is a function  $s : A \rightarrow Q$ , where  $A \in \mathfrak{T}$  (note that trivial trajectories defined on singleton or empty time sets are excluded). The set  $Tr$  satisfies the property: if  $s : A \rightarrow Q \in Tr$ ,  $B \in \mathfrak{T}$ , and  $B \subseteq A$ , then  $s|_B \in Tr$ . We will refer to this property as "Tr is closed under proper restrictions (CPR)".

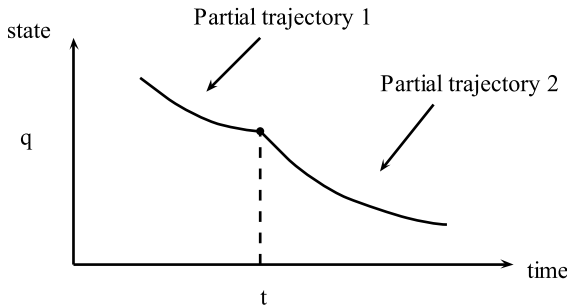
We will say that a trajectory  $s_1 \in Tr$  is a *subtrajectory* of  $s_2 \in Tr$  (denoted as  $s_1 \sqsubseteq s_2$ ), if  $dom(s_1) \subseteq dom(s_2)$  and  $s_1 = s_2|_{dom(s_1)}$ . The trajectories  $s_1$  and  $s_2$  are incomparable, if  $s_1$  is not a subtrajectory of  $s_2$  and vice versa.

According to the definition given above, for any  $t_0 \in T$  and  $q_0 \in Q$  there may exist multiple incomparable trajectories  $s$  such that  $s(t_0) = q_0$  (as well as one or none). In this sense a dynamical system can be non-deterministic. It is easy to see that  $(Tr, \sqsubseteq)$  is a (possibly empty) partially ordered set (poset).

**Definition 2.** A CPR set of trajectories  $Tr$  is

- (1) *complete*, if every non-empty chain in  $(Tr, \sqsubseteq)$  has a supremum ( $Tr$  may be empty).
- (2) *Markovian*, if  $s \in Tr$  for each  $s_1, s_2 \in Tr$  and  $t \in T$  such that  $t = \sup dom(s_1) = \inf dom(s_2)$ ,  $s_1(t) \downarrow$ ,  $s_2(t) \downarrow$ , and  $s_1(t) = s_2(t)$ , where
 
$$s(t) = \begin{cases} s_1(t), & t \in dom(A) \\ s_2(t), & t \in dom(B) \end{cases}.$$

Note that because of CPR property, the supremum of a chain  $c$  in the poset  $(Tr, \sqsubseteq)$  exists if and only if  $s_* \in Tr$ , where  $s_* : \bigcup_{s \in c} dom(s) \rightarrow Q$  is defined as follows:  $s_*(t) = s(t)$ , if  $s \in c$  and  $t \in dom(s)$  (this definition is correct, because  $c$  is a chain).



**Fig. 1.** Markovian property for non-deterministic systems. If one partial trajectory ends and another begins in state  $q$  at time  $t$  (both are defined at  $t$ ), then their concatenation is a partial trajectory.

**Proposition 1.** Let  $Q = \mathbb{R}$ . Consider the following sets of trajectories:

- $Tr_{all}$  is the set of all functions  $s : A \rightarrow Q$ ,  $A \in \mathfrak{T}$ .
- $Tr_{cont}$  is the set of all continuous functions  $s \in Tr_{all}$  (on their domains).

- $Tr_{diff}$  is the set of all differentiable functions  $s \in Tr_{all}$  (on their domains).
- $Tr_{bnd}$  is the set of all bounded functions  $s \in Tr_{all}$  (on their domains).

Then the following holds:

- (1)  $\emptyset, Tr_{all}, Tr_{cont}, Tr_{diff}, Tr_{bnd}, Tr_{diff} \cap Tr_{bnd}$  are CPR.
- (2)  $\emptyset, Tr_{all}, Tr_{cont}$  are complete and Markovian.
- (3)  $Tr_{diff}$  is complete, but is not Markovian.
- (4)  $Tr_{bnd}$  is Markovian, but is not complete.
- (5)  $Tr_{diff} \cap Tr_{bnd}$  is neither complete, nor Markovian.

The proof of this proposition follows from definitions.

**Definition 3.** A non-deterministic complete Markovian system (NCMS) is a dynamical system  $(M, T, Q, Tr)$  such that  $Tr$  is complete and Markovian.

The following simple propositions [24] give some examples of NCMS.

**Proposition 2.** Let  $Q = \mathbb{R}^d$  ( $d \in \mathbb{N}$ ) and  $M$  be a differential equation  $\frac{dy}{dt} = f(t, y)$ , where  $f : \mathbb{R} \times \mathbb{R}^d \rightarrow \mathbb{R}^d$  is a continuous function. Let  $Tr$  be the set of all functions  $s : A \rightarrow Q$ ,  $A \in \mathfrak{T}$  such that  $s$  is differentiable on  $A$  and satisfies  $M$  on  $A$ . Then  $(M, T, Q, Tr)$  is a NCMS.

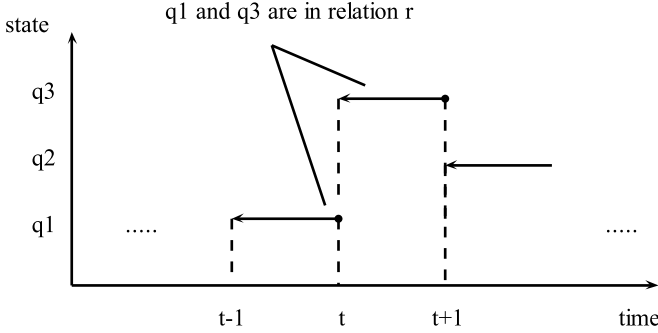
**Proposition 3.** Let  $M$  be a differential inclusion  $\frac{dy}{dt} = F(t, y)$ , where  $F : \mathbb{R} \times \mathbb{R}^d \rightarrow 2^{\mathbb{R}^d}$  is a given (total) function. Let  $M'$  be the system  $\begin{cases} \frac{dy}{dt} = x \\ y \in F(t, x) \end{cases}$ , where  $x$  is a new variable. Let  $Q = \mathbb{R}^d \times \mathbb{R}^d$  and  $Tr$  be the set of all  $s : A \rightarrow Q$ ,  $A \in \mathfrak{T}$  such that  $s$  is locally absolutely continuous on  $A$  and satisfies  $M'$  almost everywhere on  $A$  (w.r.t. Lebesgue's measure). Then  $(M, T, Q, Tr)$  is a NCMS.

**Proposition 4.** Let  $Q$  be a set equipped with discrete topology. Let  $r \subseteq Q \times Q$  be a relation on  $Q$ . Let  $M$  be a system  $\begin{cases} y(t+) = y(t), & t \notin \mathbb{N}_0 \\ (y(t), y(t+)) \in r, & t \in \mathbb{N}_0 \end{cases}$ , where  $y$  denotes an unknown function,  $y(t+)$  denotes the right limit at  $t$ . Let  $Tr$  be the set of all piecewise-constant left-continuous functions  $s : A \rightarrow Q$  (w.r.t. discrete topology on  $Q$ ) which satisfy  $M$  on  $A$  (see Fig. 2). Then  $(M, T, Q, Tr)$  is a NCMS.

Let us introduce the following terminology:

**Definition 4.** Let  $s_1, s_2 : T \dashrightarrow Q$ . Then  $s_1$  and  $s_2$  coincide:

- on a set  $A \subseteq T$ , if  $A \subseteq \text{dom}(s_1) \cap \text{dom}(s_2)$  and  $s_1(t) = s_2(t)$  for each  $t \in A$ . We denote this as  $s_1 \dot{=}_A s_2$ .
- in a left neighborhood of  $t \in T$ , if  $t > 0$  and there exists  $t' \in [0, t)$ , such that  $s_1 \dot{=}_{(t', t]} s_2$ . We denote this as  $s_1 \dot{=}_{t-} s_2$ .
- in a right neighborhood of  $t \in T$ , if there exists  $t' > t$ , such that  $s_1 \dot{=}_{[t, t')} s_2$ . We denote this as  $s_1 \dot{=}_{t+} s_2$ .



**Fig. 2.** A piecewise-constant left-continuous trajectory which models an execution of a (discrete-time) state transition system  $(Q, r)$ . At integer time moments system changes its current state  $q$  to a next state  $q'$  such that  $(q, q') \in r$  (if binary relation  $r$  is not functional, such a state is chosen non-deterministically).

Let  $Q$  be a set of states. Denote by  $ST(Q)$  the set of pairs  $(s, t)$  where  $s : A \rightarrow Q$  for some  $A \in \mathfrak{T}$  and  $t \in A$ .

**Definition 5.** A predicate  $p : ST(Q) \rightarrow Bool$  ( $Bool = \{true, false\}$ ) is called

- left-local, if  $p(s_1, t) \Leftrightarrow p(s_2, t)$  whenever  $(s_1, t), (s_2, t) \in ST(Q)$  and  $s_1 \dot{=}_{t-} s_2$ , and moreover,  $p(s, t)$  whenever  $t$  is the least element of  $dom(s)$ .
- right-local, if  $p(s_1, t) \Leftrightarrow p(s_2, t)$  whenever  $(s_1, t), (s_2, t) \in ST(Q)$ ,  $s_1 \dot{=}_{t+} s_2$ , and moreover,  $p(s, t)$  whenever  $t$  is the greatest element of  $dom(s)$ .

Let us denote by  $LR(Q)$  the set of all pairs  $(l, r)$  of a left-local predicate  $l : ST(Q) \rightarrow Bool$  and right-local predicate  $r : ST(Q) \rightarrow Bool$ .

Let us introduce an implication relation  $\Rightarrow$  on  $LR(Q)$  in the following way:  $(l, r) \Rightarrow (l', r')$  iff  $l(s, t) \Rightarrow l'(s, t)$  and  $r(s, t) \Rightarrow r'(s, t)$  for all  $s, t$ . It is easy to see that  $\Rightarrow$  is a partial order and that  $(LR(Q), \Rightarrow)$  is a complete meet semi-lattice (in the sense that every subset has an infimum), because the sets of all left-local predicates on  $ST(Q)$  and all right-local predicates on  $ST(Q)$  are closed under pointwise conjunction.

Let  $\mathbf{Tr}(Q)$  be the class of all CPR complete Markovian sets of trajectories which take values in the set of states  $Q$ . It is not difficult to check that  $\mathbf{Tr}(Q)$  is closed under intersections (of arbitrary cardinality). So we will consider it as a complete meet semi-lattice (with relation  $\subseteq$  as a partial order). Let us define a set-valued function  $\Phi$  on  $LR(Q)$  as follows:

$$\Phi((l, r)) = \{s : A \rightarrow Q \mid A \in \mathfrak{T} \wedge (\forall t \in A \ l(s, t) \wedge r(s, t))\}.$$

In the rest of the chapter we will write  $\Phi(l, r)$  instead of  $\Phi((l, r))$ .

The following theorem shows how left- and right-local predicates can represent complete Markovian sets of trajectories (and systems).



**Theorem 1.**  $\Phi$  is a surjective homomorphism from the complete meet semi-lattice  $LR(Q)$  onto  $\mathbf{Tr}(Q)$  (here homomorphism means a mapping that preserves arbitrary meets).

*Proof.* It is straightforward to check that  $\Phi$  indeed takes values in  $\mathbf{Tr}(Q)$  and that  $\Phi$  is a homomorphism. Let us prove that  $\Phi$  is surjective.

Let us fix an arbitrary  $Tr \in \mathbf{Tr}(Q)$  and define predicates  $l : ST(Q) \rightarrow Bool$  and  $r : ST(Q) \rightarrow Bool$  as follows:

- $l(s, t)$  iff  $t$  is the least element of  $dom(s)$ , or there exists  $t' < t$  such that  $[t', t] \subseteq dom(s)$  and  $s|_{[t', t]} \in Tr$ .
- $r(s, t)$  iff  $t$  is the greatest element of  $dom(s)$ , or there exists  $t' > t$  such that  $[t, t'] \subseteq dom(s)$  and  $s|_{[t, t']} \in Tr$ .

It follows immediately from CPR property of  $Tr$  that  $l$  is left-local,  $r$  is right-local, and  $Tr \subseteq \Phi(l, r)$ . Let us prove the opposite inclusion  $\Phi(l, r) \subseteq Tr$ .

Assume that  $A \in \mathfrak{T}$ ,  $s : A \rightarrow Q$ , and  $\forall t \in A \ l(s, t) \wedge r(s, t)$ .

Consider the following cases:

- $A = [a, b]$  for some  $a < b$ . For each  $t \in (a, b)$  there exists  $t' < t$  and  $t'' > t$  such that  $[t', t''] \subseteq dom(s)$  and  $s|_{[t', t'']} \in Tr$ . Denote  $O_t = (t', t'')$ . For  $a$  there exists  $t'' > a$  such that  $[a, t''] \subseteq dom(s)$  and  $s|_{[a, t'']} \in Tr$ . Denote  $O_a = [a, t'']$ . Similarly, for  $b$  there exists  $t' < b$  such that  $[t', b] \subseteq dom(s)$  and  $s|_{[t', b]} \in Tr$ . Denote  $O_b = (t', b]$ . Thus we defined  $O_t$  for all  $t \in A$ . Then  $(O_t)_{t \in A}$  is an open cover of a compact set  $A$  (in sense of induced topology). Then it has a finite subcover  $O_{t_i} = (t'_i, t''_i)$ ,  $i = 1, 2, \dots, k$ . Without loss of generality we can assume that  $a = t_1 \leq t_2 \leq \dots \leq t_k = b$ . By construction of  $O_t$ ,  $s|_{[t'_i, t''_i]} \in Tr$  and  $t'_i \leq t_i \leq t''_i$  for  $i = 1, 2, \dots, k$ . Then it is easy to see that CPR and Markovian properties of  $Tr$  imply that  $s|_{[a, b]} = s \in Tr$ .
- $A = [a, b)$  for some  $a < b$  ( $a \in T$ ,  $b \in T \cup \{+\infty\}$ ). From the previous case, left locality of  $l$ , and right locality of  $r$  we obtain  $s|_{[a, t]} \in Tr$  for all  $t \in (a, b)$ . From completeness property of  $Tr$  we conclude that  $s \in Tr$ .
- $A = (a, b]$  for some  $a < b$ . The proof is analogous to the previous case.
- $A = (a, b)$  for some  $a < b$  ( $a \in T$ ,  $b \in T \cup \{+\infty\}$ ). Let us choose an arbitrary  $c \in (a, b)$ . From the two previous cases, left locality of  $l$ , and right locality of  $r$  we obtain that  $s|_{(a, c]} \in Tr$  and  $s|_{[c, b)} \in Tr$ . Then  $s \in Tr$  by Markovian property of  $Tr$ .

We conclude that  $(l, r) \in LR(Q)$  and  $\Phi(l, r) = Tr$ . Thus  $\Phi$  is surjective.

Consider an example. Let  $Q = \mathbb{R}^d$  and  $Tr$  be the set of all functions  $s : A \rightarrow Q$ ,  $A \in \mathfrak{T}$  such that  $s$  is differentiable on  $A$  and satisfies a differential equation  $\frac{dy}{dt} = f(t, y)$  on  $A$ , where  $f : \mathbb{R} \times \mathbb{R}^d \rightarrow \mathbb{R}^d$  is a given function. Then  $Tr$  is complete and Markovian by Proposition 2. One possible representation of  $Tr$  using left-/right-local predicates can be constructed as follows.

Let  $l, r : ST(Q) \rightarrow Bool$  be predicates such that

- $l(s, t)$  iff either  $t$  is the least element of  $dom(s)$ , or  $\partial_- s(t) \downarrow = f(t, s(t))$ ,

–  $r(s, t)$  iff either  $t$  is the greatest element of  $dom(s)$ , or  $\partial_+s(t) \downarrow = f(t, s(t))$ ,

where  $\partial_-s(t)$  and  $\partial_+s(t)$  denote the left- and right- derivative of  $s$  at  $t$  (the symbol here  $\downarrow$  indicates that the left hand side of the equality is defined). Then it is not difficult to check that  $l$  is left-local,  $r$  is right-local, and  $Tr = \{s : A \rightarrow Q \mid A \in \mathfrak{T} \wedge (\forall t \in A \ l(s, t) \wedge r(s, t))\}$ .

### 3 Existence of Global-in-Time Trajectories

Let us recall our original question about global-in-time trajectories and consider it for non-deterministic complete Markovian systems.

Let  $\Sigma = (M, T, Q, Tr)$  be a NCMS.

**Definition 6.** *We say that  $\Sigma$  satisfies*

- (1) *global trajectory existence (GTE) property, if for each  $t_0 \in T$ ,  $q_0 \in Q$  there exists a (global-in-time) trajectory  $s : T \rightarrow Q$  of  $\Sigma$  such that  $s(t_0) = q_0$ .*
- (2) *local trajectory existence (LTE) property, if for each  $t_0 \in T$ ,  $q_0 \in Q$  there exists  $s \in Tr$  such that  $dom(s)$  is a neighborhood of  $t_0$  and  $s(t_0) = q_0$ .*

Our original question corresponds to GTE property. Our goal is to show that GTE can be checked using independent analysis of existence of locally defined trajectories of  $\Sigma$  in a neighborhood of each time moment.

Assume that information about trajectories of  $\Sigma$  defined in a neighborhood of each time moment  $t$  is represented as some function  $F : T \rightarrow 2^{Tr}$  such that for each  $t \in T$  there exists a neighborhood  $O(t)$  of  $t$  such that  $F(t)$  includes all trajectories which are defined on a subset of  $O(t)$  and at time  $t$ , i.e.,

$$F(t) \supseteq \{s \in Tr \mid t \in dom(s) \subseteq O(t)\} \tag{1}$$

(obviously, such a function exists and in general case it is not unique). We would like to analyze  $F(t)$  for each  $t$  independently and decide whether GTE holds. More specifically, we would like to express GTE in the form  $\forall t.P(t, F(t))$  or  $\neg \forall t.P(t, F(t))$ , where  $P$  is some predicate on  $T \times 2^{T \rightarrow Q}$  independent of  $F$  and  $Tr$ . Note that it is straightforward to express LTE in this form.

**Definition 7.** *We say that  $\Sigma$  satisfies*

- (1) *strong global extensibility (SGE) property, if for each trajectory  $s$  there exists a (global-in-time) trajectory  $s' : T \rightarrow Q$  such that  $s \sqsubseteq s'$ .*
- (2) *weak global extensibility (WGE) property, if for each trajectory  $s$  of the form  $s : [a, b] \rightarrow Q$  there exists a (global-in-time) trajectory  $s' : T \rightarrow Q$  such that  $s \sqsubseteq s'$ .*

Obviously, conjunction of LTE and SGE implies GTE. But as we have noted in the introduction, GTE may hold even if some locally defined trajectory cannot be extended to a globally defined trajectory.

**Lemma 1.**  *$\Sigma$  satisfies GTE iff  $\Sigma$  satisfies LTE and WGE.*

*Proof. "If":* Assume that  $\Sigma$  satisfies LTE and WGE. Let  $t_0 \in T$ ,  $q_0 \in Q$ . By LTE there exists  $s \in Tr$  such that  $dom(s)$  is a neighborhood of  $t_0$  and  $s(t_0) = q_0$ . Let us choose a segment  $[a, b] \subseteq dom(s)$  such that  $a < b$  and  $t_0 \in [a, b]$ . By CPR  $s|_{[a,b]} \in Tr$ . By WGE there exists a trajectory  $s' : T \rightarrow Q$  such that  $s|_{[a,b]} \sqsubseteq s'$ . Then  $s'(t_0) = s(t_0) = q_0$ . Thus  $\Sigma$  satisfies GTE.

**"Only if":** Assume that  $\Sigma$  satisfies GTE. From definitions we have that  $\Sigma$  satisfies LTE. Let us show that  $\Sigma$  satisfies WGE. Let  $s \in Tr$  and  $dom(s) = [a, b]$  ( $a < b$ ). By GTE there exist trajectories  $s_a : T \rightarrow Q$  and  $s_b : T \rightarrow Q$  such that  $s_a(a) = s(a)$  and  $s_b(b) = s(b)$ . By CPR,  $s_a|_{[0,a]} \in Tr$  and  $s_b|_{[b,+\infty)} \in Tr$ . Then  $s \in Tr$  by Markovian property.

Because of this lemma we will focus on WGE property.

Let us introduce several definitions.

**Definition 8.**  $\Sigma$  satisfies a weak local extensibility (WLE) property, if for each  $s \in Tr$  of the form  $s : [a, b] \rightarrow Q$  there exists  $s' : [a, b'] \rightarrow Q \in Tr$  such that  $s \sqsubseteq s'$  and  $b' > b$ , and if  $a > 0$ , there exists  $s'' : [a', b] \rightarrow Q \in Tr$  such that  $s \sqsubseteq s''$  and  $a' < a$ .

**Definition 9.** (1) A right dead-end path (in  $\Sigma$ ) is a trajectory  $s : A \rightarrow Q$  such that  $A$  has a form  $[a, b)$ , where  $a, b \in T$ , and there is no  $s' : [a, b] \rightarrow Q \in Tr$  such that  $s = s'|_{dom(s)}$  (i.e.,  $s$  cannot be extended to a trajectory on  $[a, b]$ ).

(2) A left dead-end path (in  $\Sigma$ ) is a trajectory  $s : A \rightarrow Q$  such that  $A$  has a form  $(a, b]$ , where  $a, b \in T$ , and there is no  $s' : [a, b] \rightarrow Q \in Tr$  such that  $s = s'|_{dom(s)}$ .

(3) A dead-end path is either a right dead-end path, or a left dead-end path.

**Definition 10.** (1) An escape from a right dead-end path  $s : [a, b) \rightarrow Q$  (in  $\Sigma$ ) is a trajectory  $s' : [c, d) \rightarrow Q$  ( $d \in T \cup \{+\infty\}$ ) or  $s' : [c, d] \rightarrow Q$  such that  $c \in (a, b)$ ,  $d > b$ , and  $s(c) = s'(c)$ . An escape  $s'$  is called infinite, if  $d = +\infty$ .

(2) An escape from a left dead-end path  $s : (a, b] \rightarrow Q$  is

– a trajectory  $s' : (c, d] \rightarrow Q$  or  $s' : [c, d] \rightarrow Q$  such that  $c < a$ ,  $d \in (a, b)$ , and  $s'(d) = s(d)$ , if  $a > 0$ .

– a trajectory  $s' : [0, d] \rightarrow Q$  such that  $d \in (a, b)$  and  $s'(d) = s(d)$ , if  $a = 0$ .

An escape  $s'$  from  $s$  is called initial, if  $0 \in dom(s')$ .

**Definition 11.** A right (or left) dead-end path  $s : [a, b) \rightarrow Q$  in  $\Sigma$  is called strongly escapable, if there exists an infinite (resp. initial) escape from  $s$ .

**Lemma 2.** (1) If  $s : [a, b) \rightarrow Q$  is a right dead-end path and  $c \in (a, b)$ , then  $s|_{[c,b)}$  is a right dead-end path.

(2) If  $s : (a, b] \rightarrow Q$  is a left dead-end path and  $c \in (a, b)$ , then  $s|_{(a,c]}$  is a left dead-end path.

The proof follows immediately from CPR and Markovian properties of  $Tr$ .

**Lemma 3.**  $\Sigma$  satisfies WGE iff  $\Sigma$  satisfies WLE and each dead-end path is strongly escapable.

*Proof. "If":* Assume that  $\Sigma$  satisfies WLE and each dead-end path in  $\Sigma$  is strongly escapable. Let  $s : [a, b] \rightarrow Q$  be a trajectory. Let  $S = \{s' \in Tr \mid s \sqsubseteq s'\}$ . Then  $S \neq \emptyset$  and because of completeness property of  $Tr$ , each  $\sqsubseteq$ -chain of elements of  $S$  has a supremum which belongs to  $S$ . Then by Zorn's lemma,  $S$  has some maximal element  $s^*$  (with respect to  $\sqsubseteq$ ). Because of WLE, domain of  $s^*$  cannot be a (compact) segment. Consider the following cases:

- $dom(s^*) = [x, y]$  for some  $x, y \in T$ . Because  $s^*$  is maximal in  $S$ ,  $s^*$  cannot be extended to a trajectory on  $[x, y]$ . Hence  $s^*$  is a right dead-end path. Moreover,  $y > b$  and  $x \leq a$ . Then  $b \in (x, y)$  and by Lemma 2,  $s^*|_{[b, y]}$  is a right dead-end path. Then there exists some infinite escape  $s_1 : [c, +\infty) \rightarrow Q$  from  $s^*|_{[b, y]}$  (where  $c \in (b, y)$ ,  $s_e(c) = s^*(c)$ ). Let us define  $s_2 : [x, +\infty)$  as follows:

$$s_2(t) = \begin{cases} s_*(t), & t \in [x, c] \\ s_1(t), & t > c \end{cases}$$

Then  $s_2 \in Tr$  by CPR and Markovian properties. Moreover,  $s \sqsubseteq s_2$ , because  $c > b$  and  $s \sqsubseteq s_*$ . Also, WLE, CPR and Markovian properties imply that  $x = 0$  (otherwise one can extend  $s^*$  to the left and obtain a contradiction with maximality of  $s^*$  in  $S$ ). Thus  $s_2$  is defined globally.

- $dom(s^*) = [x, +\infty)$  for some  $x \in T$ . Using WLE, CPR and Markovian properties it is straightforward to show that  $x = 0$ .
- $dom(s^*) = (x, y]$  for some  $x, y \in T$ . Using WLE, CPR, and Markovian properties it is straightforward to show that this case is impossible.
- $dom(s^*) = (x, y)$  for some  $x, y \in T$ . Then  $x < a$  and  $y > b$ . Because  $s^*$  is maximal in  $S$ ,  $s^*$  cannot be extended on  $[x, y)$  or  $(x, y]$ . Let us choose an arbitrary  $z \in (a, b) \subset (x, y)$ . Then CPR and Markovian properties imply that  $s^*|_{[z, y]}$  is a right dead-end path and  $s^*|_{(x, z]}$  is a left dead-end path. By Lemma 2,  $s^*|_{[b, y]}$  is a right dead-end path and  $s^*|_{(x, a]}$  is a left dead-end path. Then there exists an infinite escape  $s_r : [c_r, +\infty) \rightarrow Q$  from  $s^*|_{[b, y]}$  (where  $c_r \in (b, y)$ ) and an initial escape  $s_l : [0, c_l] \rightarrow Q$  from  $s^*|_{(x, a]}$  (where  $c_l \in (x, a)$ ). Let us define  $s_3 : [0, +\infty) \rightarrow Q$  as follows:

$$s_3(t) = \begin{cases} s_l(t), & t \in [0, c_l] \\ s_*(t), & t \in (c_l, c_r) \\ s_r(t), & t > c_r \end{cases}$$

By Markovian property,  $s_3 \in Tr$ . Moreover,  $s \sqsubseteq s_3$ , because  $[a, b] \subset (x, y) \subset (c_l, c_r)$  and  $s \sqsubseteq s_*$ .

- $dom(s^*) = (x, +\infty)$  for some  $x \in T$ . The proof is analogous to the previous case (with distinction that only left dead-end path is used).

We conclude that there exists  $s' : T \rightarrow Q$  such that  $s \sqsubseteq s'$ . Thus WGE holds.

**"Only if":** Assume that  $\Sigma$  satisfies WGE. Then  $\Sigma$  satisfies WLE because of CPR property. Let us check that each dead-end path is strongly escapable.

- Let  $s : [a, b] \rightarrow Q$  ( $a < b$ ) be a right dead-end path. Let  $c \in (a, b)$ . Then  $s|_{[a, c]} \in Tr$  by CPR. Then there exists a trajectory  $s' : T \rightarrow Q$  such that  $s|_{[a, c]} \sqsubseteq s'$  by WGE. Let  $s'' = s'|_{[c, +\infty)}$ . Then  $s'' \in Tr$  by CPR and  $s''(c) = s(c)$ . Then  $s''$  is an infinite escape from  $s$ .

- Let  $s : (a, b] \rightarrow Q$  ( $a < b$ ) be a left dead-end path. Let  $c \in (a, b)$ . Then  $s|_{[c, b]} \in Tr$  by CPR and there exists a trajectory  $s' : T \rightarrow Q$  such that  $s|_{[c, b]} \sqsubseteq s'$  by WGE. Let  $s'' = s'|_{[0, c]}$ . Then  $s'' \in Tr$  by CPR, because  $c > 0$ . Then  $s''$  is an initial escape from  $s$ , because  $s''(c) = s(c)$ .

Thus each dead-end path is strongly escapable.

**Lemma 4.** *LTE implies WLE.*

*Proof.* Assume that  $\Sigma$  satisfies LTE.

Let  $s : [a, b] \rightarrow Q$  be a trajectory. By LTE there exists  $s_1 \in Tr$  such that  $dom(s_1)$  is a neighborhood of  $b$  and  $s_1(b) = s(b)$ . Let us choose  $b' > b$  such that  $[b, b'] \subseteq dom(s_1)$ . Then  $s_1|_{[b, b']} \in Tr$  by CPR. Let us define  $s' : [a, b'] \rightarrow Q$  as follows:  $s'(t) = s(t)$ , if  $t \in [a, b]$  and  $s'(t) = s_1(t)$ , if  $t \in (b, b']$ . Then  $s' \in Tr$  by Markovian property. Moreover,  $s \sqsubseteq s'$ .

Analogously, if  $a > 0$ , we can find  $s'' : [a', b] \rightarrow Q \in Tr$  such that  $s \sqsubseteq s''$ ,  $a' < a$ . Thus  $\Sigma$  satisfies WLE.

Because of the previous two lemma, now we will focus on the question of whether each dead-end path is strongly escapable.

- Definition 12.** (1) A right extensibility measure is a function  $f^+ : T \times T \rightarrow T$  which is defined and continuous on  $\{(x, y) \in T \times T \mid x \leq y\}$  such that
- $f^+(x, y)$  is strictly decreasing in  $x$  and strictly increasing in  $y$ ;
  - $f^+(x, x) = x$  and  $f^+(x, y) > y$  for all  $x, y$  such that  $x < y$ .
- (2) A left extensibility measure is a function  $f^- : T \times T \rightarrow T$  which is defined and continuous on  $\{(x, y) \in T \times T \mid x \geq y\}$  such that
- $f^-(x, y)$  is strictly increasing in  $x$  and strictly decreasing in  $y$ ;
  - $f^-(x, x) = x$  and  $f^-(x, y) < y$  for all  $x, y$  such that  $x > y > 0$ .

Let us fix a right extensibility measure  $f^+$  and a left extensibility measure  $f^-$ .

- Definition 13.** (1) A right dead-end path  $s : [a, b] \rightarrow Q$  is called  $f^+$ -escapable, if there exists an escape  $s' : [c, d] \rightarrow Q$  from  $s$  such that  $d \geq f^+(c, b)$ .
- (2) A left dead-end path  $s : (a, b] \rightarrow Q$  is called  $f^-$ -escapable, if there exists an escape  $s' : [d, c] \rightarrow Q$  from  $s$  such that  $d \leq f^-(c, a)$ .

**Theorem 2.** Assume that  $\Sigma$  satisfies WLE. Then each right dead-end path is strongly escapable iff each right dead-end path is  $f^+$ -escapable.

**Theorem 3.** Assume that  $\Sigma$  satisfies WLE. Then each left dead-end path is strongly escapable iff each left dead-end path is  $f^-$ -escapable.

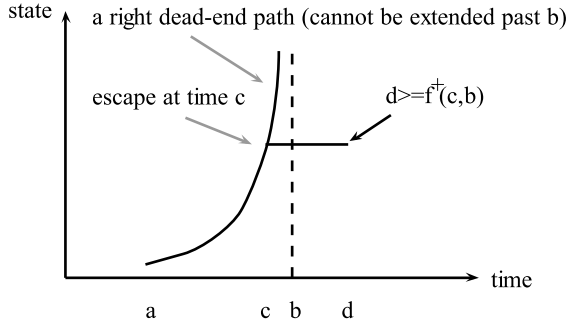
Note that these theorems hold for any fixed  $f^+$  and  $f^-$ . We will give a proof of Theorem 2 in the next section. The proof of Theorem 3 is analogous.

An example of a right extensibility measure is  $f^+(x, y) = 2y - x$  ( $x \leq y$ ). In this case for a right dead-end path to be  $f^+$ -escapable it is necessary that there exists an escape  $s' : [c, d] \rightarrow Q$  with  $d - b \geq b - c$ .

**Theorem 4.**  $\Sigma$  satisfies GTE iff  $\Sigma$  satisfies LTE and each right dead-end path is  $f^+$ -escapable and each left dead-end path is  $f^-$ -escapable.

The proof follows from lemmas 1, 3, 4 and theorems 2, 3.

Using this theorem it is not difficult to represent GTE in the form  $\forall t. P(t, F(t))$ , where  $F$  is defined as in (1).



**Fig. 3.** An  $f^+$ -escapable right dead-end path  $s : [a, b) \rightarrow Q$  (displayed here as a curve) and a corresponding escape  $s' : [c, d] \rightarrow Q$  (displayed here as a horizontal segment) such that  $d \geq f^+(c, b)$ .

### 4 Proof of Theorem about Right Dead-End Paths

In this section we give a proof of Theorem 2. Let us introduce several auxiliary definitions and lemmas.

**Definition 14.** A right  $t_0$ -bunch (in  $\Sigma$ ) is a non-empty set  $A \subseteq Tr$  such that  $\min(\text{dom}(s)) \downarrow = t_0$  for each  $s \in A$  and  $s_1 \dot{=}_{t_0} s_2$  for all  $s_1, s_2 \in A$ .

For each non-empty  $A \subseteq Tr$  denote

$$|A|^+ = \sup_{s \in A} (\sup \text{dom}(s)).$$

(we assume  $|A|^+ = +\infty$ , if  $\sup(\text{dom}(s)) = +\infty$  for some  $s \in A$ ).

**Definition 15.** A (right)  $t_0$ -bunch  $A$  is called bounded, if  $|A|^+ < +\infty$ . Otherwise it is called unbounded.

**Lemma 5.** For a right extensibility measure  $f^+$  there exists a function  $g^+ : T \times T \rightarrow T$  which is defined on  $\{(x, y) \in T \times T \mid x \leq y\}$  such that

- (1)  $g^+$  is strictly increasing in both arguments;
- (2)  $g^+(x, x) = x$  and  $x < g^+(x, y) < y$  for all  $x, y$  such that  $x < y$ ;
- (3)  $g^+(x, f^+(x, y)) = y$  for all  $x, y$  such that  $x \leq y$ .

*Proof.* For each fixed  $x \geq 0$ , function  $f^+(x, \cdot)$  maps the set  $[x, +\infty)$  to itself. Because  $f^+(x, \cdot)$  is strictly increasing, it has a strictly increasing inverse. Denote this inverse as  $g^+(x, \cdot)$ . Then  $g^+(x, y)$  is defined for all  $x \leq y$ . It is straightforward to check that it satisfies (1)-(3).

Let us fix a right extensibility measure  $f^+$  and let  $g^+$  be a corresponding function described in Lemma 5. We call  $g^+$  a right bunch convergence measure.

**Definition 16.** A bounded right  $t_0$ -bunch  $A$  is called  $g^+$ -convergent, if for each  $t' \in (t_0, |A|^+)$  and  $s_1, s_2 \in A$ :

$$\text{if } \min\{\sup(\text{dom}(s_1)), \sup(\text{dom}(s_2))\} \geq g^+(t', |A|^+), \text{ then } s_1 \dot{=}_{[t_0, t']} s_2.$$

**Definition 17.** A function  $\alpha : [0, +\infty) \rightarrow [0, +\infty)$  is of class  $K_\infty$ , if it is continuous, strictly increasing, unbounded ( $\lim_{x \rightarrow +\infty} \alpha(x) = +\infty$ ), and  $\alpha(0) = 0$ .

**Lemma 6.** There exists a function  $\alpha \in K_\infty$  such that  $\alpha(x) < x$  for all  $x > 0$  and the function  $x \mapsto f^+(\alpha(x), x)$  is of class  $K_\infty$ .

The proof follows from continuity and monotonicity of  $f^+$  and that  $f^+(x, x) = x$ .

To continue we need the notion of iterative root [12]. A function  $f$  which satisfies the functional equation  $\underbrace{f(f(\dots f(x)\dots))}_{N \text{ times}} = g(x)$  ( $N \in \mathbb{N}$ ) is called an  $N$ -th order iterative root of the function  $g$ . The existence of iterative roots can be established in some cases using the following lemma.

**Lemma 7 ([12], Theorem 11.2.2).** Let  $X \subseteq \mathbb{R}$  be an interval and  $f$  be a strictly increasing and continuous self-mapping of  $X$ . Then  $f$  possess strictly increasing and continuous iterative roots of all orders.

Let us choose  $\alpha$  as in Lemma 6. Then by Lemma 7 there exists a continuous and strictly increasing function  $\xi$  on  $[0, +\infty)$  such that for all  $x \geq 0$ ,

$$\xi(\xi(x)) = f^+(\alpha(x), x) \tag{2}$$

**Lemma 8.**  $\xi$  is of class  $K_\infty$  and  $\xi(x) > x$  for all  $x > 0$ .

The proof follows immediately from definitions.

Let  $\phi$  be a strictly increasing and continuous function such that

$$\phi(\phi(x)) = \xi(x)$$

for all  $x > 0$  (it exists by Lemma 7). Then  $\phi(x) > x$  for all  $x > 0$  (because otherwise there exists  $x_0 > 0$  such that  $\phi(x_0) \leq x_0$ , whence  $\xi(x_0) = \phi(\phi(x_0)) \leq \phi(x_0) \leq x_0$  – a contradiction). Let  $\psi$  be a strictly increasing and continuous function such that

$$\psi(\psi(x)) = \phi(x)$$

for all  $x > 0$  (it exists by Theorem 7). Then  $\psi(x) > x$  for all  $x > 0$ , because  $\phi(x) > x$  for all  $x > 0$ . Then for all  $x > 0$ ,

$$x < \psi(x) < \psi(\psi(x)) = \phi(x) < \phi(\phi(x)) = \xi(x). \tag{3}$$

For each set of sets  $S$  and a binary relation  $<\subseteq S \times S$  let us denote:

1.  $Ch(S, <)$  is the set of all subsets  $c \subseteq S$  such that
  - $c$  is a Dedekind-complete chain with respect to the subset relation  $\subseteq$ , i.e.,  $A \subseteq B$  or  $B \subseteq A$  for each  $A, B \in c$ , and for each non-empty subset  $c' \subseteq c$ , if there exists  $X \in c$  such that  $\bigcup c' \subseteq X$ , then  $\bigcup c' \in c$ ;

- for each non-maximal  $A \in c$  (i.e.,  $A \subset A'$  for some  $A' \in c$ ) there exists  $A' \in c$  such that  $A \subseteq A'$  and  $A \prec A'$ .
- 2.  $\leq$  is a binary relation on  $Ch(S, \prec)$  such that  $c_1 \leq c_2$  iff  $c_1 \subseteq c_2$ , and  $A \subseteq B$  for all  $A \in c_1$  and  $B \in c_2 \setminus c_1$ .

For each  $t_0 \geq 0$  let us define:

1.  $S_{t_0}^+$  is the set of all bounded  $g^+$ -convergent right  $t_0$ -bunches (in  $\Sigma$ ).
2.  $\prec^+$  is a binary relation on  $S_{t_0}^+$  such that  $A \prec^+ B$  iff  $|A|^+ < |B|^+ < \psi(|A|^+)$ .

Let us consider some general properties of  $Ch(S, \prec)$ .

**Lemma 9.** (1)  $\leq$  is a partial order on  $Ch(S, \prec)$ .  
 (2) Each chain in the poset  $(Ch(S, \prec), \leq)$  has an upper bound.

*Proof.* (1) follows immediately from definition of  $\leq$ .

Let us show (2). Let  $C \subseteq Ch(S, \prec)$  be a  $\leq$ -chain. Let us show that  $c = \bigcup C \in Ch(S, \prec)$ . It is straightforward to check that  $c$  is a  $\subseteq$ -chain.

Let us check that for each non-empty  $c' \subseteq c$ , if there exists  $X \in c$  such that  $\bigcup c' \subseteq X$ , then  $\bigcup c' \in c$ . Assume  $c' \subseteq c$ ,  $c' \neq \emptyset$ ,  $X \in c$  and  $\bigcup c' \subseteq X$ . Then there exists  $c_X \in C$  such that  $X \in c_X$ , because  $X \in c = \bigcup C$ . Firstly, let us show that  $c' \subseteq c_X$ . Let  $A \in c'$ . Because  $A \in c' \subseteq c = \bigcup C$ , there exists  $c_A \in C$  such that  $A \in c_A$ . Now assume that  $A \notin c_X$ . Then  $c_A \not\leq c_X$  (otherwise  $A \in c_A \subseteq c_X$ ). Then  $c_X \leq c_A$ , because  $C$  is a  $\leq$ -chain and  $c_A, c_X \in C$ . Then  $X \subseteq A$ , because  $X \in c_X$ ,  $A \in c_A \setminus c_X$ . Then  $A \subseteq \bigcup c' \subseteq X \subseteq A$ . Then we have a contradiction  $A = X \in c_X$ . Thus  $A \in c_X$  and we conclude that  $c' \subseteq c_X$ . Now we have that  $c' \subseteq c_X$ ,  $c' \neq \emptyset$  and there exists  $X \in c_X$  such that  $\bigcup c' \subseteq X$ . Then  $\bigcup c' \in c_X$  by the definition of  $Ch(S, \prec)$ , because  $c_X \in C \subseteq Ch(S, \prec)$ . Thus  $\bigcup c' \in \bigcup C = c$ .

Let us check that for each non-maximal  $A \in c$  (with respect to  $\subseteq$ ) there exists  $A' \in c$  such that  $A \subseteq A'$  and  $A \prec A'$ . Assume that  $A \in c$  is non-maximal. Then there exists  $B \in c$  such that  $A \subset B$ . Then there exist  $c_A, c_B \in C$  such that  $A \in c_A$ ,  $B \in c_B$ , because  $c = \bigcup C$ . Moreover, either  $c_A \leq c_B$  or  $c_B \leq c_A$ . If  $A \in c_B$ , then  $A$  is a non-maximal element of  $c_B$ . Then there exists  $A' \in c_B$  such that  $A \subseteq A'$  and  $A \prec A'$ , because  $c_B \in Ch(S, \prec)$ . Then  $A' \in \bigcup C = c$ . On the other hand, if  $A \notin c_B$ , then  $c_B \leq c_A$  (because otherwise  $A \in c_B$ ) and  $B \subseteq A$ , because  $B \in c_B$  and  $A \in c_A \setminus c_B$ . This contradicts the inclusion  $A \subset B$  given above. We conclude that there exists  $A' \in c$  such that  $A \subseteq A'$  and  $A \prec A'$ .

Thus  $c \in Ch(S, \prec)$  by the definition of  $Ch(S, \prec)$ .

**Lemma 10.** Let  $c_m$  be a  $\leq$ -maximal element of  $Ch(S, \prec)$  and  $X = \bigcup c_m \in S$ .

- (1)  $X \in c_m$ .
- (2) There is no set  $Y \in S$  such that  $X \subset Y$  and  $X \prec Y$ .

*Proof.* Let us prove (1). Let  $c'_m = c_m \cup \{X\}$ . Let us show that  $c'_m \in Ch(S, \prec)$ .

We have  $c'_m \subseteq S$  and  $A \subseteq X$  for all  $A \in c_m$ , because  $X = \bigcup c_m$ . Moreover,  $c_m$  is a  $\subseteq$ -chain. Thus  $c'_m = c_m \cup \{X\}$  is a  $\subseteq$ -chain.

Let us check that for each non-empty  $c' \subseteq c'_m$ , if there exists  $X' \in c'_m$  such that  $\bigcup c' \subseteq X'$ , then  $\bigcup c' \in c'_m$ . Assume that  $c' \subseteq c'_m$ ,  $c' \neq \emptyset$ . If  $X \in c'$ , then



$X \subseteq \bigcup c' \subseteq \bigcup c'_m = (\bigcup c_m) \cup X = X$  and  $\bigcup c' \in c'_m$ . Consider the case when  $X \notin c'$ . Then  $c' \subseteq c_m$  and because  $c_m$  is a  $\subseteq$ -chain, for each  $A \in c_m$ , either  $A \subseteq B$  holds for some  $B \in c'$ , or  $B \subseteq A$  holds for all  $B \in c'$ . Then for each  $A \in c_m$ , either  $A \subseteq \bigcup c'$  or  $\bigcup c' \subseteq A$ . If  $\bigcup c' \subseteq A$  for some  $A \in c_m$ , then by taking into account that  $c' \subseteq c_m$ ,  $c' \neq \emptyset$  and  $c_m \in Ch(S, \prec)$ , we have  $\bigcup c' \in c_m \subseteq c'_m$ . If  $A \subseteq \bigcup c'$  for all  $A \in c_m$ , then  $X = \bigcup c_m \subseteq \bigcup c' \subseteq \bigcup c_m = X$ , because  $c' \subseteq c_m$ . Then  $\bigcup c' \in c'_m$ . Thus in all cases  $\bigcup c' \in c'_m$ .

Let us check that for each non-maximal  $A \in c'_m$  (with respect to  $\subseteq$ ) there exists  $A' \in c'_m$  such that  $A \subseteq A'$  and  $A \prec A'$ . Assume that  $A \in c'_m$  is non-maximal. Then  $A \neq X$ . Then  $A \in c_m$ . Moreover,  $A$  is a non-maximal element of  $c_m$ , because otherwise  $X = \bigcup c_m = A$ . Then because  $c_m \in Ch(S, \prec)$ , there exists  $A' \in c_m \subseteq c'_m$  such that  $A \subseteq A'$  and  $A \prec A'$ .

Thus  $c'_m \in Ch(S, \prec)$  by definition of  $Ch(S, \prec)$ .

We have  $c_m \subseteq c'_m$  and  $A \subseteq B$  for all  $A \in c_m$  and  $B \in c'_m \setminus c_m$ , because  $c'_m \setminus c_m \subseteq \{X\}$  and  $A \subseteq \bigcup c_m = X$ . Then  $c_m \leq c'_m$ . Then  $c_m = c'_m$ , because  $c'_m \in Ch(S, \prec)$  and  $c_m$  is a  $\leq$ -maximal element of  $Ch(S, \prec)$ . Thus  $X \in c_m$ .

Now let us prove (2) by contradiction. Assume that there exists  $Y \in S$  such that  $X \subset Y$  and  $X \prec Y$ . Let  $c'_m = c_m \cup \{Y\}$ . Let us show that  $c'_m \in Ch(S, \prec)$ .

We have  $c'_m \subseteq S$ . Also,  $A \subseteq Y$  for all  $A \in c_m$ , because  $\bigcup c_m = X \subseteq Y$ . Moreover,  $c_m$  is a  $\subseteq$ -chain. Thus  $c'_m = c_m \cup \{Y\}$  is a  $\subseteq$ -chain.

Let us check that for each non-empty  $c' \subseteq c'_m$ , if there exists  $X' \in c'_m$  such that  $\bigcup c' \subseteq X'$ , then  $\bigcup c' \in c'_m$ . Let  $c' \subseteq c'_m$ ,  $c' \neq \emptyset$ . If  $Y \in c'$ , then  $Y \subseteq \bigcup c' \subseteq \bigcup c'_m = (\bigcup c_m) \cup Y = X \cup Y = Y$  and  $\bigcup c' \in c'_m$ . Consider the case when  $Y \notin c'$ . Then  $c' \subseteq c_m$  and  $\bigcup c' \subseteq \bigcup c_m = X$ . Moreover,  $X \in c_m$  by the statement (1) of this lemma. From this and from  $c' \subseteq c_m$ ,  $c' \neq \emptyset$  and  $c_m \in Ch(S, \prec)$ , we have  $\bigcup c' \in c_m \subseteq c'_m$ . Thus in both cases we have  $\bigcup c' \in c'_m$ .

Let us check that for each non-maximal  $A \in c'_m$  (with respect to  $\subseteq$ ) there exists  $A' \in c'_m$  such that  $A \subseteq A'$  and  $A \prec A'$ . Assume  $A \in c'_m$  is non-maximal element. Then  $A \neq Y$ , because  $Y$  is a maximal element of  $c'_m$ . Then  $A \in c_m$ . If  $A$  is a non-maximal element of  $c_m$ , then there exists  $A' \in c_m \subseteq c'_m$  such that  $A \subseteq A'$  and  $A \prec A'$ , because  $c_m \in Ch(S, \prec)$ . If  $A$  is a maximal in  $c_m$ , then  $X = \bigcup c_m = A$ ,  $Y \in c'_m$ ,  $A \subseteq Y$  and  $A \prec Y$ . Thus  $c'_m \in Ch(S, \prec)$ .

We have  $c_m \subseteq c'_m$  and  $A \subseteq B$  for all  $A \in c_m$  and  $B \in c'_m \setminus c_m$ , because  $c'_m \setminus c_m \subseteq \{Y\}$  and  $A \subseteq \bigcup c_m = X \subseteq Y$ . Then  $c_m \leq c'_m$ . Also, we have  $X = \bigcup c_m \neq Y = \bigcup c'_m$ , because  $X \subset Y$ . Then  $c_m \neq c'_m$ . Then  $c_m$  is not a  $\leq$ -maximal element of  $Ch(S, \prec)$ , because  $c_m \leq c'_m$  and  $c'_m \in Ch(S, \prec) \setminus \{c_m\}$ . We have a contradiction with assumptions of the lemma. Thus there is no set  $Y \in S$  such that  $X \subset Y$  and  $X \prec Y$ .

Let us consider some properties of the set  $Ch(S_{t_0}^+, \prec^+) \setminus \{\emptyset\}$  for a fixed  $t_0 \in T$ . Note for each element  $c$  of this set,  $\bigcup c \neq \emptyset$ , because  $\emptyset \notin S_{t_0}^+$ .

**Lemma 11.** *If  $c \in Ch(S_{t_0}^+, \prec^+) \setminus \{\emptyset\}$  and  $|\bigcup c|^+ < +\infty$ , then  $\bigcup c \in S_{t_0}^+$ .*

*Proof.* Let  $c \in Ch(S_{t_0}^+, \prec^+) \setminus \{\emptyset\}$ ,  $X = \bigcup c$ , and  $|X|^+ < +\infty$ . Then  $X \neq \emptyset$ .

Let us show that  $X$  is a bounded right  $t_0$ -bunch. For each  $s \in X$  there exists  $A \in c$  such that  $s \in A$ . Then  $\min(dom(s)) \downarrow = t_0$ , because  $A$  is a right  $t_0$ -bunch.

Let  $s_1, s_2 \in X$ . Then there exist  $A_1, A_2 \in c$  such that  $s_1 \in A_1$  and  $s_2 \in A_2$ . Then  $A_1 \subseteq A_2$  or  $A_2 \subseteq A_1$ , because  $c$  is a  $\subseteq$ -chain. Moreover,  $A_1, A_2 \in S_{t_0}^+$ . If  $A_1 \subseteq A_2$ , then  $s_1, s_2 \in A_2$  for  $i = 1, 2$ . Then  $s_1 \dot{=}_{t_0} s_2$ , because  $A_2$  is a right  $t_0$ -bunch. Similarly, if  $A_2 \subseteq A_1$ , then  $s_1 \dot{=}_{t_0} s_2$ , because  $A_1$  is a right  $t_0$ -bunch. In both cases  $s_1 \dot{=}_{t_0} s_2$ . Thus  $X$  is a bounded right  $t_0$ -bunch, because  $|X|^+ < +\infty$ .

Let us show that  $X$  is  $g^+$ -convergent. Let  $t' \in (t_0, |X|^+)$ ,  $s_1, s_2 \in X$ . Then there exist  $A_1, A_2 \in c$  such that  $s_1 \in A_1$ ,  $s_2 \in A_2$ . Then  $A_1 \subseteq A_2$  or  $A_2 \subseteq A_1$ , because  $c$  is a chain. Also,  $A_1, A_2$  are bounded  $t_0$ -bunches, because  $A_1, A_2 \in S_{t_0}^+$ .

Let  $t_i = \sup(\text{dom}(s_i))$ ,  $i = 1, 2$ . Assume that  $\min\{t_1, t_2\} \geq g^+(t', |X|^+)$ .

Let us show that  $s_1 \dot{=}_{[t_0, t']} s_2$ .

Consider the case  $A_1 \subseteq A_2$ . Then  $s_1, s_2 \in A_2$  and  $t_1, t_2 \leq |A_2|^+$ . Then

$$|X|^+ \geq |A_2|^+ \geq \min\{t_1, t_2\} \geq g^+(t', |X|^+) > t',$$

because  $A_2 \subseteq X$  and  $t' < |X|^+$ . If  $t' \in (t_0, |A_2|^+)$ , then  $s_1 \dot{=}_{[t_0, t']} s_2$ , because  $A_2$  is  $g^+$ -convergent. Otherwise,  $t' = |A_2|^+$ . Then

$$\min\{t_1, t_2\} \geq g^+(t', |X|^+) \geq g^+(t', |A_2|^+) = |A_2|^+,$$

by monotonicity of  $g^+$ , whence  $t_1 = t_2 = |A_2|^+$ , because  $t_1, t_2 \leq |A_2|^+$ .

For each  $t'' \in (t_0, |A_2|^+)$  we have

$$\min\{t_1, t_2\} = |A_2|^+ > g^+(t'', |A_2|^+).$$

Then  $s_1 \dot{=}_{[t_0, t'']} s_2$ , because  $A_2$  is  $g^+$ -convergent. Then  $s_1 \dot{=}_{[t_0, t']} s_2$ , because  $t'' \in (t_0, |A_2|^+) = (t_0, t')$  is arbitrary.

In the case  $A_2 \subseteq A_1$  we can show that  $s_1 \dot{=}_{[t_0, t']} s_2$  using analogous arguments. Thus  $X$  is  $g^+$ -convergent. Then  $X \in S_{t_0}^+$  by the definition of  $S_{t_0}^+$ .

Let us define a prefix relation  $\trianglelefteq$  on  $Tr$ :  $s_1 \trianglelefteq s_2$  iff  $s_1 \sqsubseteq s_2$  and there exists  $E \subseteq T$  such that  $t_1 < t_2$  for all  $t_1 \in \text{dom}(s_1)$ ,  $t_2 \in E$ , and  $\text{dom}(s_2) = \text{dom}(s_1) \cup E$ . It is easy check see that  $\trianglelefteq$  is a partial order on  $Tr$ .

Let us define a prefix closure operation  $pcl$  on  $2^{Tr}$

$$pcl(A) = \{s \in Tr \mid \exists s' \in A \ s \trianglelefteq s'\}.$$

Then  $pcl$  is a closure operator (extensive, monotone, and idempotent).

**Lemma 12.**  $pcl(A) \in S_{t_0}^+$  for each  $A \in S_{t_0}^+$ .

The proof follows immediately from definitions.

**Lemma 13.** If  $c \in Ch(S_{t_0}^+, \prec^+)$ , then  $\{pcl(A) \mid A \in c\} \in Ch(S_{t_0}^+, \prec^+)$ .

*Proof.* Let  $c \in Ch(S_{t_0}^+, \prec^+)$  and  $\hat{c} = \{pcl(A) \mid A \in c\}$ . Each  $A \in \hat{c}$  belongs to  $S_{t_0}^+$  by Lemma 12. Also,  $\hat{c}$  is a  $\subseteq$ -chain, because  $c$  is a  $\subseteq$ -chain and  $pcl$  is monotone.

Let us check that  $\hat{c}$  is Dedekind-complete. Assume that for a non-empty  $c' \subseteq \hat{c}$  there exists  $X \in \hat{c}$  such that  $\bigcup c' \subseteq X$ . Then there exists non-empty  $c'' \subseteq c$  and

$Y \in c$  such that  $\bigcup c' = \bigcup \{pcl(A) \mid A \in c''\} \subseteq pcl(Y) = X$ . Then  $\bigcup \{pcl(A) \mid A \in c''\} = pcl(\bigcup c'') \subseteq pcl(Y)$ . If there exists  $Z \in c$  such that  $\bigcup c'' \subseteq Z$ , then  $\bigcup c'' \in c$ , because  $c$  is Dedekind-complete, and  $\bigcup c' = pcl(\bigcup c'') \in \hat{c}$ . Otherwise,  $\bigcup c'' = \bigcup c$ , because  $c$  is a chain. Then

$$pcl(\bigcup c) = pcl(\bigcup c'') = \bigcup c' \subseteq X = pcl(Y) \subseteq pcl(\bigcup c)$$

by monotonicity of  $pcl$ , whence  $\bigcup c' = X \in \hat{c}$ . Thus  $\hat{c}$  is Dedekind-complete.

Let  $A \in \hat{c}$  be non-maximal and  $B \in c$  be such that  $A = pcl(B)$ . Then  $B$  is non-maximal in  $c$ . Then there exists  $B' \in c$  such that  $B \subseteq B'$  and  $B \prec^+ B'$ . Then  $pcl(B) \in \hat{c}$ ,  $A \subseteq pcl(B')$ , and  $A \prec^+ pcl(B')$ , because  $|pcl(B')|^+ = |B'|^+$ .

**Lemma 14.** *If  $c \in Ch(S_{t_0}^+, \prec^+) \setminus \{\emptyset\}$  and  $|\bigcup c|^+ = +\infty$ , then there exists a trajectory  $s_* : [t_0, +\infty) \rightarrow Q$  and  $A \in c$  such that  $s_* \dot{=}_{t_0} s'$  for all  $s' \in A$ .*

*Proof.* Let  $c \in Ch(S_{t_0}^+, \prec^+) \setminus \{\emptyset\}$  and  $|\bigcup c|^+ = +\infty$ . Let  $\hat{c} = \{pcl(A) \mid A \in c\}$ . Because  $c \neq \emptyset$ , we have  $\hat{c} \in Ch(S_{t_0}^+, \prec^+) \setminus \{\emptyset\}$  by Lemma 13. Moreover,  $|\bigcup \hat{c}|^+ = +\infty$ , because  $c \subseteq \hat{c}$ .

Let us construct a  $\subseteq$ -monotone sequence  $A_n \in \hat{c}$ ,  $n \in \mathbb{N}$  and a sequence  $s_n \in A_n$ ,  $n \in \mathbb{N}$  as follows.

Lemma 8 implies that the function  $\xi$  has an inverse function  $\xi^{-1}$  which is defined and strictly increasing on  $[0, +\infty)$ . Moreover,  $\xi^{-1}(x) < x$  for all  $x > 0$ .

Let us choose  $A_1 \in \hat{c}$  arbitrarily and choose  $s_1 \in A_1$  in such a way that  $\sup dom(s_1) = \xi^{-1}(|A_1|^+)$  (this is possible, because  $A_1 \neq \emptyset$ ,  $0 < \xi^{-1}(|A_1|^+) < |A_1|^+$ , and  $A_1$  is prefix-closed, i.e.,  $pcl(A_1) = A_1$ ).

Suppose that elements  $A_1, \dots, A_n$  and  $s_1, \dots, s_n$  are already constructed. Let us construct  $A_{n+1}$ ,  $s_{n+1}$  in the following way.

1. Let  $C = \{A' \in \hat{c} \mid A_n \subseteq A', A_n \prec^+ A'\}$ . Then  $C \neq \emptyset$ , because  $A_n$  is not a  $\subseteq$ -maximal element of  $\hat{c}$  ( $\hat{c}$  has no maximal elements, because  $|\bigcup \hat{c}|^+ = +\infty$ ). Let  $A^* = \bigcup C$ . Then  $|A^*|^+ \leq \psi(|A_n|^+) < +\infty$ , because  $|A'|^+ < \psi(|A_n|^+)$  for all  $A' \in C$ . Let us choose  $X \in \hat{c}$  such that  $|X|^+ > |A^*|^+$ . Because  $\hat{c}$  is a  $\subseteq$ -chain,  $A' \subseteq X$  for all  $A' \in C \subseteq \hat{c}$ . Then  $A^* \subseteq X$ . Then  $A^* \in \hat{c}$ , because  $\hat{c}$  is Dedekind-complete. Then there exists  $B \in \hat{c}$  such that  $A^* \subseteq B$  and  $A^* \prec^+ B$ , because  $A^*$  is a non-maximal element of  $\hat{c}$  and  $\hat{c} \in Ch(S_{t_0}^+, \prec^+)$ . Let us define  $A_{n+1} = B$ . Then  $A_n \subseteq A_{n+1}$  and  $A_{n+1} \in \hat{c}$ .
2. Let us choose  $s \in A_{n+1}$  such that  $\sup dom(s) = \xi^{-1}(|A_{n+1}|^+)$  (this is possible, because  $A_{n+1} \neq \emptyset$ ,  $\xi^{-1}(|A_{n+1}|^+) < |A_{n+1}|^+$ , and  $A_{n+1}$  is prefix-closed). Then let us define  $s_{n+1} = s$ .

We have defined sequences  $A_n$  and  $s_n$ ,  $n \geq 1$ . The sequence  $A_n$  is obviously  $\subseteq$ -monotone. Let us show that for each  $n \geq 1$ ,

$$|A_n|^+ < \psi(|A_n|^+) \leq |A_{n+1}|^+ < \xi(|A_n|^+). \tag{4}$$

Let  $n \geq 1$ . Like above, let  $C = \{A' \in \hat{c} \mid A_n \subseteq A', A_n \prec^+ A'\}$  and  $A^* = \bigcup C$ . Then  $|A_{n+1}|^+ \leq \psi(\psi(|A_n|^+))$ , because  $|A^*|^+ \leq \psi(|A_n|^+)$  and  $A^* \prec^+ A_{n+1}$ .

Moreover,  $A_{n+1} \notin C$  and  $|A_*|^+ < |A_{n+1}|^+ \leq \psi(|A_*|^+)$ , because  $A^* \subseteq A_{n+1}$  and  $A^* \prec^+ A_{n+1}$ . Then  $A_n \not\prec^+ A_{n+1}$  by definition of  $C$ , because  $A_{n+1} \in \hat{c}$ , and  $A_n \subseteq A_{n+1}$ . Then  $\psi(|A_n|^+) \leq |A_{n+1}|^+$  or  $|A_{n+1}|^+ \leq |A_n|^+$  by definition of  $\prec^+$ . However,  $|A_{n+1}|^+ \geq |A^*|^+ > |A_n|^+$ , because  $C \neq \emptyset$ . Thus  $\psi(|A_n|^+) \leq |A_{n+1}|^+ \leq \psi(\psi(|A_n|^+))$ . From this and (3) we finally have (4).

The sequence  $|A_n|^+$  is monotone. If it is bounded from above, then its limit is a fixed point of  $\psi$ , because  $\psi$  is continuous. But  $\psi(x) > x$  for all  $x > 0$ , whence

$$\lim_{n \rightarrow \infty} |A_n|^+ = +\infty.$$

By construction of  $s_n$ ,  $\sup \text{dom}(s_n) = \xi^{-1}(|A_n|^+)$  for all  $n \geq 1$ , thus

$$\lim_{n \rightarrow \infty} \sup \text{dom}(s_n) = +\infty. \tag{5}$$

From (2) and Lemma 6 we have for all  $x \geq 0$ ,

$$g^+(\alpha(x), \xi(\xi(x))) = g^+(\alpha(x), f^+(\alpha(x), x)) = x > \alpha(x).$$

Let us prove that for each  $n \geq 1$ ,

$$s_n(t) = s_{n+1}(t) \text{ for all } t < \alpha(\sup \text{dom}(s_n)). \tag{6}$$

Let  $n \geq 1$  and  $x = \sup \text{dom}(s_n)$ ,  $a = |A_n|^+$ ,  $b = |A_{n+1}|^+$ . Then  $x = \xi^{-1}(a)$  and  $a < b < \xi(a)$  by (4). Then

$$x \geq g^+(\alpha(x), \xi(\xi(x))) = g^+(\alpha(x), \xi(a)) \geq g^+(\alpha(x), b).$$

by monotonicity of  $g^+$ . Then

$$\begin{aligned} \min\{\sup \text{dom}(s_n), \sup \text{dom}(s_{n+1})\} &= \min\{\xi^{-1}(|A_n|^+), \xi^{-1}(|A_{n+1}|^+)\} = \\ &= \xi^{-1}(|A_n|^+) = x \geq g^+(\alpha(x), |A_{n+1}|^+). \end{aligned}$$

Because  $x \in (0, |A_{n+1}|^+)$  and  $\alpha(x) \in (0, |A_{n+1}|^+)$ ,  $s_n, s_{n+1} \in A_{n+1}$ , and  $A_{n+1}$  is  $g^+$ -convergent, we have  $s_n(t) = s_{n+1}(t)$  for all  $t < \alpha(x) = \alpha(\sup \text{dom}(s_n))$ .

Let us define a function  $s_*$  on  $[t_0, +\infty)$  such that for each  $t \geq t_0$ ,  $s_*(t) = s_{m(t)}(t)$  where  $m(t) = \min\{n \mid t \in [t_0, \alpha(\sup \text{dom}(s_n))]\}$ . Because  $\alpha$  is unbounded and  $\alpha(y) < y$  for all  $y > 0$ , from (5) it follows that  $s_*(t)$  is defined for all  $t \geq t_0$ .

The sequence  $\sup \text{dom}(s_n)$ ,  $n \geq 1$  is monotone (by construction of  $s_n$ ) and  $\alpha$  is monotone, therefore (6) implies that  $s_m(t) = s_n(t)$  for all  $m, n \geq m$  and  $t < \alpha(\sup \text{dom}(s_m))$ . Then  $s_*(t) = s_n(t)$  for each  $t$  such that  $t < \alpha(\sup \text{dom}(s_{m(t)}))$  and  $n \geq m(t)$ . But  $t < \alpha(\sup \text{dom}(s_{m(t)}))$  for all  $t \geq t_0$ . Thus

$$s_*(t) = s_n(t) \text{ for all } t \geq t_0 \text{ and } n \geq m(t).$$

It is easy to see that the function  $m(t)$  is monotonically non-decreasing, so for each  $t \geq t_0$  and  $\tau \in [t_0, t + 1]$ ,  $s_*(\tau) = s_{m(t+1)}(\tau)$ , whence  $s_* \dot{=}_{t+s_{m(t+1)}}$  and  $s_* \dot{=}_{t-s_{m(t+1)}}$ . Let  $(l, r) \in LR(Q)$  be a pair of predicates such that  $\Phi(l, r) = Tr$  (it exists by Theorem 1). Because for all  $n$ ,  $s_n \in Tr$  and  $\min(\text{dom}(s_n)) = t_0$  (each

$s_n$  belongs to some right  $t_0$ -bunch), and for each  $t \geq t_0$ ,  $\sup(\text{dom}(s_{m(t+1)})) \geq t + 1$ , we have  $l(s_{m(t+1)}, t) \wedge r(s_{m(t+1)}, t)$  for all  $t \geq t_0$ . Then because  $l$  is left-local and  $r$  is right-local,  $l(s_*, t) \wedge r(s_*, t)$  for all  $t \geq t_0$ . Hence  $s_* \in \Phi(l, r) = Tr$ . Moreover,  $s_* \dot{=}_{t_0} s_{m(t_0+1)}$ . Let  $A \in c$  be a set such that  $A_{m(t_0+1)} = \text{pcl}(A)$ . Because  $A$  is a right  $t_0$ -bunch, we have  $s_* \dot{=}_{t_0} s'$  for all  $s' \in A \subseteq A_{m(t_0+1)}$ .

**Lemma 15.** *Assume that  $\Sigma$  satisfies WLE and each right dead-end path is  $f^+$ -escapable. Then for each  $X \in S_{t_0}^+$  there exists  $\bar{s} \in Tr$  such that  $X \cup \{\bar{s}\} \in S_{t_0}^+$  and  $X \prec^+ X \cup \{\bar{s}\}$ .*

*Proof.* Assume  $X \in S_{t_0}^+$ . Then  $X \neq \emptyset$  and  $|X|^+ < +\infty$ . Denote  $t_m = |X|^+$  and

$$H = \{(t, q) \in T \times Q \mid t \geq t_0 \wedge \\ \exists t' \in (t, t_m) \exists s \in X (s(t) \downarrow = q \wedge \sup(\text{dom}(s)) \geq g^+(t', t_m))\}$$

Denote  $H_1 = \{t \mid \exists q (t, q) \in H\}$ , i.e.,  $H_1$  is the domain of relation  $H$ .

Let us show that  $H_1 = [t_0, t_m)$ . The inclusion  $H_1 \subseteq [t_0, t_m)$  follows from the definition of  $H$ . Let  $t \in [t_0, t_m)$ . Let us choose an arbitrary  $t' \in (t, t_m)$ . Then  $g^+(t', t_m) < t_m$ . Because  $t_m = |X|^+$ , there exists  $s \in X$  such that  $\sup(\text{dom}(s)) \geq g^+(t', t_m) \geq t' > t$ . Because  $t \geq t_0$ , we have  $t \in \text{dom}(s)$  and  $(t, s(t)) \in H$ . Then  $[t_0, t_m) \subseteq H_1$ , because  $t \in [t_0, t_m)$  is arbitrary.

Let us show that  $H$  is a functional binary relation. Assume that  $(t, q_1) \in H$  and  $(t, q_2) \in H$ . Then there exist  $t'_1, t'_2$  and  $s_1, s_2 \in X$  such that  $t'_i \in (t, t_m)$ ,  $\sup(\text{dom}(s_i)) \geq g^+(t'_i, t_m)$ , and  $q_i = s_i(t)$  for  $i = 1, 2$ . Let  $t' = \min\{t'_1, t'_2\}$ . Then  $t' \in (t_0, |X|^+)$ , because  $t \geq t_0$ . Moreover,

$$\min\{\sup(\text{dom}(s_1)), \sup(\text{dom}(s_2))\} \geq \min\{g^+(t'_1, t_m), g^+(t'_2, t_m)\} \geq g^+(t', t_m),$$

by monotonicity of  $g^+$ . Then  $s_1(t) = s_2(t)$  for  $t \in [t_0, t')$ , because  $X$  is  $g^+$ -convergent. Then  $s_1(t) = q_1 = q_2 = s_2(t)$ , because  $t \in [t_0, t'_1)$  and  $t \in [t_0, t'_2)$ .

We conclude that  $H$  is a graph of some (total) function  $s_* : [t_0, t_m) \rightarrow Q$ . Let  $(l, r) \in LR(Q)$  be a pair such that  $\Phi(l, r) = Tr$  (it exists by Theorem [10](#)).

Let us show that  $s_* \in Tr$ . Let  $t \in (t_0, t_m)$ . Then  $(t, s_*(t)) \in H$  and there exists  $t' \in (t, t_m)$ , and  $s \in X$  such that  $s_*(t) = s(t)$  and  $\sup(\text{dom}(s)) \geq g^+(t', t_m) \geq t' > t$ . For each  $\tau \in [t_0, t')$ ,

$$\tau \geq t_0 \wedge t' \in (\tau, t_m) \wedge \sup(\text{dom}(s)) \geq g^+(t', t_m) \wedge s \in X$$

Then  $(\tau, s(\tau)) \in H$ . Hence  $s(\tau) = s_*(\tau)$  for all  $\tau \in [t_0, t')$ . Then  $s \dot{=}_{t_0} s_*$  and  $s \dot{=}_{t_0} s_*$ , because  $t \in (t_0, t')$ . Because  $t_0 < t < \sup(\text{dom}(s))$  and  $s \in Tr = \Phi(l, r)$ , we have  $l(s, t) \wedge r(s, t)$ . Then  $l(s_*, t) \wedge r(s_*, t)$ , because  $l$  is left-local and  $r$  is right-local. Thus  $l(s_*, t) \wedge r(s_*, t)$  for all  $t \in (t_0, t_m)$ . Moreover, because  $t_0 \in H_1$ , there exists  $t' \in (t_0, t_m)$  and  $s \in X$  such that  $\sup(\text{dom}(s)) \geq g^+(t', t_m)$  and  $s_*(t_0) = s(t_0)$ . Then  $t \geq t_0$  and  $t' \in (t, t_m)$  for each  $t \in (t_0, t')$ . Hence  $(t, s(t)) \in H$  for each  $t \in (t_0, t')$ . Then  $s(t) = s_*(t)$  for  $t \in [t_0, t')$ . Then  $r(s_*, t_0)$ , because  $r(s, t_0)$ . We conclude that  $s_* \in \Phi(l, r) = Tr$ . Moreover,  $s_* \dot{=}_{t_0} s$  for all  $s \in X$ , because  $X$  is a right  $t_0$ -bunch.

Consider the case when  $s_*$  is not a dead-end path, i.e., there exists a continuation of  $s_*$  to  $[t_0, t_m]$ . Then by WLE and CPR there exists  $t'_m \in (t_m, \psi(t_m))$  and a trajectory  $\bar{s} : [t_0, t'_m] \rightarrow Q$  such that  $\bar{s} \sqsubseteq s'_*$ . Then using monotonicity of  $g^+$  it is straightforward to show that  $X \cup \{\bar{s}\}$  is a bounded  $g^+$ -convergent right  $t_0$ -bunch (i.e.,  $X \cup \{\bar{s}\} \in S_{t_0}^+$ ), and  $X \prec^+ X \cup \{\bar{s}\}$ .

Consider the case when  $s_*$  is a right dead-end path. Then  $s_*$  is  $f^+$ -escapable. Let us choose  $\tau \in (t_0, t_m)$  such that  $f^+(\tau, t_m) < \psi(t_m)$  (this is possible, because  $f^+(t_m, t_m) = t_m$ ,  $f^+(\cdot, t_m)$  is continuous on  $(t_0, t_m]$ , and  $\psi(t_m) > t_m$ ).

CPR and Lemma 2 imply that there exists an escape  $s_e$  from  $s_*$  of the form  $s_e : [t_e, t'_e] \rightarrow Q$ , where  $t_e \in (\tau, t_m)$  and  $t'_e = f^+(t_e, t_m)$ . Because  $t_e \geq \tau$ , we have

$$t_m < t'_e = f^+(t_e, t_m) \leq f^+(\tau, t_m) < \psi(t_m).$$

Let us define a function  $\bar{s} : [t_0, t'_e] \rightarrow Q$  as follows:

$$\bar{s}(t) = \begin{cases} s_*(t), & t \in [t_0, t_e) \\ s_e(t), & t \in [t_e, t'_e] \end{cases}$$

Markovian property implies that  $\bar{s} \in Tr$ . Moreover,  $X \cup \{\bar{s}\}$  is a bounded right  $t_0$ -bunch, because  $\bar{s} \dot{=}_{t_0} s_*$  and  $dom(\bar{s})$  is bounded. Also,  $X \prec^+ X \cup \{\bar{s}\}$ , because  $|X|^+ = t_m < t'_e = |X \cup \{\bar{s}\}|^+ < \psi(t_m) = \psi(|X|^+)$ .

Let us prove  $X \cup \{\bar{s}\}$  is  $g^+$ -convergent. Assume that  $t' \in (t_0, t'_e)$ ,  $s_1, s_2 \in X \cup \{\bar{s}\}$ ,  $t_i = \sup(dom(s_i))$ ,  $i = 1, 2$ , and  $\min\{t_1, t_2\} \geq g^+(t', t'_e)$ . Let us show that  $s_1 \dot{=}_{[t_0, t']} s_2$ . Consider the following cases.

- Suppose that  $s_1, s_2 \in X$  and  $t' < t_m$ . Then  $\min\{t_1, t_2\} \geq g^+(t', t'_e) \geq g^+(t', t_m)$ . Then  $s_1(t) = s_2(t)$  for all  $t \in [t_0, t')$ , because  $X$  is  $g^+$ -convergent.
- Suppose that  $s_1, s_2 \in X$  and  $t' \geq t_m$ . Then  $\min\{t_1, t_2\} \geq g^+(t', t'_e) \geq t' \geq t_m$ . Then  $t' = t_1 = t_2 = t_m$ , because  $s_1, s_2 \in X$ . The definition of  $H$  implies that  $(t, s_1(t)) \in H$  and  $(t, s_2(t)) \in H$  for all  $t \in [t_0, t_m)$ , because  $t_m \geq g^+(t'', t_m)$  for all  $t'' < t_m$ . Thus  $s_1 \dot{=}_{[t_0, t']} s_2$ .
- Suppose that  $\{s_1, s_2\} \not\subseteq X$ . The case  $s_1 = s_2 = \bar{s}$  is trivial, so assume either  $s_1 \in X$  and  $s_2 = \bar{s}$ , or  $s_2 \in X$  and  $s_1 = \bar{s}$ . We consider only the former case, because the latter case is analogous. Let  $s_1 \in X$ ,  $s_2 = \bar{s}$ . Then  $t_m \geq t_1 = \min\{t_1, t_2\} \geq g^+(t', t'_e) \geq t'$ . Also,  $t_1 \geq g^+(t', t_m)$  because  $t' \leq t_m < t'_e$ . We have  $(t, s_1(t)) \in H$  for all  $t \in [t_0, t')$  by definition of  $H$ , because  $t_m \geq g^+(t'', t_m)$  for all  $t'' < t_m$ . Hence  $s_1 \dot{=}_{[t_0, t']} s_*$ . Assume that  $t' > t_e$ . Then  $t_1 \geq g^+(t', t'_e) \geq g^+(t_e, t'_e) \geq t_m$ , because  $t'_e = f^+(t_e, t_m)$ . Then  $g^+(t', t'_e) = g^+(t_e, t'_e) = t_m$ . Then  $t_e \geq t'$  by monotonicity of  $g^+$ . This contradicts assumption  $t' > t_e$ . Thus  $t' \leq t_e$ . Moreover,  $s_*(t) = \bar{s}(t)$  for all  $t \in [t_0, t_e]$ . Thus  $s_1(t) = \bar{s}(t) = s_2(t)$  for all  $t \in [t_0, t')$ .

Now we have lemmas that are necessary to prove Theorem 2.

*Proof (of Theorem 2).* The "Only if" part of theorem follows from CPR, so let us show the "If" part. Let  $s : [t_0, t^*] \rightarrow Q$  be a right dead-end path in  $\Sigma$ ,  $A_0 = \{s\}$ , and  $c_0 = \{A_0\}$ . It is easy to see that  $c_0 \in Ch(S_{t_0}^+, \prec^+)$ . From Lemma 9 and Zorn's lemma it follows that there exists a maximal element  $c_m \in Ch(S_{t_0}^+, \prec^+)$

(with respect to  $\leq$ ) such that  $c_0 \leq c_m$ . Let  $X = \bigcup c_m$ . Then  $X \neq \emptyset$ ,  $c_m \neq \emptyset$ , and  $A_0 \subseteq X$ , because  $c_0 \subseteq c_m$ . Assume  $|X|^+ < +\infty$ . Then  $X \in S_{t_0}^+$  by Lemma [11](#). Then by Lemma [15](#) there exists  $\bar{s}$  such that  $X \cup \{\bar{s}\} \in S_{t_0}^+$  and  $X \prec^+ X \cup \{\bar{s}\}$ . Then  $X \subset X \cup \{\bar{s}\}$ , but this contradicts Lemma [10](#). Thus  $|X|^+ = +\infty$ . Then by Lemma [14](#) there exists a trajectory  $s_* : [t_0, +\infty) \rightarrow Q$  and  $A \in c_m$  such that  $s_* \doteq_{t_0} s'$  for all  $s' \in A$ . Because  $c_m$  is a  $\subseteq$ -chain,  $A \neq \emptyset$ , and  $\{s\} \in c_0 \subseteq c_m$ , we have  $s \in A$ . Then  $s_* \doteq_{t_0} s$ . Then  $s_*$  is an infinite escape from  $s$ .

## 5 Conclusion

We have studied the question of existence of a global-in-time trajectory for each initial condition for non-deterministic complete Markovian systems. We have shown that this question can be answered using analysis of existence of locally defined trajectories of the system in a neighborhood of each time.

The obtained results can be useful for proving existence of trajectories of non-deterministic continuous-time systems such as differential inclusions, switched systems, etc.

## References

1. Coddington, E., Levinson, N.: Theory of ordinary differential equations. McGraw-Hill, New York (1955)
2. Fillipov, A.: Differential equations with discontinuous right-hand sides. AMS Trans. 42, 199–231 (1964)
3. Gliklikh, Y.: Necessary and sufficient conditions for global-in-time existence of solutions of ordinary, stochastic, and parabolic differential equations. Abstract and Applied Analysis 2006, 1–17 (2006)
4. Tangiguchi, T.: Global existence of solutions of differential inclusions. Journal of Mathematical Analysis and Applications 166, 41–51 (1992)
5. Aubin, A., Cellina, A.: Differential inclusions. Springer, Berlin (1984)
6. Seah, S.W.: Existence of solutions and asymptotic equilibrium of multivalued differential systems. J. Math. Anal. Appl. 89, 648–663 (1982)
7. Heemels, W., Camlibel, M., Van der Schaft, A.J., Schumacher, J.M.: On the existence and uniqueness of solution trajectories to hybrid dynamical systems. In: Johansson, R., Rantzer, A. (eds.) Nonlinear and Hybrid Control in Automotive Applications, pp. 391–422. Springer, Berlin (2003)
8. Goebel, R., Sanfelice, R., Teel, R.: Hybrid dynamical systems. IEEE Control Systems Magazine 29, 29–93 (2009)
9. Henzinger, T.: The theory of hybrid automata. In: IEEE Symposium on Logic in Computer Science, pp. 278–292 (1996)
10. Doob, J.B.: Stochastic processes. Wiley-Interscience (1990)
11. Willems, J.: Paradigms and Puzzles in the Theory of Dynamical Systems. IEEE Transactions on Automatic Control 36, 259–294 (1991)
12. Kuczma, M., Choczewski, B., Ger, R.: Iterative Functional Equations. Cambridge University Press, Cambridge (1990)
13. Constantin, A.: Global existence of solutions for perturbed differential equations. Annali di Matematica Pura ed Applicata 168, 237–299 (1995)

# Combining Verification and MDE Illustrated by a Formal Java Development

Selma Djedjai<sup>1,\*</sup>, Mohamed Mezghiche<sup>2</sup>, and Martin Strecker<sup>1,\*</sup>

<sup>1</sup> IRIT (Institut de Recherche en Informatique de Toulouse)  
Université de Toulouse  
118 Route de Narbonne, 31062 Toulouse Cedex 9, France  
`{firstname.lastname}@irit.fr`

<sup>2</sup> LIMOSE, UMBB, Boumerdès, Algeria

**Abstract.** Formal methods are increasingly used in software engineering. They offer a formal frame that guarantees the correctness of developments. However, they use complex notations that might be difficult to understand for unaccustomed users. It thus becomes interesting to formally specify the core components of a language, implement a provably correct development, and manipulate its components in a graphical/textual editor.

This contribution constitutes a first step towards using Model Driven Engineering (MDE) technology in an interactive proof development. It presents a transformation process from functional data structures, commonly used in proof assistants, to Class diagrams in Ecore. To perform the transformation we use an MDE-based methodology. The resulting metamodels from the transformation process are used to generate textual or graphical editors for domain specific languages (DSLs) using tools provided by the Eclipse environment. To illustrate this approach we use as example a simple DSL description. It represents a Java-like language enriched with timing annotations.

**Keywords:** Model Driven Engineering, Model Transformation, Formal Methods, Verification.

## 1 Introduction

Domain Specific Languages (DSL) have conquered many different aspects of computer science. They are used in different fields such as aerospace, web-services, multi-media, etc. [8]. Certain DSLs define their semantics in natural languages. However, even though these tend to be quite easy to understand, they usually suffer from incompleteness in some cases and ambiguity in others. Therefore, there emerges a need for defining the formal semantics of DSLs in a mathematically founded framework using proof assistants. Such a phase consists in defining the abstract syntax of a DSL and then grafting a semantics on top

---

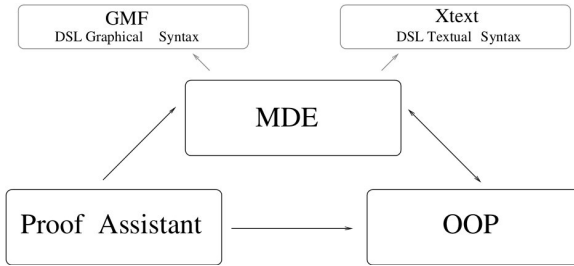
\* Part of this research has been supported by the project *Verisync* (ANR-10-BLAN-0310).



of it, using well-understood mechanisms like structural recursion or inductive relations. Such a semantics is often not executable, but other elements of a formal development are, such as compilers or static analyses whose correctness is proved on the basis of the formal semantics.

Interactive proof assistants such as Coq [6] or Isabelle [18] often use paradigms stemming from functional programming (type systems, function definitions), but they are as such not a programming language. It is however possible to export the formal development to programming languages such as Caml [17] or Scala [19]. A formally verified compiler, for example, can therefore be effectively executed in a standard programming language.

In order to improve the user interface for interacting with a DSL, we aim at a textual or graphical concrete syntax as provided, for example, by the Eclipse Xtext or GMF environments. Frequent changes of the DSL during the design phase make it necessary to adapt this interface easily and to re-generate it automatically, as far as possible.



**Fig. 1.** Meta-modeling(MM), Verification environment and OO languages

Figure 1 depicts the essence of our approach based on studying the interplay of three formalisms that offer different and complementary aspects. On one hand, we have Model Driven Engineering (MDE) [4, 22] that supplies us with frameworks (for example Eclipse Modeling Framework) allowing to specify, visualize and understand DSLs. Also these frameworks are equipped with tools that permit to define graphical and textual syntax for these DSLs (Xtext or Graphical Modeling Framework GMF). They are rather close to Object Oriented programming which is the choice when it comes to developing graphical user interfaces. Besides these facilities, they often suffer from lack of precise semantics.

On the contrary, proof assistants (such as Isabelle) have solid formal bases and precise semantics. They are increasingly used to verify the correctness of software. Nevertheless, they use complex notations that might be difficult to understand for a non-initiated public.

Thus, this work constitutes a first step towards using MDE technology in an interactive proof development. The guiding example (see Section 3) is a Java-like language enriched with assertions developed by ourselves for which no off-the-shelf definition exists. This “meta-model” (in MDE parlance) is sufficiently

complex to illustrate the method and to be a case study of realistic size for a DSL. However, its formal model can be entirely defined as an inductive datatype (and this is so for most formally defined languages). In this case study, we can therefore not demonstrate some aspects of our work, such as the translation of genuine graph structures that go beyond instances of inductive data types.

Section 2 constitutes the technical core of the article; it describes a translation from data models in the functional programming world, used in verification environments, to meta models in *Ecore*: the core language of the Eclipse Modeling Framework. We illustrate the methodology in Section 3 with a case study. In Section 4 we compare our work to other approaches, before concluding in Section 5 with perspectives of further work.

## 2 From Datatypes to Meta-models

In this part, we present in detail the translation process from functional data types to meta-models. We start in Section 2.1 by giving an overview of our methodology, then we introduce the source and the target of the transformation in Sections 2.2 and 2.3 respectively. The essence of the translation is further developed in Section 2.4.

### 2.1 Methodology

Model Driven Engineering (MDE) is a software development methodology where the (meta-)models are the central elements in the development process. A meta-model defines the elements of a language. The instances of these elements are used to construct a model of the language. A model transformation is defined by a mapping from elements of the source meta-model to those of the target meta-model. Consequently, each model conforming to the source meta-model can be automatically translated to an instance model of the target meta-model. The Object Management Group (OMG) [20] defined the Model Driven Architecture (MDA) standard [15], as specific incarnation of the MDE.

We apply this method in order to define a generic transformation process from datatypes (used in ML-style languages and interactive provers) into *Ecore* models. Figure 2 shows an overview of our approach. Using an EBNF representation of the datatype definition grammar [18], we derive a meta-model of datatypes. This meta-model is the source meta-model of our transformation. We also define a subset of the *Ecore* meta-model [12] to be the target meta-model. The transformation rules are defined on the meta-level and map elements from the source meta-model to their counterparts in the target meta-model. They are detailed in Section 2.4. The *DataTypeToEcore* function implements these rules in Java. It takes as input models which conform to the source meta-model and returns their equivalent in a model which conforms to the target meta-model. The implementation process is further developed in Section 3.2.

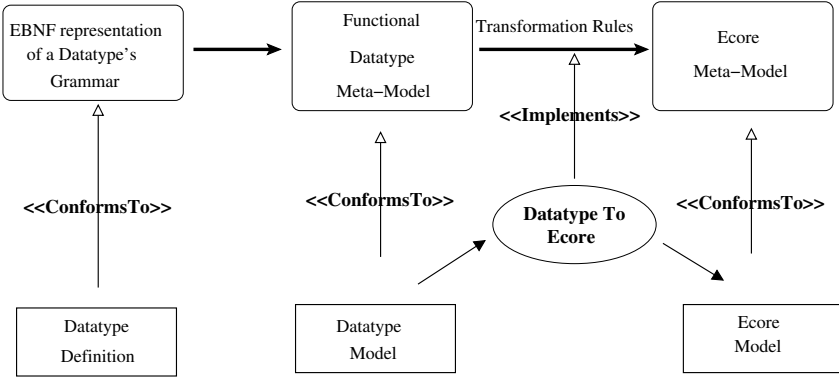


Fig. 2. Overview of the Transformation Method

## 2.2 Source Meta-model: The Datatype Meta-model

Functional programming is a programming paradigm that implements  $\lambda$ -calculus: a formal system in mathematical logic that formalizes systems through the notion of function. A function, in functional programming, consists in the mapping of elements from a set to another. These sets are called *types*. Usually, they restrict the set of legal programs. We can count among the languages implementing functional programming: *Lisp* [24], *Haskell* [21], and the *ML languages*.

We are interested in *ML languages*. ML stands for Meta Language. It is based on a user-friendly syntax of  $\lambda$ -calculus augmented with polymorphism. It is known for its ability to automatically infer the types of expressions without explicit type annotations. ML languages are considered as non purely functional languages. In fact, they admit the use of mutable data structures, features allowing to program in an imperative way. The most famous dialects of the ML family are SML (Standard ML) and OCaml (Objective Caml) [17].

To perform the transformation, taking all the features provided by ML languages, would be unnecessarily complex, because some features which are specific to functional programming are not used in MDE modeling and would have no equivalent supported by *Ecore*. This is why we defined a subset of data structure schemas provided by ML languages that allows to define data types and that is convenient to be translated into *Ecore* models.

In this subset, we treat primitive types (integers, Booleans, floats and strings) and user defined data types. We allow the use of some keywords introducing lists, references and type option. However, we do not handle mutable constructs and mutable data structures (including arrays). Also for now, we do not implement a specific treatment for mutually recursive types.

Figure 3 depicts the datatype meta-model that is constructed from the subset of datatype declaration grammars of typical functional languages [17, 18]. To construct this meta-model we were inspired by the work of [1] and [25]. They worked widely on defining generic processes to transform EBNF grammars into

Meta-models and vice-versa. We mainly focused on the definition of transformation rules and the correspondence between the elements of the two formalisms. However, we did not use any tools or algorithms developed.

In our subset represented by the meta-model depicted by Figure 3, a *Module* may contain several *Type Definitions*. Each *Type Definition* has a *Type Constructor*. It corresponds to the data types' name. It is also composed of at least one *Constructor Declaration*. These declarations are used to express variant types. *Type declarations* have names, it is the name of a particular type case. It takes as argument some (optional) *type expressions* which can either represent a *Primitive Type* (*int*, *bool*, *float*, etc.) or also a data type defined previously in the module. The *list* option is used to represent lists in functional programming. The *type option* feature describes the presence or the absence of a value. The *ref* option is used for references (pointers).

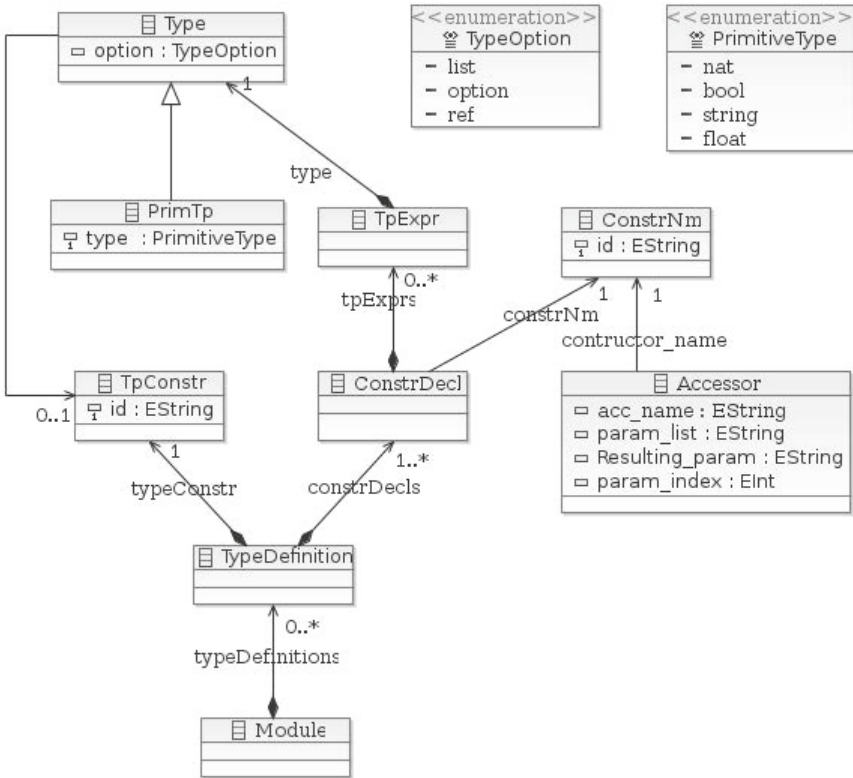


Fig. 3. Datatype Meta-model

We can notice that elements composing type definitions are often unnamed and just expressed with *type expressions*. However, for the rest of our work these typed elements have to be distinguishable by their names. Therefore, we

enriched the type definition grammar with a new element named *Accessor*. It is a function introduced by a special annotation (`*@accessor*`). It allows to assign a name to a special part of the type declaration. These accessor functions are essential for the transformation process, their absence would lead to nameless `EStructuralFeatures`. The syntax of these functions in the OCaml language is presented in Figure 4.

```
(*@ accessor *)
  let acc_namei ([constr-name] (x1, ..., xn)) = xi / 1 ≤ i ≤ n
```

Fig. 4. Syntax of Accessor functions in OCaml

### 2.3 Target Meta-model: The Ecore Meta-model

Eclipse Modeling Framework (EMF) is an Eclipse framework for building applications based on model definitions. It unifies three technologies: Java, XML and UML. It allows to describe a model as a class diagram, class interfaces in the Java programming language or in the form of an XML schema. Moreover, it is possible to describe a model and generate it in the two others.

**Ecore** is the model that is used to describe and handle models in EMF. It has been developed as a small and simplified implementation of full UML. Its main components are:

- The **EPackage** is the root element in serialized **Ecore** models. It encompasses **EClasses** and **EDataTypes**.
- The **EClass** component represents classes in **Ecore**. It describes the structure of objects. It contains **EAttributes** and **EOperations**.
- The **EDataType** component represents the types of **EAttributes**, either pre-defined (types: Integer, Boolean, Float, etc.) or defined by the user. There is a special datatype to represent enumerated types **EEnum**, each enumeration is called **EEnumLiteral**.
- **EReferences** is comparable to the UML Association link. It defines the kinds of the objects that can be linked together. The **containment** feature is a Boolean value that makes a stronger type of relations. When it is set to true, it represents a whole/part relationship known as “by-value aggregation” in UML.

The Meta Object Facility (MOF) standardized by the OMG defines a subset of UML class diagram [11]. It represents the Meta-Meta-Model of UML. **Ecore** is comparable to MOF but simpler. They are similar in their ability to specify classes, structural and behavioral features, inheritance and packages. However, their difference appears in the data type structures, package relationships and complex aspects of association links. EMOF (Essential Meta-Object Facility) is the new core meta-model that is very close to Ecore [5].

Figure 5 represents a subset of the *Ecore* language. This subset contains essentially the elements that are needed for the transformation process. In this meta-model appear only basic classes features and operation. The items that do not appear are not used by our transformation process.

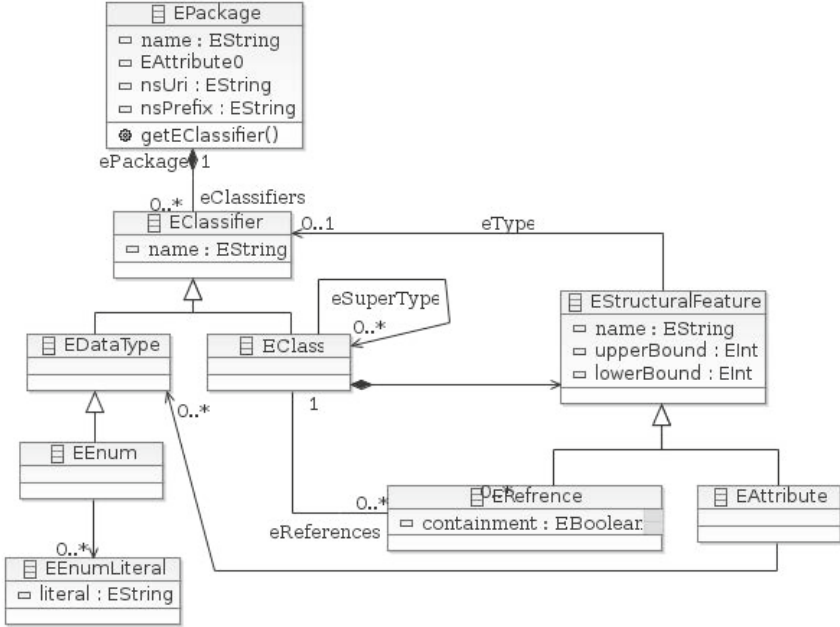


Fig. 5. Simplified subset of the *Ecore* Meta-model

## 2.4 From Datatypes to Meta-models

The transformation method is from functional datatypes to *Ecore* meta-models. To precisely define transformation rules, the transformation method is presented in a formal notation in the form of a function noted  $Tr()$ . The transformation rules are presented as sub-functions relatively to the component given as input. In each rule definition, we start by an informal description, then we present it formally and finally we show an effective example.

$$Tr : DataTypes \longrightarrow Ecore\ Meta-model$$

The following translation sub-functions are given for a concrete syntax in the style of Caml [17]. Since most functional languages (including the language of proof assistants) have great similarities, the concrete syntax can be mapped to different functional languages.

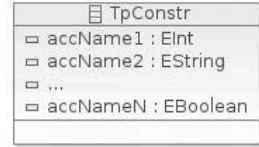
**Rule DatatypeToEClass.** This rule is applied when the datatype is formed of only one constructor. the latter is translated to an **EClass**. The EClass name is the name of the type constructor. The types composing the datatype are translated using other rules (PrimitiveTypeToEAttribute or TypeToEReference).

$$\begin{aligned}
 Tr(tpConstr = cn\ t_1\dots t_n) &= createEClass(); \\
 &\quad setName(tpConstr); \\
 &\quad Tr_{type}(acc_i, t_i) \\
 &\quad / 1 \leq i \leq n
 \end{aligned}$$

**Example:**

```

type tpConstr =
  Cn of int * string * ... * bool
    
```



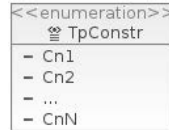
**Rule DatatypeToEEnum.** Datatypes composed only of constructors (without type expressions *typeexpr*) are translated to **EEnums** which are usually employed to model enumerated types in **Ecore**. Then, each constructor composing the datatype is translated into a literal named **EEnumLiteral**. The name of each constructor becomes the name of a literal.

$$\begin{aligned}
 Tr(tpConstr = cn_1|\dots|cn_p) &= createEEnum(); \\
 &\quad setName(tpConstr); \\
 &\quad Tr_{constrNm}(cn_i) \quad / 1 \leq i \leq p \\
 Tr_{constrNm}(cn_i) &= EEnumLiteral(cn_i) \quad / 1 \leq i \leq p
 \end{aligned}$$

**Example:**

```

type tpConstr=
  Cn1 | Cn2 | ... | CnN
    
```



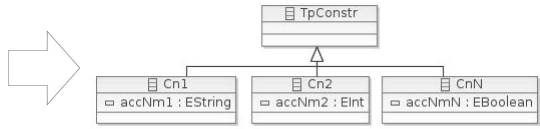
**Rule DatatypeToEClasses.** When constructor declarations are composed of more than one constructor declaration containing type expressions: a first **EClass** is created to represent the type constructor (*tpConstr*). Then, for each constructor, an **EClass** is created too, and inherits from the *tpConstr* one. To transform the types expressions of each constructor, we call the functions for translating the type expressions.

$$\begin{aligned}
 Tr(tpConstr = cd_1 | \dots | cd_n) &= createEClass(); \\
 & \quad setName(tpConstr); \\
 & \quad Tr_{decl}(cd_i) \\
 & \quad / 1 \leq i \leq n \\
 Tr_{decl} : ConstructorDeclaration &\longrightarrow EClass \\
 Tr_{decl}(cn_i \ t_1 \dots t_m) &= createEClass(); \\
 & \quad setName(cn_i); \\
 & \quad setSuperType(EClass(tpConstr)); \\
 & \quad Tr_{type}(acc_j, t_j) \\
 & \quad / 1 \leq j \leq m
 \end{aligned}$$

**Example:**

```

type tpConstr =
  Cn1 of string
  | Cn2 of int
  | ...
  | CnN of bool
    
```



**Rule PrimitivTypeToEAttribute.** If a type expression is formed of a primitive type, the translation function generates a new **EAttribute**. The name of this **EAttribute** is the name of its corresponding accessor, and its type is the EMF representation of the the primitive type : **EInt** for *int*, **EBoolean** for *bool*, **EString** for *string*, etc.

$$\begin{aligned}
 Tr_{type} : (accessor, type) &\longrightarrow EStructuralFeature \\
 Tr_{type}(acc, primTp) &= createEAttribute(); \\
 & \quad setName(acc); \\
 & \quad setType(primTp_{EMF});
 \end{aligned}$$

**Example:**

```

type tpConstr =
  Cn of int * string * ... * bool
    
```



**Rule TypeToEReference.** When a type expression contains a type which is not a primitive type, the latter has to be previously defined in the Isabelle *theory*. Then, a containment link is created between the current **EClass** and the **EClass** referenced by type constructor, and the multiplicity is set to 1.

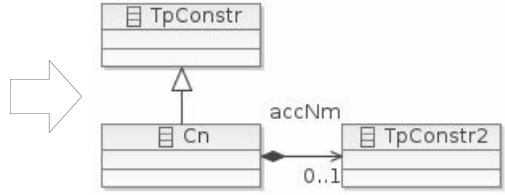


$$\begin{aligned}
 Tr_{type} : (accessor, type) &\longrightarrow EStructuralFeature \\
 Tr_{type}(acc, tpConstr) &= createEReference(); \\
 & \quad setName(acc); \\
 & \quad setType(tp\_constr); \\
 & \quad setContainment(true); \\
 & \quad setLowerBound(1); \\
 & \quad setUpperBound(1);
 \end{aligned}$$
**Example:**

```

type tpConstr=
  Cn of tpConstr2

```



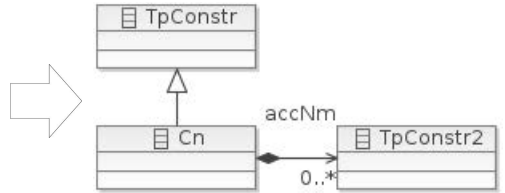
**Rule TypeOptionToMultiplicity.** The type expressions can also appear in the form of a *type list*. In this case the multiplicity is set to  $0..*$ . The type expression *type option* is used to express whether a value is present or not. It returns `None`, if it is absent and `Some` value, if it is present. This is modeled by changing the cardinality to  $0..1$ .

$$\begin{aligned}
 Tr_{type} : (accessor, type) &\longrightarrow EStructuralFeature \\
 Tr_{type}(acc, t \text{ list}) &= Tr_{type}(acc, t) \\
 & \quad setLowerBound(0); \\
 & \quad setUpperBound(*); \\
 Tr_{type}(acc, t \text{ option}) &= Tr_{type}(acc, t) \\
 & \quad setLowerBound(0); \\
 & \quad setUpperBound(1);
 \end{aligned}$$
**Example:**

```

type tpConstr=
  Cn of tpConstr2 list

```

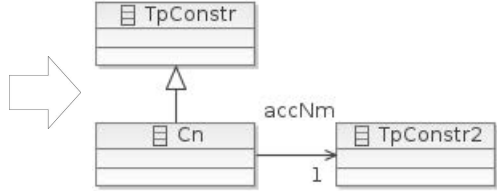


The last case that we deal with is references (*type ref*). References are used to represent pointers in ML programming and Isabelle. It is translated to simple references without containment option in Ecore.

$$Tr_{type}(acc, t \text{ ref}) = Tr_{type}(acc, t) \text{ setContainment}(False);$$

**Example:**

```
type tpConstr=
  Cn of tpConstr2 ref
```



**Rule AccessorToStructuralFeaturesName.** This rule is spelled out to define how the *accessor\_name* is selected for naming a particular **EStructuralFeature**. Accessors are regrouped in *accessors\_List*. Each accessor structure is formed of an *accessor\_name*, a *constructor\_name* and an integer value named "index". This index corresponds to the place of the type the accessor is accessing in the type expressions.

The *constructor\_name* is used to select the corresponding **EClass** where the **EStructuralFeature** is created. Then the index value is compared to the value **FeatureID** given by **Ecore** to represent the rank of the **EStructuralFeature** creation in a particular **EClass**. When these values are equal, the corresponding accessor's name is selected to name this **EStructuralFeature**.

**Example:**

```
type tp1= Constr1 of int
| Constr2 of (int list)* bool
```

```
type tp2 = Tp2 of tp1 * string
```

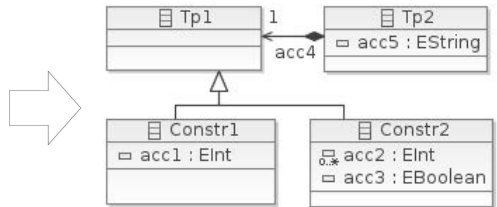
```
(*@accessor*)
let acc1 (Constr1 (x)) =x ;;
```

```
(*@accessor*)
let acc2 (Constr2 (x,y)) =x ;;
```

```
(*@accessor*)
let acc3 (Constr2 (x,y)) =y ;;
```

```
(*@accessor*)
let acc4 (Tp2 (x,y)) =x ;;
```

```
(*@accessor*)
let acc5 (Tp2 (x,y)) =y ;;
```



### 3 Case Study

In this section, we apply the method presented in Section 2 on a detailed example that consist of a Domain Specific Language. We start by the DSL definition, then we show the architecture of the application before finishing with the effective results of the transformation.

#### 3.1 Presentation of the Case Study

We are currently working on a real-time dialect of the Java language allowing us to carry out specific static analyses of Java programs. We only sketch this language here; details are described in [3]. This language is not a genuine subset of Java, since we have added annotations characterizing timing behavior of program parts that are inserted in particular comments into the program. Neither is the language a superset of Java, because we have to impose syntactic restrictions on the shape of the program, and also static restrictions on the number of objects that are allocated.

All this made us opt for writing our own syntax analysis, which is integrated into the Eclipse Xtext environment [9]. After syntax analysis and verification of the above-mentioned static restrictions, the program together with its timing annotations is translated to Timed Automata (TA) for model checking. The language is currently not entirely stable and will be modified while we refine and improve the translation from Java to TA, and while the formal model evolves.

The formal aspect comes into play at the following point: We are currently developing a real-time semantics of Java in the proof assistant Isabelle, based on an execution semantics using inductive relations. Performing the translation for the whole language description would generate a huge meta-model that couldn't be presented in the contribution. We thus choose to present only an excerpt of it, corresponding to a method definition.

Figure 7 shows the datatype definitions in the Isabelle proof assistant, where a method definition is composed of a method declaration, a list of variables, and statements. Each method declaration has an access modifier that specifies its kind. It also has a type, a name, and some variable declarations. The *stmt* datatype describes the statements allowed in the method body: Assignments, Conditions, Sequence of statements, Return and the annotation statement (for timing annotations). In this example we use Booleans, integers, strings for types and values.

#### 3.2 Implementation: DatatypesToEcore

Our approach is implemented using the Eclipse environment which includes among others

- Eclipse Modeling Framework (EMF) [5]: a framework for modeling and code generation that builds tools and applications based on data models.

- Eclipse Modeling Project (EMP) [12]: a framework allowing the manipulation of DSLs by defining their (textual/graphical) concrete syntax based on a corresponding meta-model using Xtext or GMF tools.

In this chapter we use the Xtext tool [9]. It is a tool that supports the development of textual concrete syntax for DSLs. In the first versions of Xtext, it was only possible to create a DSL textual editor starting from an Extended Backus-Naur Form-like grammar and generating a corresponding Ecore-based meta-model. But since Xtext 2.0, it is possible to start from a meta-model and get the corresponding EBNF-like grammar. Starting from this grammar, the generator creates a parser as well as a functional Eclipse textual editor, complete with syntax highlighting, code assist and outline view [12].

Figure 6 shows the architecture of our application. Non-dashed arrows represent automatic model transformations or code generation. On the contrary, the dashed one stands for a manual intervention added to Xtext code generation facilities. In our approach, the base element is an Isabelle *theory* where both of the datatypes and the properties to be checked are defined. The corresponding meta-model is generated using the translation function described in Section 2.4. Starting from a generated Ecore meta-model, we use the Xtext tool to define a textual concrete syntax. First, Xtext builds an EBNF grammar depending on the structure of the meta-model. The grammar is then adapted using the right key words of the language, yielding a textual editor as an Eclipse plug-in. We thus generated code for a DSL textual tool.

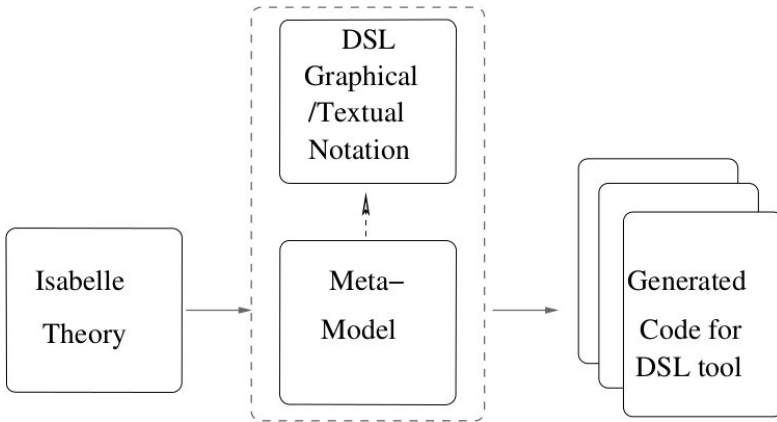


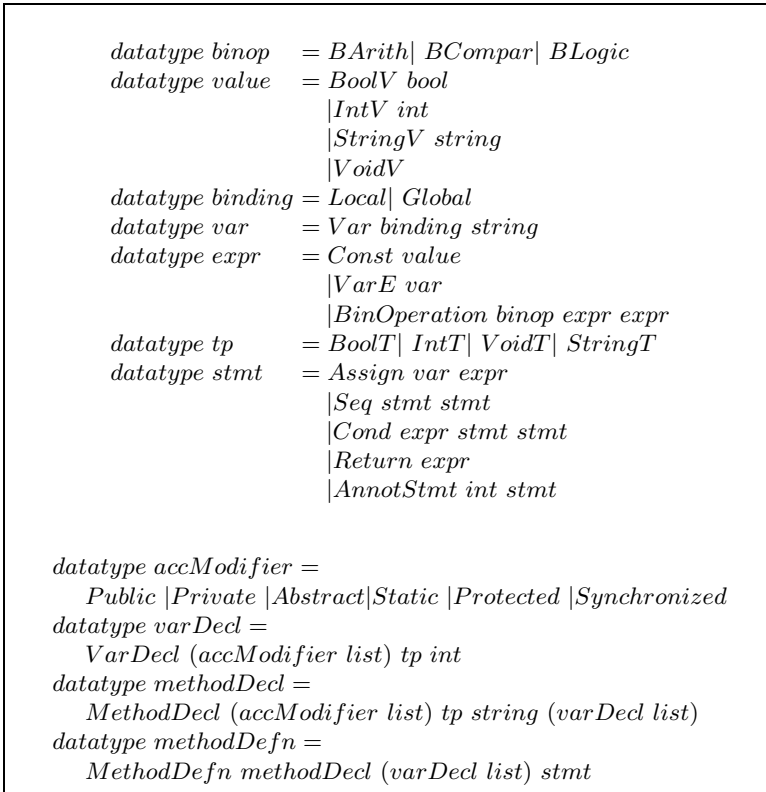
Fig. 6. Datatype To Ecore implementation architecture

### 3.3 Applying the Transformation

Figure 7 shows datatypes taken from the Isabelle *theory* where the verifications were performed. These datatypes are used to express the elements of a

method declaration in our DSL. This part of the *theory* was given as input to the implementation of our translation rules presented in Section 2.4. The resulting Ecore diagram is presented in Figure 8.

As it is shown on the figure, data type definitions built only of type constructors (*Tp*, *AccModifier*, *Binop*, *Binding*) are treated as enumerations in the meta-model, whereas *Datatype MethodDecl* composed of only one constructor derive a single class. As for type expressions that represent list of types (like *accModifier list* in *varDecl*), they generate a structural feature in the corresponding class and their multiplicities are set to  $(0..*)$ . The result of type definitions containing more than one constructor and at least a type expression (*stmt* and *expr*) is modeled as a number of classes inheriting from a main one. Finally, the translation of the *int*, *bool* and *string* types is straightforward. They are translated to respectively *EInt*, *EBoolean* and *EString*.



**Fig. 7.** Datatypes in Isabelle

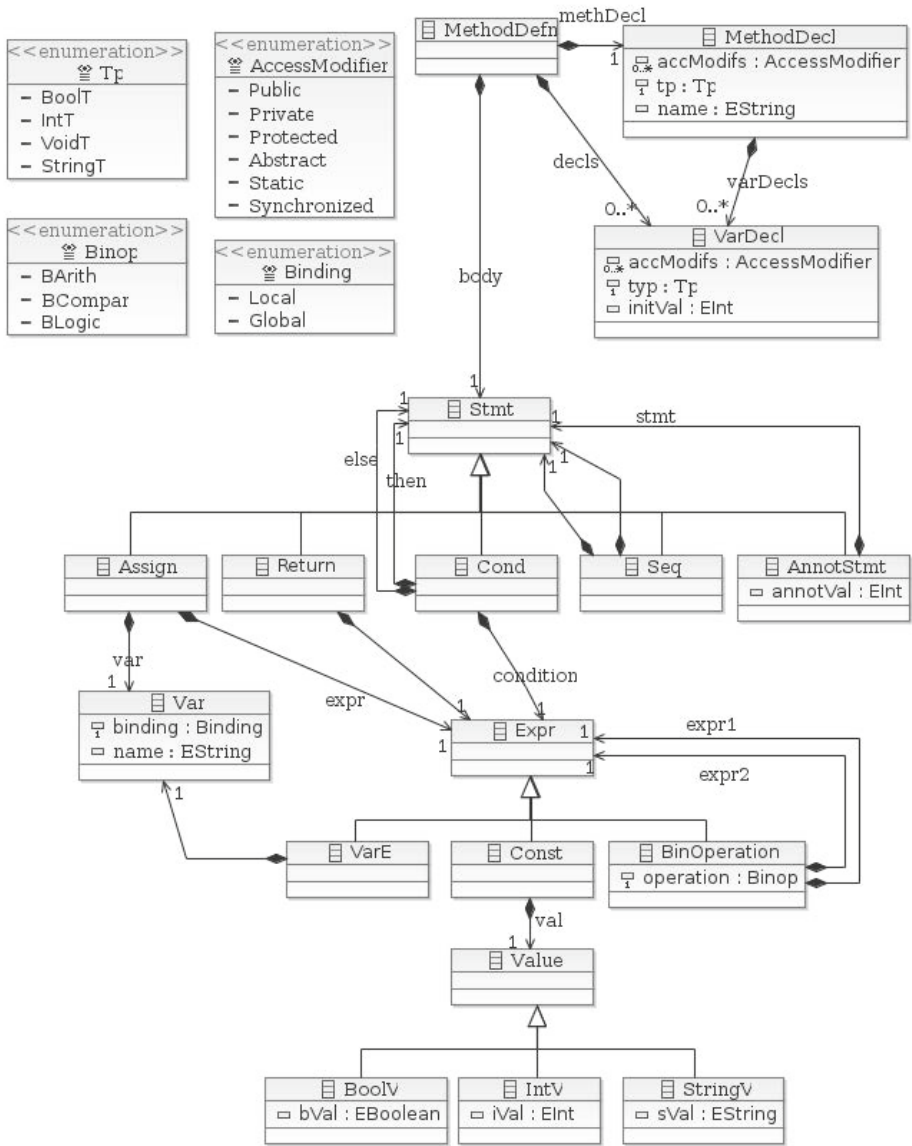


Fig. 8. Resulting Ecore Diagram after Transformation

## 4 Related Work

EMF models are comparable to Unified Modeling Language Class diagrams. For this reason, we are interested in the mappings from other formal languages to UML Class diagrams. Some research is dedicated to establishing the link between these two formalisms. We cite the work of *Idani & al.* that consists of a generic transformation of UML models to B constructs [14] and vice-versa [13]. The authors propose a metamodel-based transformation method based on defining a set of structural and semantic mappings from UML to B (a formal method that allows to construct a program by successive refinement, using abstract specifications).

Similarly, there is an MDE based transformation approach for generating Alloy (a textual modeling language based on first order logic) specifications from UML class diagrams and backwards [2, 23].

*Delahaye & al.* describe in [7] a formal and sound framework for transforming Focal specification into UML models.

These methods enable to generate UML components from a formal description but their formal representation is significantly different from our needs: functional data structures.

Also, graph transformation tools [10, 16] permit to define source and target metamodels all along with a set of transformation rules and use graphical representations of instance models to ease the transformation process. However, the verification functionality they offer is often limited to syntactic aspects (such as confluence of transformation rules) and does not allow to model deeper semantic properties (such as an operational semantics of a programming language and proofs by bisimulation).

Our approach combines the two views by offering the possibility to define the abstract syntax of a DSL, to run some verifications on the top of it and to generate the corresponding metamodel to graphically document the formal developments. Furthermore, this metamodel can be used to easily generate a textual editor using Xtext facilities.

## 5 Conclusion

Our work constitutes a first step towards a combination of interactive proof and Model Driven Engineering. We have presented a generic method based on MDE for transforming data type definitions used in proof assistants to class diagrams.

The approach is illustrated with the help of a Domain Specific Language developed by ourselves. It is a Java-like language enriched with annotations. Starting from data type definitions, set up for the semantic modeling of the DSL, we have been able to generate an EMF meta-model. In addition to its benefits for documenting and visualizing the DSL, it is manipulated in the Eclipse workbench to generate a textual editor as an Eclipse plug-in.

Currently, we are working on extending the subset of data type definitions by adding a way to transform parameterized types to generic types in Ecore,

and coupling our work with the generation of provably correct object oriented code from proof assistants. Moreover, we intend to work on the opposite side of the transformation, namely the possibility to generate data structure definitions from class diagrams.

## References

1. Alanen, M., Porres, I.: A relation between context-free grammars and meta object facility metamodels. Tech. rep., Turku Centre for Computer Science (TUCS) (March 2003), <http://www.cis.uab.edu/courses/cs593/spring2010/TR606.pdf>
2. Anastasakis, K., Bordbar, B., Georg, G., Ray, I.: UML2Alloy: A Challenging Model Transformation. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, pp. 436–450. Springer, Heidelberg (2007)
3. Baklanova, N., Strecker, M., Féraud, L.: Resource Sharing Conflicts Checking in Multithreaded Java Programs. In: Informal Proceedings FAC 2012 (April 2012)
4. Bézivin, J.: Model Driven Engineering: An Emerging Technical Space. In: Lämmel, R., Saraiva, J., Visser, J. (eds.) GTTSE 2005. LNCS, vol. 4143, pp. 36–64. Springer, Heidelberg (2006), <https://www.uni-koblenz.de/~laemmel/gttse/2005/pdfs/41430036.pdf>
5. Budinsky, F., Brodsky, S.A., Merks, E.: Eclipse Modeling Framework. Pearson Education (2003)
6. Coq Development Team: The Coq proof assistant reference manual. version 8.31 (2010), <http://coq.inria.fr/refman/>, <http://coq.inria.fr/refman/>
7. Delahaye, D., Étienne, J.F., Viguié Donzeau-Gouge, V.: A Formal and Sound Transformation from Focal to UML: An Application to Airport Security Regulations. In: UML and Formal Methods (UML&FM), vol. 4, pp. 267–274 (2008), <http://cedric.cnam.fr/~delahaye/?page=publis>
8. van Deursen, A., Klint, P., Visser, J.: Domain-specific languages: An annotated bibliography. SIGPLAN Notices 35(6), 26–36 (2000)
9. Eclipse Community: Tutorials and documentation for Xtext 2.0 (2011), <http://www.eclipse.org/Xtext/documentation/>
10. Ehrig, K., Ermel, C., Hänsgen, S., Taentzer, G.: Generation of visual editors as Eclipse plugins. In: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, ASE 2005, pp. 134–143. ACM, New York (2005), <http://doi.acm.org/10.1145/1101908.1101930>
11. France, R.B., Evans, A., Lano, K., Rumpe, B.: The UML as a formal modeling notation. Computer Standards & Interfaces 19(7), 325–334 (1998)
12. Gronback, R.C.: Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley, Upper Saddle River (2009)
13. Idani, A.: UML Models Engineering from Static and Dynamic Aspects of Formal Specifications. In: Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Soffer, P., Ukor, R. (eds.) BPMDS 2009 and EMMSAD 2009. LNBP, vol. 29, pp. 237–250. Springer, Heidelberg (2009)
14. Idani, A., Boulanger, J.L., Philippe, L.: A generic process and its tool support towards combining UML and B for safety critical systems. In: Hu, G. (ed.) CAINE, pp. 185–192. ISCA (2007)
15. Kleppe, A.G., Warmer, J., Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing Co., Inc., Boston (2003)



16. de Lara, J., Vangheluwe, H.: Using AToM<sup>3</sup> as a meta-case tool. In: Proceedings of the 4th International Conference on Enterprise Information Systems (ICEIS), Ciudad Real, Spain, pp. 642–649 (April 2002), <http://www.cs.mcgill.ca/~hv/publications/02.ICEIS.MCASE.pdf>
17. Leroy, X., Doligez, D., Frisch, A., Garrigue, J., Rémy, D., Vouillon, J.: The OCaml system release 3.12. documentation and user's manual (July 2011), <http://caml.inria.fr/pub/docs/manual-ocaml/index.html>
18. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL. LNCS, vol. 2283. Springer, Heidelberg (2002), <http://isabelle.in.tum.de>
19. Odersky, M., Altherr, P., Cremet, V., Emir, B., Maneth, S., Micheloud, S., Mihaylov, N., Schinz, M., Stenman, E., Zenger, M.: An Overview of the Scala Programming Language. Tech. Rep. IC/2004/64, EPFL Lausanne, Switzerland (2007)
20. OMG: Meta Object Facility (MOF) Core v. 2.0 Document (2006), <http://www.omg.org>
21. Peyton-Jones, S.: Haskell 98 language and libraries: the revised report. Cambridge University Press, Cambridge (2003), <http://www.worldcat.org/isbn/9780521826143>
22. Selic, B.: The pragmatics of model-driven development. *IEEE Software* 20(5), 19–25 (2003)
23. Shah, S.M.A., Anastasakis, K., Bordbar, B.: From UML to Alloy and Back Again. In: Ghosh, S. (ed.) MODELS 2009. LNCS, vol. 6002, pp. 158–171. Springer, Heidelberg (2010)
24. Steele, G.L.: Common LISP, 2nd edn. Digital Press (1990)
25. Wimmer, M., Kramler, G.: Bridging Grammarware and Modelware. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 159–168. Springer, Heidelberg (2006)

# About One Efficient Algorithm for Reachability Checking in Modeling and Its Implementation

Alexander Letichevsky<sup>1</sup>, Olexander Letychevskiy<sup>1</sup>, and Vladimir Peschanenko<sup>2</sup>

<sup>1</sup> Glushkov Institute of Cybernetics of NAS of Ukraine, 40 Glushkova ave., Kyiv, Ukraine  
let@cyfra.net, lit@iss.org.ua

<sup>2</sup> Kherson State University, 27, 40 Rokiv Zhovtnya str., Kherson, Ukraine  
vladimirius@gmail.com

**Abstract.** The problem of reachability of states of transition systems is considered hereby. The notions of partial unfolding and permutability of two operators (including the notion of statically permutable operators) are presented. New algorithm for reachability problem in terms of insertion modeling, and implementation of this algorithm are described. An example of application of the proposed algorithm is considered.

**Keywords:** insertion modeling, reachability, verification, interleaving.

## 1 Introduction

Verification of models of multiagent distributed systems and models of parallel computation usually involves symbolic modeling [1] and is performed on a high level of abstraction. These models are highly nondeterministic because of their symbolic nature and usage of parallel composition, which adds a new level of non-determinism by interleaving (nondeterministic choice of actions order in parallel composition) [2].

The main problem of verification is the problem of combinatorial explosion of the number states of the model. A state in model checking includes a lot of attributes and processes. The total number of states could be very large even if the number of processes is finite and all of the attributes have finite number of possible values. Main source of combinatorial explosion of states are the number of processes, interleaving between them and nondeterministic behavior of them. Usually the systems are parallel and the number of their states grows exponentially with the number of processes. Obviously, model checking by naive enumeration of states is not feasible. Many different technologies are devoted to solve this problem: methods that introduce partial order approach to reduce interleaving [4], methods for determining the symmetry when verifying the equivalence of states [5], techniques of abstraction [6], approximation [7], symbolic modeling, etc. These standard model checking algorithms work only when the set of states that are reachable from the given initial state is finite.

Insertion modeling [15] allows to perform symbolic modeling with infinite number of states. There are various model checking techniques for infinite-state systems, but they are less developed than finite-state techniques and tend to place stronger

limitations on the kind of systems and/or the properties that can be model checked. One of such techniques are presented in [9, 10]. In Petri net theory there is a well-known McMillan's algorithm of unfolding [11] that in many cases helps to exponentially decrease interleaving in system analysis. The book [12] generalizes this technique to finite automata networks. This chapter presents a new algorithm for reachability problem in terms of insertion modeling [13, 14, 15] for models with infinite number of states. The algorithm combines the ideas of economic unfolding of McMillan with online reachability checking using some general assumptions about information dependencies of actions expressed in terms of permutability [14].

The chapter is devoted to the solution of the problem of interleaving reduction in insertion models with infinite number of states.

The algebra of behaviors is presented in section Behavior Algebras, the verification environments, corresponding insertion function, and predicate transformer are considered in section Verification Environments. The normal form of behavior is defined in section Behaviors Over Basis  $B$ . The problem of reachability of the states is described in section Verification. The notion of partial unfolding is examined in section Partial Unfolding. The optimization of partial unfolding by statically permutable operators is reviewed in section Static Permutability Property. The algorithm for reducing of interleaving for transition systems is presented in section Algorithm of Interleaving Reduction. The implementation of this algorithm is considered in section Algorithm Implementation. The example which demonstrates a good result of using the partial unfolding algorithm is presented in section Examples of Application.

## 2 Behavior Algebras

Behavior algebra [14] is a kind of process algebra; it is used to express the behavior of agents (transition systems) considered up to bisimilarity or trace equivalence. To make economic unfolding we need to distinguish sequential and parallel behaviors. So we consider the following modification of the notion of behavior algebra. It is a multisorted algebra with three components: the algebra of *actions*, the algebra of *sequential behaviors*, and the algebra of *parallel behaviors*.

The algebra of sequential behaviors has operations of prefixing:  $\langle action \rangle . \langle sequential\ behavior \rangle$  and one internal operation of nondeterministic choice  $(\langle \rangle + \langle \rangle)$ , which is associative, commutative, and idempotent operation with neutral element  $0$ . We also consider the constant behavior  $\Delta$  (successful termination), which is the common element of the algebra of sequential and the algebra of parallel behaviors. The operations of action algebra will be considered later.

The algebra of parallel behaviors has the parallel composition  $(\langle \rangle \parallel \langle \rangle)$  of sequential behaviors as the main binary operation. It is associative commutative (but is not idempotent) and has the neutral element  $\Delta$ . It also has the prefixing operation and nondeterministic choice. The algebra of sequential behaviors is implicitly included to the algebra of parallel behaviors by the identity  $u = u \parallel \Delta$  (parallel composition with one component). Unfolding of parallel composition by interleaving will be considered only after inserting of agents that are formed by parallel composition, into the environment.

### 3 Verification Environments

Verification environments of the form  $E = E(U, P, B)$  are defined by the following parameters: the set of *conditional expressions*  $U$ , the set of *operators*  $P$ , and the set of *basic behaviors*  $B$ . The set of actions is defined as a union of the set of conditions and the set of operators. The set of basic behaviors is used to define the behaviors of agents inserted into environment in the way which will be explained later. We also suppose that some logic language (first order or temporal) called *basic language* is fixed to define the states of environment and checking conditions for verification. The conditional expressions also belong to this language.

The state of environment is represented as  $E[u]$ , where  $E$  is a statement of basic language and  $u$  is a parallel composition of sequential behaviors of agents inserted into environment. We suppose that operators are divided into the set of conditional and unconditional operators. Conditional operator has the form  $\alpha \rightarrow a$  where  $\alpha$  is a condition and  $a$  is an unconditional operator. Unconditional operator  $a$  is identified with conditional operator  $1 \rightarrow a$ . The associative product  $(*)$  and the function  $pt: U \times P \rightarrow U$  (predicate transformer) are defined by the set of actions so that the following identities are valid:

$$\begin{aligned} pt(\alpha, \beta \rightarrow a) &= pt(\alpha \wedge \beta \rightarrow a) \\ pt(pt(\alpha, a), b) &= pt(\alpha, a * b) \\ (\alpha \rightarrow a) * (\beta \rightarrow b) &= pt(pt(\alpha, a) \wedge \beta, b) \\ \alpha * \beta &= \alpha \wedge \beta \end{aligned}$$

Here  $\alpha$  and  $\beta$  are conditions,  $a$  and  $b$  are unconditional operators.

Predicate transformer is supposed to be monotonic:  $\alpha \rightarrow \beta \Rightarrow pt(\alpha, a) \rightarrow pt(\beta, a)$ . In general case, the  $pt$  function is defined by some concrete syntax. An example of such pair (*syntax*,  $pt$ ) can be found in [16].

**Example.** The basic language is a first order language. Conditions are formulae over simple attributes - symbols that change their values when a system changes its state. Formally they are considered as function symbols with arity 0. Unconditional operators are assignments (parallel assignments, sequences of assignments, if-then-else operators, loops with finite number of repetitions, etc.). As usually in this case,

$$pt(\alpha(x), (x_1 := t_1(x), x_2 := t_2(x), \dots)) = \exists z(\alpha(z) \wedge (x_1 = t_1(z) \wedge x_2 = t_2(z) \wedge \dots))$$

Actually this is the strongest postcondition for precondition  $\alpha$ .

**Insertion function** is defined by the following identities and rules.

1.  $E[u, v] = E[u \parallel v]$ ,  $u, v$  are agents with sequential behavior (see sec. 1).

#### Identities for conditions

2.  $E[\alpha.u + v] = E[v]$ , if  $(E \wedge \alpha) = 0$ .

3.  $E[\alpha.\beta.u + v] = E[\alpha \wedge \beta.u + v]$ , if  $(E \wedge \alpha) \neq 0$  (merging conditions).

4.  $E[\alpha.\beta \rightarrow a.u + v] = E[\alpha \wedge \beta \rightarrow a.u + v]$ , if  $(E \wedge \alpha \wedge \beta) \neq 0$ . Special cases of these identities are obtained when  $v=0$  or  $\beta=1$ .

5.  $E[\alpha.\varepsilon] = E \wedge \alpha[\varepsilon]$ , if  $(E \wedge \alpha) = 0$ .

**Identities for operators**

6.  $E[au + v] = E[v]$ , if  $pt(E, a) = 0$ .

7.  $E[au] = a.pt(E, a)[u \parallel \varphi(a, E)]$ , if  $pt(E, a) \neq 0$ ,  $\varphi(a, E)$  is a parallel composition of sequential behaviors (it generates some new parallel branches). If  $\varphi(a, E) = \Delta$ , then  $u \parallel \varphi(a, E) = u \parallel \Delta = u$  and  $u$  remains unchanged.

**Nondeterministic choice**

8.  $E[au + av + w] = E[a.(u + v) = w]$ . The use of left distributivity means that environment considers behavior expressions up to trace equivalence. It also means that a system uses delayed (angelic) choice.

9.  $E[u + \Delta] = E[u] + E[\Delta]$ . The states  $E[0]$  and  $E[\Delta]$  are called *terminal states of the environment*. Formally, the states of the form  $E[0]$  are equivalent to 0, and states of the form  $E[\Delta]$  are equivalent to  $\Delta$  (if  $E[\Delta] = E[\Delta] + \Delta$  is added). But from the point of view of verification, it is useful to distinguish syntactically different terminal states.

**Parallel behaviors**

10.  $E[u] = E[v] \Rightarrow E[u \parallel w] = E[v \parallel w]$ . Therefore all identities for conditions and operators can be applied within the parallel composition. A component  $a_1.u_1 + \dots + a_n.u_n$  of parallel composition is called *degenerated relative to the state E*, if for all operators  $a_i.pt(E, a_i) = 0$  and for all conditions  $\alpha_i$  it is true that  $(E \wedge \alpha_i) = 0$ . Each component that is degenerated relatively to the state  $E$  is equivalent to 0 relatively to this state.

11.  $E[u] + F[v] = F[v]$ , if parallel composition  $u$  contains degenerated component relative to  $E$ . So all states of environment with degenerated components are equivalent to 0.

12.  $E[u + \Delta \parallel v] = E[u \parallel v] + E[v]$ .

13.  $E[a_1.u_1 + a_2.u_2 + \dots] = E[a_1.u_1 \parallel v] + E[a_2.u_2 \parallel v] + \dots$ , if all actions  $a_i$  are different, if  $a_i$  is a condition then  $u_i$  is terminal constant, and  $v$  does not contain components degenerated relatively to the state  $E$ . The state of environment  $E[u]$  is called *dead lock state*, if there are no transitions from this state, but  $u$  is not a successful termination. If there is at least one degenerated component in parallel composition, then the corresponding state is a dead lock state. All dead lock states are equivalent to 0, but it is useful to distinguish them as well as terminal constants. The rules (9), (12), and (13) are called *unfolding of nondeterministic choice*.

14.  $E[a_1.u_1 \parallel \dots \parallel a_n.u_n] = \sum_{i=1}^n a_i.( \dots \parallel a_{i-1}.u_{i-1} \parallel u_i \parallel a_{i+1}.u_{i+1} \parallel \dots )$ , if all components of parallel composition are non-degenerated. This relation is called a *full unfolding algorithm for a parallel composition*. This is a complete unfolding and the main result of this chapter shows that it is not needed to make the complete unfolding at each step of verification. Let  $u = a_1.u_1 \parallel \dots \parallel a_n.u_n$ , and  $unfold(u, i) = a_i.( \dots \parallel a_{i-1}.u_{i-1} \parallel u_i \parallel a_{i+1}.u_{i+1} \parallel \dots )$ . Then identity (14) can be rewritten as

14a.  $E[a_1.u_1 \parallel \dots \parallel a_n.u_n] = \sum_{i=1}^n unfold(u, i)$ .

Environment does not distinguish trace equivalent behaviors and consequently, bisimilar states of environment are trace equivalent [14]. The identity (14) defines the main transition rule for the system:

$$E[u] \xrightarrow{a_i} E'[u'],$$

if  $u$  is a parallel composition with non-degenerated components and  $E'[u']$  is defined by the identity (7).

## 4 Behaviors over Basis B

The set of symbols is given for the set  $B$  of behavior basis. These symbols are called *basic sequential behaviors*. The expression of the algebra of sequential behaviors constructed from these symbols and termination constants are called *sequential behavior over basis B*. Suppose that for each symbol  $v \in B$  an equation of the form  $v = F_v(v_1, v_2, \dots)$  is given with sequential behavior over basis  $B$  as a right hand side. This equation is called the *definition of a basic behavior v*. The application of this definition as rewriting rule is called the *unfolding of behavior v*. System of basic behaviors is called non-degenerated if each path in the tree representation of the expression  $F_v(v_1, v_2, \dots)$  contains at least one operator.

**Normal form of sequential behavior** is an expression of the form  $a_1.u_1 + a_2.u_2 + \dots + a_n.u_n + \varepsilon$  where  $u_1, u_2, \dots$  are sequential behaviors. If  $a_i$  is a condition, then  $u_i$  is a termination constant,  $n \geq 0$ , and all actions are different (not equivalent with respect to the environment  $E$ ), because of delayed (angelic) choice (see sec. 2).

It can be easily proved that each sequential behavior  $u$  over non-degenerated basis in a state  $E[u]$  can be reduced to a normal form  $v$  equivalent to  $u$  with respect to  $E$ .

*Parallel behavior over B* is a parallel composition of sequential behaviors over  $B$ .

**Normal form of parallel behavior** is a nondeterministic sum of behaviors of the form  $a_1.u_1 + a_2.u_2 + \dots$ , where  $u_1, u_2, \dots$  are sequential behaviors over  $B$ ,  $a_1, a_2, \dots$  are operators or conditions such that if  $a_i$  is a condition, then  $u_i$  is a termination constant.

**Normal form of environment state** is a term of the form  $\sum_{i \in I} a_i.E_i[u_i] + \sum_{j \in J} \Delta$  or 0. *Each environment state with non-degenerated system of basic behaviors is trace equivalent to some normal form.*

## 5 Verification

A property  $\xi$  of environment state is said to be *correct* if it does not distinguish equivalent states. A property  $\xi$  of environment state is *monotonic* if  $E \rightarrow E' \Rightarrow \xi(E[u]) \rightarrow \xi(E'[u])$ . A property  $\xi$  is *reachable* from the given state of a

model  $M$  if there exist a trace that starts in this state and finishes in the state which satisfies property  $\xi$ .

Let  $\Xi$  be the set of correct and monotonic properties, defined on the set of environment states of a model  $M$  (checked properties). The *verification* of  $M$  is to check which properties from  $\Xi$  are reachable (not reachable) from the initial states of  $M$  for a finite number of steps or a number of steps bounded by some constant.

Usually models are highly non-deterministic. This non-determinism is caused by nondeterminism of a model which increases by interleaving of parallel processes:  $a.u \parallel b.v = a.(u \parallel b.v) + b.(a.u \parallel v)$ . So the reducing interleaving is an important problem of efficient verification. It is supposed that the set of properties to be checked contains the property of a state “to be a dead lock” and a property “to be a state of successful termination”.

The simplest verification algorithm is exhaustive unfolding of initial states up to saturation or depletion of a given number of steps. It uses the following formula of unfolding:  $\sum_{i=1}^n E[\text{unfold}(u,i)]$ . The properties to be checked are checked in the process of unfolding and the states that satisfy checked properties are collected. A more economic unfolding algorithm reducing interleaving can be constructed using the following notion of *partial unfolding*.

## 6 Partial Unfolding

Two operators  $a$  and  $a'$  are called *permutable with respect to a state  $E$*  (denoted  $a \xleftarrow{E} a'$ ), if  $E[a * a'] = E[a' * a]$ . Let the state of environment  $E[u] = E[a_1.u_1 \parallel \dots \parallel a_n.u_n]$  be given. Select a component  $s = a_i.u_i$  and construct for this component the set  $\text{nonp}(E,s)$  of those components  $a_j.u_j, i \neq j$  such that  $a_i$  and  $a_j$  are not permutable with respect to the current state  $E$ . Obtain

$$\begin{aligned} \text{punfold}(E,u,i) &= A(i) + B(E,i) + C(E,i) \\ A(i) &= a_i.( \dots \parallel a_{i-1}.u_{i-1} \parallel u_i \parallel a_{i+1}.u_{i+1} \parallel \dots) \\ B(E,i) &= \sum_{i \neq j \wedge (a_i, a_j) \in \text{nonp}(E,s)} a_j.( \dots \parallel a_{j-1}.u_{j-1} \parallel u_j \parallel a_{j+1}.u_{j+1} \parallel \dots) \\ C(E,i) &= \sum_{k \neq i \wedge (a_k, a_i) \notin \text{nonp}(E,s) \wedge a_k \xleftarrow{E} a_w} a_k.( \dots \parallel a_{k-1}.u_{k-1} \parallel ((p; a_w); u'_k) \parallel a_{k+1}.u_{k+1} \parallel \dots) \end{aligned}$$

In the last formula  $((p; a_w); u'_k) = u_k$ , and  $p$  is a sequential composition of actions. The function *punfold* is called a *partial unfolding for parallel composition*. Let us consider the following algorithm of checking reachability. Check needed properties on the current state and the states reachable on one step from the current. Proceed using partial unfolding of the current state. This algorithm is called a *partial unfolding algorithm* (algorithm *punfold*) of reachability checking.

In general case, the algorithm *punfold* uses dynamic permutability of operators, but it is not optimal because *pt* function is used four times for each pair of the operators. Algorithm *punfold* can be improved by using the notion of *static permutability* property of operators.

## 6.1 Static Permutability Property

**Theorem 1.** If two operators  $p = \alpha \rightarrow a, q = \beta \rightarrow b$  are permutable with respect to the states  $E_1 = \alpha \wedge \beta, E_2 = \neg\alpha \wedge \beta, E_3 = \alpha \wedge \neg\beta$  then they are permutable for all states.

Assume the contrary that  $\exists e(e[p^*q] \neq e[q^*p])$  and  $E_1[p^*q] = E_1[q^*p], E_2[p^*q] = E_2[q^*p], E_3[p^*q] = E_3[q^*p]$ . Consider  $E_1[p^*q] = E_1[q^*p]$ :

$$\begin{aligned}
 & (\alpha \wedge \beta[p^*q] \Rightarrow pt(\alpha \wedge \beta \wedge \alpha, a)[p] = pt(\beta \wedge pt(\alpha \wedge \beta, a), b)) \wedge \\
 & \wedge (\alpha \wedge \beta[q^*p] \Rightarrow pt(\alpha \wedge \beta \wedge \beta, b)[q] = pt(\alpha \wedge pt(\alpha \wedge \beta, b), a)) \wedge \\
 & \wedge (\alpha \wedge \beta[p^*q] = \alpha \wedge \beta[q^*p]) \Rightarrow \\
 & \Rightarrow pt(\beta \wedge pt(\alpha \wedge \beta, a), b) = pt(\alpha \wedge pt(\alpha \wedge \beta, b), a) \Rightarrow \\
 & \Rightarrow pt(\beta \wedge pt(\alpha \wedge \beta \wedge (e \vee \neg e), a), b) = pt(\alpha \wedge pt(\alpha \wedge \beta \wedge (e \vee \neg e), b), a) \Rightarrow \\
 & \Rightarrow pt(\beta \wedge pt(\alpha \wedge \beta \wedge e), a, b) \vee pt(\beta \wedge pt(\alpha \wedge \beta \wedge \neg e), a, b) = \\
 & = pt(\alpha \wedge pt(\alpha \wedge \beta \wedge e, b), a) \vee pt(\alpha \wedge pt(\alpha \wedge \beta \wedge \neg e, b), a)
 \end{aligned}$$

Consider  $E_2[p^*q] = E_2[q^*p]$ :

$$\begin{aligned}
 & (\neg\alpha \wedge \beta[p^*q] \Rightarrow pt(\alpha \wedge \neg\alpha \wedge \beta, a)[q] \Rightarrow 0) \wedge \\
 & \wedge (\neg\alpha \wedge \beta[q^*p] \Rightarrow pt(\beta \wedge \neg\alpha \wedge \beta, b)[p] \Rightarrow pt(\alpha \wedge pt(\neg\alpha \wedge \beta, b), a)) \wedge \\
 & \wedge (\neg\alpha \wedge \beta[p^*q] = \neg\alpha \wedge \beta[q^*p]) \Rightarrow \\
 & \Rightarrow (pt(\alpha \wedge pt(\neg\alpha \wedge \beta, b), a) = 0)
 \end{aligned}$$

Consider  $E_3[p^*q] = E_3[q^*p]$ :

$$\begin{aligned}
 & (\alpha \wedge \neg\beta[q^*p] \Rightarrow pt(\beta \wedge \alpha \wedge \neg\beta, b)[q] \Rightarrow 0) \wedge \\
 & \wedge (\alpha \wedge \neg\beta[p^*q] \Rightarrow pt(\alpha \wedge \alpha \wedge \neg\beta, b)[q] \Rightarrow pt(\beta \wedge pt(\alpha \wedge \neg\beta, a), b)) \wedge \\
 & \wedge (\alpha \wedge \neg\beta[p^*q] = \alpha \wedge \neg\beta[q^*p]) \Rightarrow \\
 & \Rightarrow (pt(\beta \wedge pt(\alpha \wedge \neg\beta, a), b) = 0)
 \end{aligned}$$

Let us try to prove this theorem by contradiction. Let us consider insertion of two operators  $p, q$ :

$$\begin{aligned}
 e[p^*q] &= e[(\alpha \rightarrow a)^*(\beta \rightarrow b)] \Rightarrow pt(e \wedge \alpha, a)[\beta \rightarrow b] \Rightarrow pt(\beta \wedge pt(e \wedge \alpha, a), b) \\
 e[q^*p] &= e[(\beta \rightarrow b)^*(\alpha \rightarrow a)] \Rightarrow pt(e \wedge \beta, b)[\alpha \rightarrow a] \Rightarrow pt(\alpha \wedge pt(e \wedge \beta, b), a)
 \end{aligned}$$

So,  $\exists e(pt(\beta \wedge pt(e \wedge \alpha, a), b) \neq pt(\alpha \wedge pt(e \wedge \beta, b), a))$ .

$$\begin{aligned}
 & \exists e(pt(\beta \wedge pt(e \wedge \alpha \wedge (\beta \vee \neg\beta), a), b) \neq pt(\alpha \wedge pt(e \wedge \beta \wedge (\alpha \vee \neg\alpha), b), a)) \Rightarrow \\
 & \Rightarrow \exists e(pt(\beta \wedge pt(e \wedge \alpha \wedge \beta \vee e \wedge \alpha \wedge \neg\beta, a), b) \neq \\
 & \neq pt(\alpha \wedge pt(e \wedge \beta \wedge \alpha \vee e \wedge \beta \wedge \neg\alpha, b), a)) \Rightarrow \\
 & \Rightarrow \exists e(pt(\beta \wedge pt(e \wedge \alpha \wedge \beta, a), b) \vee pt(\beta \wedge pt(e \wedge \alpha \wedge \neg\beta, a), b) \neq \\
 & \neq pt(\alpha \wedge pt(e \wedge \alpha \wedge \beta, b), a) \vee pt(\alpha \wedge pt(e \wedge \neg\alpha \wedge \beta, b), a)) \Rightarrow
 \end{aligned}$$



So, using monotonicity property of  $pt$  function obtain

$$\begin{aligned}
e \wedge \neg \alpha \wedge \beta &\rightarrow \neg \alpha \wedge \beta \Rightarrow pt(e \wedge \neg \alpha \wedge \beta, b) \rightarrow pt(\neg \alpha \wedge \beta, b) \Rightarrow \\
&\Rightarrow pt(\alpha \wedge pt(e \wedge \neg \alpha \wedge \beta, b), a) \rightarrow pt(\alpha \wedge pt(\neg \alpha \wedge \beta, b), a) \Rightarrow \\
&\Rightarrow pt(\alpha \wedge pt(e \wedge \neg \alpha \wedge \beta, b), a) \rightarrow 0 \\
e \wedge \alpha \wedge \neg \beta &\rightarrow \alpha \wedge \neg \beta \Rightarrow pt(e \wedge \alpha \wedge \neg \beta, a) \rightarrow pt(\alpha \wedge \neg \beta, a) \Rightarrow \\
&\Rightarrow pt(\beta \wedge pt(e \wedge \alpha \wedge \neg \beta, a), b) \rightarrow pt(\beta \wedge pt(\alpha \wedge \neg \beta, a), b) \Rightarrow \\
&\Rightarrow pt(\beta \wedge pt(e \wedge \alpha \wedge \neg \beta, a), b) \rightarrow 0
\end{aligned}$$

It means that

$$\begin{aligned}
&\exists e(pt(\beta \wedge pt(e \wedge \alpha \wedge \beta, a), b) \neq pt(\alpha \wedge pt(e \wedge \alpha \wedge \beta, b), a)) \wedge \\
&\wedge (E_1[p^* q] = E_1[q^* p]) \Rightarrow \\
&\Rightarrow \exists e((pt(\alpha \wedge pt(e \wedge \alpha \wedge \beta, b), a) = pt(\beta \wedge pt(\neg e \wedge \alpha \wedge \beta, a), b)) \wedge \\
&\wedge (pt(\beta \wedge pt(e \wedge \alpha \wedge \beta, a), b) = pt(\alpha \wedge pt(\neg e \wedge \alpha \wedge \beta, b), a)))
\end{aligned}$$

Let us consider the case when

$$\exists e(pt(\alpha \wedge pt(e \wedge \alpha \wedge \beta, b), a) = pt(\beta \wedge pt(\neg e \wedge \alpha \wedge \beta, a), b)).$$

$e \wedge \alpha \wedge \beta \wedge \neg e \wedge \alpha \wedge \beta = 0$  means that  $pt$  function translates this formulae into one state independently from  $e$  and  $\neg e$ . It is only possible when all attribute expressions from  $e$  are in  $r, s$  list of  $pt$ , and after application of both protocols  $p, q$  no restrictions are left from  $e$  and  $\neg e$ , because of contradiction in other cases. It means that both operators  $p, q$  translate all subformulae, which depend on attribute expressions from  $e$ , into one subformula, independently from sequence of application. So, by obtaining a contradiction, we have  $pt(\alpha \wedge pt(e \wedge \alpha \wedge \beta, b), a) = pt(\beta \wedge pt(e \wedge \alpha, a), b)$ , theorem is proved.

Two operators  $p = \alpha \rightarrow a, q = \beta \rightarrow b$  are called *statically permutable* if they satisfy the following conditions:

1.  $pt(\alpha \wedge pt(\alpha \wedge \beta, b), a) = pt(\beta \wedge pt(\alpha \wedge \beta, a), b)$
2.  $pt(\alpha \wedge pt(\neg \alpha \wedge \beta, b), a) = 0$
3.  $pt(\beta \wedge pt(\alpha \wedge \neg \beta, a), b) = 0$

From the practical point of view, it is a slow process to call  $pt$  function 8 times in order to check the static permutability property for two operators. So, let us define the weak property for static permutability of two operators.

Let  $r(p), s(p), z(p)$  be the lists of predicate transformer for operator  $p = \alpha \rightarrow a$ , where  $r(p)$  is the list of the attribute expressions from left part of assignment of  $a$ ,  $s(p)$  is the list of the attribute expressions from the formulae part of  $a$ ,  $z(p)$  is the list of attribute expressions from  $\alpha$  that are not in  $r(p) \cup s(p)$  [16].

Two operators  $p = \alpha \rightarrow a, q = \beta \rightarrow b$  are called *weakly statically permutable* if

$$\begin{aligned}
&((r(p) \cup s(p)) \cap (r(q) \cup s(q) \cup z(q)) = \emptyset) \wedge \\
&\wedge ((r(q) \cup s(q)) \cap (r(p) \cup s(p) \cup z(p)) = \emptyset)
\end{aligned}$$

**Theorem 2.** If two operators  $p = \alpha \rightarrow a, q = \beta \rightarrow b$  satisfy weak condition of static permutability then they are statically permutable

If  $p, q$  satisfy weak condition of static permutability then both operators work with different memory. It means that all conditions for static permutability are satisfied and the theorem is proved.

## 7 Algorithm of Interleaving Reduction

The algorithm of interleaving reduction is presented in Fig.1.

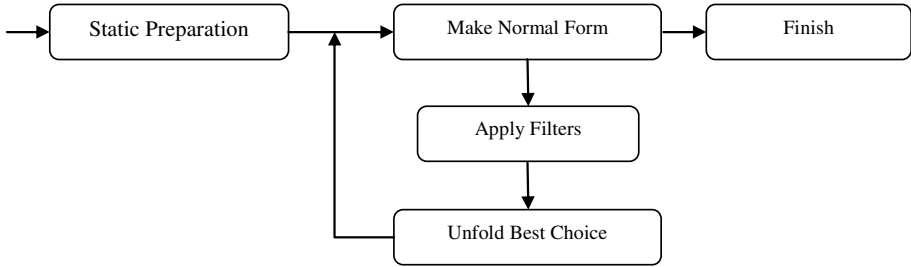


Fig. 1. Algorithm of Interleaving Reduction

Verification environments of the form  $E = E(U, P, B)$  are input for our algorithm. The first component *Static Preparation* finds all statically non-permutable operators from the behavior  $B$ . Component *Make Normal Form* translates behavior into normal form of  $B$  (section 3). Next component *Apply Filters* checks input verification environments with the following filters:

1. *Visited*. If current verification environment was visited previously, then it should be removed from current consideration. If not, then it is added to the storage of visited verification environments.
2. *Terminal States*. If verification environment is one of the terminal states  $E[0]$  (deadlock) or  $E[\Delta]$  (successful termination state), then it should be removed from current consideration and saved to terminal storage.
3. *Goal*. Goal filter is by user defined formula  $\varphi_g$ . If formula  $\varphi_g \wedge E$  is satisfiable, then current verification environment is removed from the consideration.
4. *Safety*. Safety filter is by user defined formula  $\varphi_s$ . If formula  $\neg\varphi_s \wedge E$  is satisfiable, then current verification environment is removed from the consideration.

Of course, the algorithms for detecting visited states and dead locks should be defined for partial unfolding.

The state in the set of search tree is considered as *visited* if it is in the set of already visited states and all its possible successors are in this set:

$$\begin{aligned}
 & E[a_1.(u_1 \parallel a_2.u_2 \parallel \dots \parallel a_i.u_i \parallel \dots \parallel a_n.u_n)] - \textit{visited} \\
 & \dots \\
 & E[a_i.(a_1.u_1 \parallel \dots \parallel a_{i-1}.u_{i-1} \parallel u_i \parallel a_{i+1}.u_{i+1} \parallel \dots \parallel a_n.u_n)] - \textit{visited} \\
 & \dots \\
 & E[a_n.(a_1.u_1 \parallel \dots \parallel a_{n-1}.u_{n-1} \parallel u_n)] - \textit{visited}
 \end{aligned}$$

The state in the set of search tree is considered as *dead lock* if all of these expressions are equal to 0.

Component *Unfold Best Choice* selects verification environment and component of parallel composition which should be unfolded first.

Let's consider the process of choice of verification environment and component for unfolding. Usage of the breadth-first search (BFS) algorithm is better than the depth-first search (DFS) algorithm for checking of reachability property. But, in general case our algorithm could be used with any traversal strategy. So, let us use BFS algorithm and let the chosen component be in normal form  $E[a_1.u_1 \parallel a_2.u_2 \parallel \dots]$ ,  $s_i = a_i.u_i$ . Then component  $j = \min_i \text{nonp}(s_i)$  is chosen for partial unfolding  $\text{punfold}(E, (a_1.u_1 \parallel a_2.u_2 \parallel \dots), j)$ ,  $j \in \{1, 2, \dots\}$ .

In general case the partial unfolding loses states, because  $\text{punfold}(u, i) \neq \text{unfold}(u, i)$ .

Let  $S_1(S_2)$  be sets of all reachable states, which were obtained with function *unfold* (*punfold*), let  $S_1/S_2$  be a set of states, which are called *lost states*. Situation where state  $S_2$  a is reachable from  $S_1$  with the *unfold*, but not reachable with the *punfold* is called "broken reachability property".

**Theorem 3.** The partial unfolding algorithm does not break the reachability property.

The proof is based on the correctness and monotonicity of checked properties, and the following property of partial unfolding algorithm: if  $E[u] \xrightarrow{a} E'[u']$  and the state  $s$  is reachable from  $E'[u']$  then there exists a transition  $E[u] \xrightarrow{b} E''[u'']$  made by partial unfolding algorithm such that  $s$  is also reachable from  $E''[u'']$ . To prove this statement it is sufficient to consider all cases for transitions in partial unfolding algorithm.

If the algorithm of partial unfolding is used for checking reachability property of goal and safety states, then algorithm should check those states after each call of the *punfold*.

Finally, if where are no verification environment left, then component Finish prints statistic of model verification: number of covered states, list of terminal, goal, safety states, time of verification and finishes its work.

## 8 Algorithm Implementation

The described algorithm was implemented in Insertion Modeling System as a component of insertion machine - analytical Model Driver[8], which is intended for model analysis, investigation of its properties etc.

So, the main function of implemented algorithm is *run\_model*. The part of implementation could be presented as the following APLAN [18] (algebraic programming language) code:

```
run_model:=proc(init_states) (
new_states:=0;
generate_static_permutable_table(init_states);
```

```

active_states-->make_active(init_states);
active_states-->process_active(active_states);
loop(
active_states-->check_filters_disj(active_states);
new_states-->mrg(active_states|/new_states);
equ(new_states,0)->return 1;
S-->select_best(new_states);
S-->punfold(S);
S-->process_active(S)
active_states-->S)
);

```

The function *generate\_static\_permutable\_table* creates two lists: list of full statically permutable operators for all of the parallel processes, list of non-statically permutable pairs of two operators from different parallel processes.

The function *make\_active* makes the internal environment presentation using composition of insertions. For example, now we use the following internal normal form:  $obj(nonp:0, dist:1, hist:Nil)[E[u]]$ . Here *nonp* is the number of nonpermutable operators according to environment state *E* (*nonp*=0 for any full statically permutable operator, the function *pt* should be called for checking permutability property for non statically permutable pairs only), *dist* is the depth of this state search, *hist* is the list of insertion functions which appeared in this search branch (this field is used as restriction for depth search), *E[u]* is the initial insertion function.

The function *process\_active* returns the disjunction of insertion functions with the behaviors in the normal form  $(\dots \vee obj(nonp : x, dist : y, hist : z)[E^i[a_1^i, u_1^i \parallel \dots]] \vee \dots)$ . Here  $a_1^i$  is an action which should be inserted in this environments state *E*, *x* is the number of nonpermutable operators in  $a_1^i, u_1^i \parallel \dots$  for  $a_1^i$ , *y* is a number of depth distance, *z* is history.

The *loop* is an operator which could be defined with the help of the following equation:  $loop(x)=while(1,x)$ . The function *check\_filters\_disj* checks filters settings for all new environment states (see sec. 7) and returns all environment states which were not filtered. It replaces each filtered environment state by 0 (neutral element for disjunction). The function *mrg* creates canonical form for a formula, operation */* is a disjunction [17].

The function *select\_best* returns the environment state from the list *new\_states* with which algorithm will work in the next step. The implementation of it depends on problem which we should check. Here a user could try to apply the different types of heuristics (for example, BFS, DFS etc). Current implementation of algorithm returns a state with  $\min_i(x)$ . If there are several states with the same value of  $\min_i(x)$  then it returns a state with  $\min_i(y)$  or first of them.

The function *punfold* makes partial unfolding for selected state and chosen action in behavior. If after partial unfolding the situation appears where all new states are visited, then algorithm chooses one of other components of parallel composition for partial unfolding. If no new components are chosen for partial unfolding then it

returns 0. If not, then it makes the partial unfolding for component of parallel composition which was chosen.

The algorithm finishes its work when there is no state for unfolding is left ( $new\_states=0$ ).

## 9 Examples of Application

Let us discuss the simple example which demonstrates work of the function *run\_model*. Let the environment description have three integer attributes:

```
obj(
  attributes:obj(
    x:int,
    y:int,
    z:int))
```

Initial environment state has the form:

```
obj(formula:1)[
  1->(x:=1)||1->(x:=2)||1->(y:=3)||1->(z:=4)]
```

The function *generate\_static\_permutable\_table* returns two lists:

- $L_1 = ((1 \rightarrow (y := 3)), (1 \rightarrow (z := 4)))$  – list of full permutable operators.
- $L_2 = (1 \rightarrow (x := 1), 1 \rightarrow (x := 2))$  – list of nonstatic permutable pairs.

*make\_active* function makes a template for initial environment state:

```
obj(nonp:0,dist:0,hist:nil)[
  obj(formula:1)[ 1->(x:=1)||1->(x:=2)||1->(y:=3)||1->(z:=4)]]
```

The function *process\_active* checks which component could be unfolded in this step. It is  $1 \rightarrow (y := 3)$  and it returns:

```
obj(nonp:0,dist:0,hist:nil)[
  obj(formula:1)[ 1->(y:=3)||1->(x:=1)||1->(x:=2)||1->(z:=4)]]
```

The next step is to check filters. The initial state is not filtered and system adds the new visited state (current one). *select\_best* returns the current state because it is only one case here. *punfold* makes partial unfolding for first component. Operator  $1 \rightarrow (y := 3)$  is full statically permutable, in that case  $B(E, 1 \rightarrow (y := 3)) = C(E, 1 \rightarrow (y := 3)) = 0$ . So, obtained:

```
obj(nonp:0,dist:1,hist:(1->(y:=3),nil))[
  obj(formula:(y:=3)[ 1->(x:=1)||1->(x:=2)||1->(z:=4)]]
```

After that *process\_active* returns the state with next full statically permutable operator for partial unfolding:

```
obj(nonp:0,dist:1,hist:(1->(y:=3),nil))[
  obj(formula:(y:=3)[ 1->(z:=4)||1->(x:=1)||1->(x:=2)]]
```

Starting the new iteration of loop. The next step is to check filters. The current state is not filtered (add it to the storage of visited states). *select\_best* returns the current state. Next, *punfold* returns

```
obj(nonp:0,dist:2,hist:(I->(z:=4),I->(y:=3),Nil))[
obj(formula:((y=3) ^ ((z=4)))[ I-> (x:=1)||I-> (x:=2)]]
```

The left two operators are nonpermutable statically and dynamically. So, *process\_active* marks that case:

```
obj(nonp:1,dist:2,hist:(I->(z:=4),I->(y:=3),Nil))[
obj(formula:((y=3)&((z=4)))[ I-> (x:=1)||I-> (x:=2)]]
```

Starting the new iteration of loop. This state is not filtered too and only one for *select\_best*. *punfold* returns here nondeterministic states:

```
obj(nonp:0,dist:3,hist:(I->(x:=1),I->(z:=4),I->(y:=3),Nil))[
obj(formula:((y=3)&((z=4)&(x=1)))[I-> (x:=2)]] ∨
obj(nonp:0,dist:3,hist:(I->(x:=2),I->(z:=4),I->(y:=3),Nil))[
obj(formula:((y=3)&((z=4)&(x=2)))[I-> (x:=1)]]
```

*process\_active* leaves this state in the same view because there is no parallel components.

Starting the new iteration of loop. Both states are not filtered. *select\_best* returns first argument. *punfold* returns terminal state:

```
obj(nonp:0,dist:4,hist:( I-> (x:=2),I->(x:=1),I->(z:=4),I->(y:=3),Nil))[
obj(formula:((y=3)&((z=4)&(x=2)))[ Δ ]] (Successful termination state)
process_active leave terminal state in the same view.
```

Starting the new iteration of loop. Successful termination state is filtered as a one of terminal states and function *check\_filters\_disj* returns 0. *select\_best* returns one left state which was created in previous step of loop. *punfold* returns terminal state:

```
obj(nonp:0,dist:4,hist:( I-> (x:=1),I->(x:=2),I->(z:=4),I->(y:=3),Nil))[
obj(formula:((y=3)&((z=4)&(x=1)))[ Δ ]] (Successful termination state)
process_active leave terminal state in the same view.
```

Starting the new iteration of loop. Successful termination state is filtered as one of terminal states and function *check\_filters\_disj* returns 0. There are no states left to consider in the loop, so the loop breaks,

It was a simple example which demonstrates some internal data structures and the work of the algorithm. Let us consider another example to demonstrate efficiency of this algorithm. Let the initial state be  $E[R20 \parallel U312]$ . The behavior  $R20$  and  $U312$  is defined by the following graphs fig. 2, fig. 3 respectively.

The  $*R2$  means all operators except  $R2$ , the grey color in fig. 2 means the special situation when the  $R2$  could be inserted after any of operators except  $R2$  and that after insertion of  $R2$  any protocol could be inserted except  $R2$ . The grey color on fig. 3 is used to mark a subpath for successful termination state (terminal states).

The following results are obtained after analyzing the two lists of operators from behavior  $R20$  and  $U312$ :

1. Behavior  $R20$  has 5 out of 20 operators that are statically permutable for all operators from  $U312$ .
2. Behavior  $U312$  has 31 out of 33 operators that are statically permutable for all operators from  $R20$ .

3. The time for creating the list of statically permutable operators is approx. 1 min (CPU - Intel Core Duo 2.0 MHz, RAM - 2 Gb).
4. The whole state space was covered after approx. 25 min.
5. Total number of covered states is 1102.

Out of memory error was obtained without partial unfolding algorithm after approx. 1 hour of work and approx. 50000 of states were covered. So, it is a good result for this example, because if one of the operators is statically permutable for all of the operators from other parallel processes, then  $punfold(u, i) = A(i)$ , because  $B(i) = C(i) = 0$ .



Fig. 2. Behavior for R20

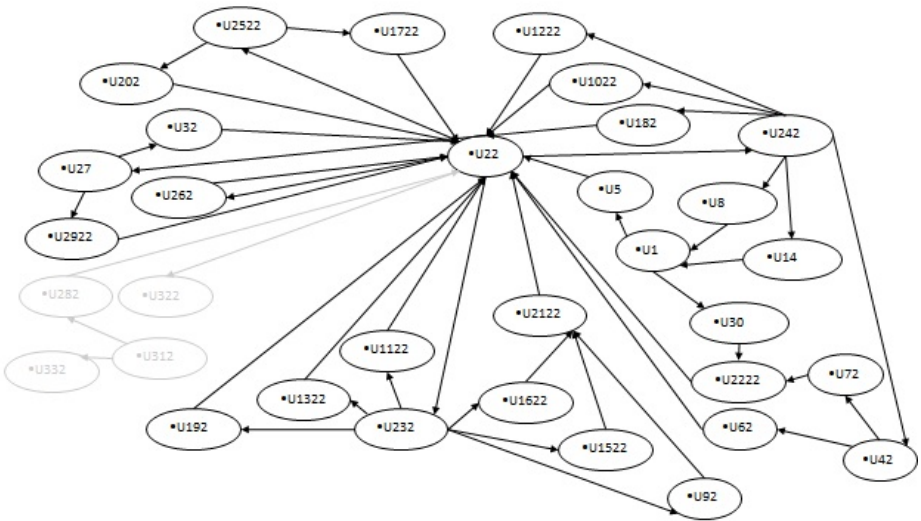


Fig. 3. Behavior for U312

## 10 Conclusions

The verification algorithm based on partial unfolding has been implemented in the system of insertion modeling IMS. It has shown considerable decreasing of the verification time on several examples taken from industrial applications. It is expected that pure C++ version of this algorithm will be at least 10 times faster. It is a well-known case for IMS that C++ implementations of algorithms are generally 10 times faster than their prototypes written in APLAN, which is an interpreted language IMS is written in.

In the near future this algorithm will be applied for the verification of distributed systems in VRS, for verification of parallel programs [8], and for verification of formal specifications in new subject domains using our tool for symbolic modeling with infinite number of states.

## References

1. Symbolic modeling, [http://en.wikipedia.org/wiki/Model\\_checking](http://en.wikipedia.org/wiki/Model_checking)
2. Bergstra, J.A., Klop, J.W.: Process algebra for synchronous communications. *Information and Control* 60, 109–137 (1984)
3. Letichevsky, A., Kapitonova, J., Volkov, V., Letichevsky Jr., A., Baranov, S., Kotlyarov, V., Weigert, T.: System Specification with Basic Protocols. *Cybernetics and System Analysis* 4, 3–21 (2005)
4. Lomuscio, A., Penczek, W., Qu, H.: Partial Order Reductions for Model Checking Temporal-epistemic Logics over Interleaved Multi-agent Systems. *Fundam. Inf.* 101, 71–90 (2010)
5. Ip, C.N., Dill, D.L.: Better verification through symmetry. *Methods Syst. Des.* 9, 41–75 (1996)
6. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. *ACM Trans. Program. Lang. Syst.* 16, 1512–1542 (1994)
7. D'silva, V., Purandare, M., Kroening, D.: Approximation Refinement for Interpolation-Based Model Checking. In: Logozzo, F., Peled, D.A., Zuck, L.D. (eds.) *VMCAI 2008*. LNCS, vol. 4905, pp. 68–82. Springer, Heidelberg (2008)
8. APS & IMS Systems, <http://apsystem.org.ua>
9. Escobar, S., Meseguer, J.: Symbolic Model Checking of Infinite-State Systems Using Narrowing. In: Baader, F. (ed.) *RTA 2007*. LNCS, vol. 4533, pp. 153–168. Springer, Heidelberg (2007)
10. Herbretreau, F., Sutre, G., Tran, T.Q.: Unfolding Concurrent Well-Structured Transition Systems. In: Grumberg, O., Huth, M. (eds.) *TACAS 2007*. LNCS, vol. 4424, pp. 706–720. Springer, Heidelberg (2007)
11. McMillan, K.L.: Trace Theoretic Verification of Asynchronous Circuits Using Unfoldings. In: Wolper, P. (ed.) *CAV 1995*. LNCS, vol. 939, pp. 180–195. Springer, Heidelberg (1995)
12. Esparza, J., Heljanko, K.: *Unfoldings - A Partial-Order Approach to Model Checking*. Springer, Heidelberg (2008)
13. Letichevsky, A., Gilbert, D.: A Model for Interaction of Agents and Environments. In: Bert, D., Choppy, C., Mosses, P.D. (eds.) *WADT 1999*. LNCS, vol. 1827, pp. 311–328. Springer, Heidelberg (2000)



14. Letichevsky, A.: Algebra of behavior transformations and its applications. In: Kudryavtsev, V.B., Rosenberg, I.G. (eds.) *Structural Theory of Automata, Semigroups, and Universal Algebra*. NATO Science Series II. Mathematics, Physics and Chemistry, vol. 207, pp. 241–272. Springer, Heidelberg (2005)
15. Letichevsky, A., Kapitonova, J., Kotlyarov, V., Letichevsky Jr., A., Nikitchenko, N., Volkov, V., Weigert, T.: Insertion modeling in distributed system design. *Problems of Programming* 4, 13–39 (2008)
16. Letichevsky, A.A., Godlevsky, A.B., Letichevsky Jr., A.A., Potienko, S.V., Peschanenko, V.S.: Properties of Predicate Transformer of VRS System. *Cybernetics and System Analyses* 4, 3–16 (2010)
17. Letichevsky, A.A., Letychevskiy, O.A., Peschanenko, V.S.: Insertion Modeling System. In: Clarke, E., Virbitskaite, I., Voronkov, A. (eds.) *PSI 2011*. LNCS, vol. 7162, pp. 262–273. Springer, Heidelberg (2012)
18. Letichevsky, A.A., Kapitonova, J.V.: Algebraic Programming in the APS System. In: *International Symposium on Symbolic and Algebraic Computation*, pp. 68–75. ACM, New York (1990)

# Cross-Diagram UML Design Verification

Iryna Zaretska, Oleksandra Kulankhina, and Hlib Mykhailenko

V.N. Karazin Kharkiv National University, Kharkiv, Ukraine  
zar@univer.kharkov.ua, {mary.cauliflower,tas.nix}@gmail.com

**Abstract.** The chapter presents a general method and software implementation for checking inconsistencies in UML design of a software project. The proposed method uses its own model and first order predicate logic to specify relations between components of the design. Unlike various existing methods the proposed one is focused mostly on cross-diagram inconsistencies and strong adhering to object-oriented principles. The model used in the method is based on the unified graph representation of UML diagrams.

**Keywords:** Software design, object-oriented approach, UML, design model, verification.

## 1 Introduction

Software design has become an increasingly important part of software lifecycle due to the increasing complexity of software under development.

The Unified Modeling Language (UML) [13] is the de facto standard for modeling software systems. The UML supports a wide range of diagrams for modeling software. UML diagrams are independent but connected; their meta-model describes them under a common roof.

Detection of errors at the software design level allows reducing a great number of problems in the late stages of software life cycle. There are several approaches for testing design models but they are mainly dealing with intra-diagram inconsistencies or using scripts for design execution.

In this chapter model faults concerned with cross-diagram inconsistencies are considered. We present a method for detecting cross-diagram inconsistencies in a UML design and its Java implementation. The general model of UML design for verification and refining purposes is introduced and discussed.

## 2 Related Work

This section describes some existing approaches to UML design verification.

### 2.1 OCL Constraints

The most common approach to set the rules of consistency for a UML model is to use Object Constraint Language (OCL) which is supported by the majority of UML CASE tools. In fact a lot of OCL constraints are embedded into the wide spread UML

CASE tools in order to provide inconsistency verification of the model. The fact is that one can freely use OCL on the intra-diagram level but not on the cross-diagram level.

## 2.2 Critic Approach

A number of UML visual design tools provide model verification support including syntax checking and structural and consistency analysis. One of the components of such integrated support tools are critics.

There are various definitions of a design critic or a critic system in the literature. A critic can be considered as an intelligent user interface that evaluates a design made by a user and provides feedback to assist the user to improve the design. Generally critic tools detect potential problems, give advice and alternative solutions, and possibly automated or semi-automated design improvements to the end user.

Design critic tools have been used in design tools for various domains, including software engineering, design sketches, education, etc. Several studies report the benefits of applying design critic tools in software developments activities [1 – 8].

One of the critic tools is ArgoUML (<http://argouml.tigris.org/>), an open source UML CASE tool. This tool supports the editing of UML notation diagrams and detects common errors made by software designers. For example, after placing a class in a class diagram, several critiques are displayed reminding the user that the class requires a better attribute name, needs operations, constructor and associations with other classes, and its class name needs to be capitalized. Thus, the user is helped to improve the design through the critiques. Java API (<http://docs.oracle.com/javase/7/docs/api/>) is used to implement these features. Other critic-based tools like ArchStudio (<http://www.isr.uci.edu/projects/archstudio/>), IDEA (<http://www.jetbrains.com/idea/>) or ABCDE-Critic [5] provide knowledge to architects, designers, and requirement engineers who lack specific understanding of the problem or solution domains. These critic tools all produce critiques that are specific to their problem domain. They use various approaches such as Java API, Prolog rules and knowledge bases, first-order production systems etc. to design and define critiques constraints. These tools have several limitations such as particular code or design language orientation or difficulties in customization requiring comprehension of the critic's domain.

## 2.3 UML Design Execution

A number of approaches have been proposed to execute UML models [9 – 12]. Most of them use UML models to generate high level language code and execute the generated code.

Mellor and Balcer [11] for example use model compilers to support UML model execution. A set of domain and platform-specific model compilers are available commercially for realtime system modeling. At present the compilers cannot be extended to incorporate specific checks as the compiler source is not freely available for modifications.

Another technique for executing UML designs is to execute code that is generated from the model. Assuming that the code and model both contain the same information, executing the code is the same as executing the model.

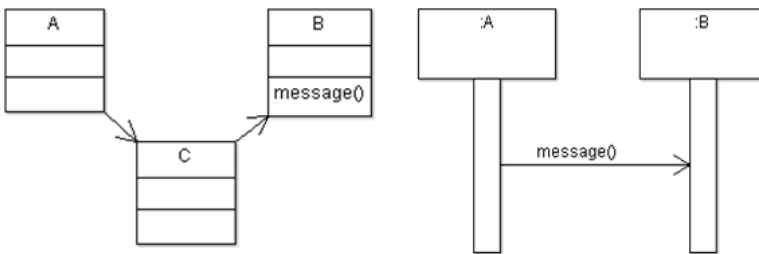
Trung T. Dinh-Trong et al. [12] offer the systematic approach to testing UML designs based on a Java-like action language (JAL) used to transform the UML design under test into the executable form and then exercise them with generated inputs.

### 3 Cross-Diagram Inconsistencies

It is quite important to verify that the information about the model of the system in one UML diagram does not contradict to the information in the other UML diagram. We call such contradictions cross-diagram inconsistencies.

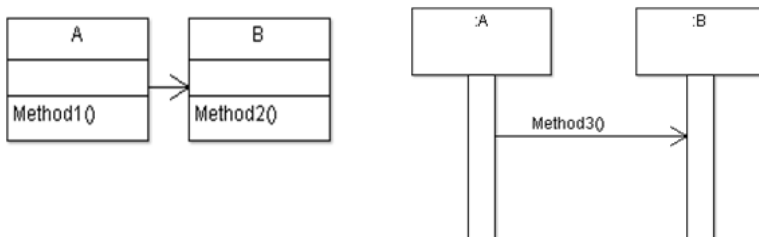
A UML design may contain different cross-diagram inconsistencies. Some of them are listed below.

1. An instance of the class A sends the message to the instance of the class B in the Sequence diagram, but the class B isn't visible for the class A in the Class diagram (Fig. 1).



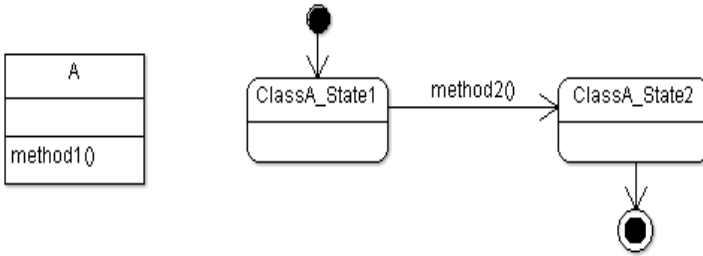
**Fig. 1.** Class B should be visible

2. An instance of the class A sends the message to an instance of the class B in the Sequence diagram, but there is no corresponding method in the class B (Fig. 2).



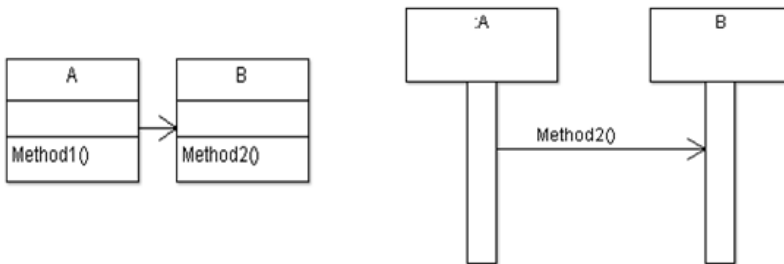
**Fig. 2.** Class B does not have method3()

- Transition from one state of the class A to another one in the State Machine diagram occurs by the class A method invocation, but there is no such method in the class A in the Class diagram (Fig. 3).



**Fig. 3.** Class A does not have method2()

- An instance of the class A sends the message to the class B (but not to the instance of this class) in the Sequence diagram, but the corresponding method of the class B isn't specified as static (Fig.4).



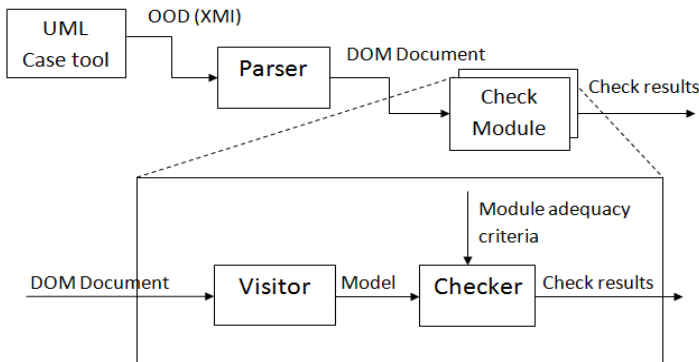
**Fig. 4.** In class B method2() is not specified as static

As our analysis shows the most wide spread UML CASE tools cannot detect such faults in the design and neither critic tools nor the design execution method are of any help in a cross-diagram verification process.

#### 4 Simple Method for Detecting Cross-Diagram Inconsistencies in UML Design

As most of the UML CASE tools allow exporting an object oriented design (OOD) of a target system into XMI format it is natural to use a simple method of cross-diagram verification: parse XMI file and find UML components dependences we are interested

in. Depending on the type of an inconsistency under check we developed different check modules with their own models and consistency rules. The whole process looks like shown in Fig. 5. After parsing the XMI file into the Document Object Model (DOM) the Visitor takes care of getting over its elements and creating instances of the classes from the Checker's model. Then the concrete Checker verifies this model according to its own rules and generates check results. To support this process the Java plug-in was developed. We called it Cross Diagram Inconsistency Check Plug-in (CDICP). It can be easily added to most of the UML CASE tools.



**Fig. 5.** The process of checking UML design

As an example of a Check Module (Fig. 5) we developed Visibility Check Module (VCM) which finds the inconsistencies of the first type (Fig. 1).

VCM uses three classes for UML components representing; we call them Role, AsRole, and SeqRole (Fig. 6). The class Role represents a class in the Class diagram, the class SeqRole represents this class in the Sequence diagram (in fact they are two different UML components), and the class AsRole represents an association or dependency between classes in the Class diagram or a message between classes in the Sequence diagram.

The Visitor in VCM identifies these components in the DOM, creates their instances and places them to the lists of classes, associations, messages, etc. Then the Checker applies its rules, verifies them and generates the result messages. The Checker of VCM for every instance of the class AsRole, which represents some message between classes, checks if there is an association or dependency between these classes (another instance of the class AsRole) and detects its direction. If the connection is not found the Checker forms an error message. This message contains information about the cross-diagram inconsistency specifying its type and names of classes.

The CDICP plug-in for the VCM was developed and incorporated into Eclipse.

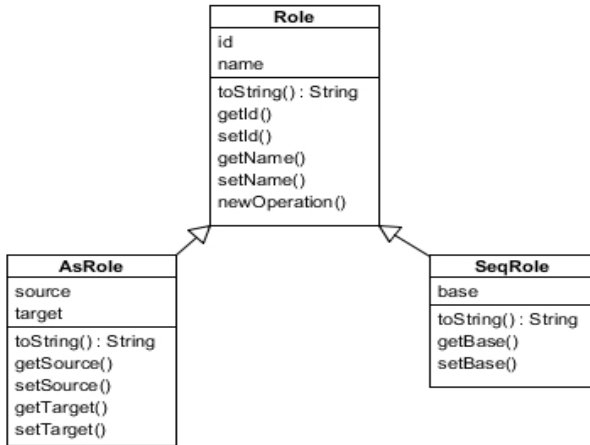


Fig. 6. Classes used by VCM

## 5 General Method for Detecting Cross-Diagram Inconsistencies in UML Design

Analyzing only four diagrams, most important at the design stage, namely Class diagram, Sequence diagram, Object Diagram and State Machine diagram (in UML 2.0 specification) [13, 14] we defined more than 30 intra- and cross-diagram relations to be checked. None of well known CASE tools offers such checks. Using the simple method mentioned above is quite tedious as it supposes developing a special Checker (Fig. 5) for each relation, which in its turn requires multiple searches on XMI file. Even for a middle size project this file is quite big. The main idea here is to develop a special model of the system design for verification purposes (as usually done in verification methods), build it once by parsing XMI file, and then make all checks on this model. Moreover such model can be used for refining design on account of lessening couplings, strengthening cohesion and applying design patterns. All results of this model analysis regardless of the purpose take the form of recommendations so the corrective changes are up to the designer.

Thorough analysis of UML 2.0 specification led us to using graph representation of such model. It allows unified representation of all four diagrams by graphs with different types of vertices and edges. In this case checking relations between UML diagrams is just searching for the definite types of vertices or edges or their interconnections in the model. The first order predicate logic is used to formulate the relations leading to inconsistencies. In fact graph representation simplifies the description of diagrams comparing to their formal specification but is sufficient for verification purposes. For a class diagram the corresponding graph's vertices are classes and edges are connections between them which are association, dependency, generalization and

interface realization. The information about generalization sets is stored separately to simplify search algorithms. For an object diagram the corresponding graph's vertices are objects and edges are links between them. For a sequence diagram the vertices are objects and edges are messages between them. For a state machine (or state chart) diagram the vertices are states and edges are transitions between them. Each type of vertex and each type of edge stores information needed to check intra- and cross-diagram inconsistencies. Say an association of a class diagram as an edge of a graph keeps the name of the association, roles and multiplicities of its participants, etc. An example of the simple class diagram and its graph representation is given in Fig. 7. The edges of the graph represent different types of connections between classes and hence store different information.

Here is the formal representation of our model consisting of graphs of four types for Class, Object, Sequence and State Machine diagrams respectively:

$$D = \{ \{D_{cl}\} \cup \{D_{ob}\} \cup \{D_{seq}\} \cup \{D_{st}\} \} \tag{1}$$

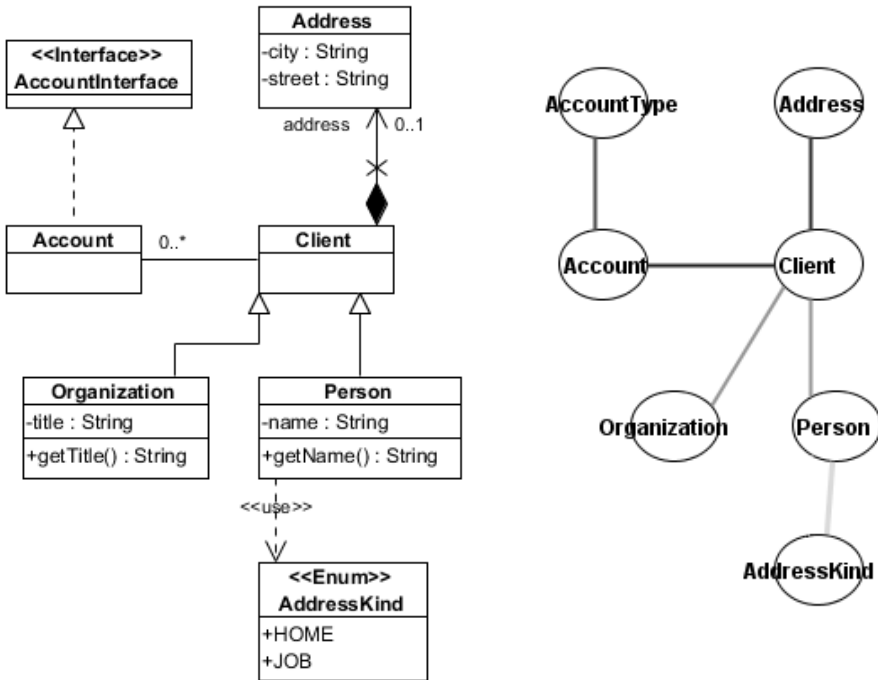


Fig. 7. Example of graph representation of a class diagram

Each of these graphs consists of two sets:  $V$  stands for vertices and  $E$  stands for edges. Their description is given below.<sup>1</sup>

<sup>1</sup> Elements in [] are optional.



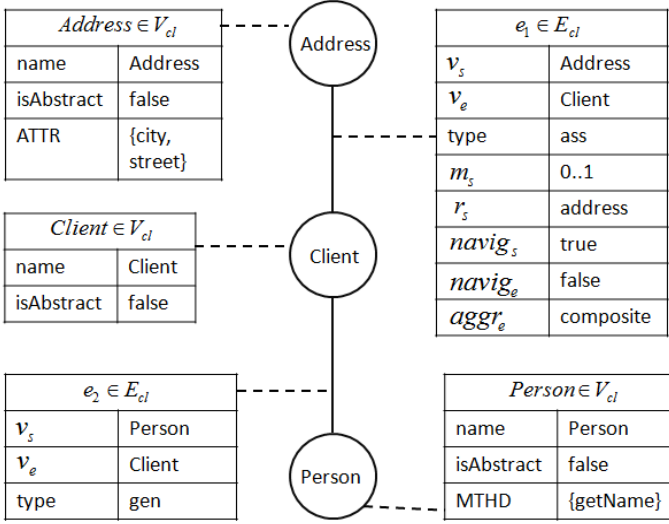
$$\begin{aligned}
D_{cl} &= \{V_{cl}, E_{cl}\} \\
V_{cl} &= \{v : v = (\text{name}, \text{isAbstract}[, \text{ATTR}, \text{MTHD}, \text{STRT}, \text{visibility}])\} \\
\text{ATTR} &= \{\text{attr} : \text{attr} = (\text{name}, \text{domain}, \text{scope}[, \text{visibility}, \text{multiplicity}])\} \\
\text{MTHD} &= \{\text{mthd} : \text{mthd} = (\text{mthdSgn}, \text{scope}, \text{visibility})\} \\
&\quad \text{scope} = \text{instance} \mid \text{classifier} \\
&\quad \text{visibility} = \text{public} \mid \text{private} \mid \text{protected} \mid \text{package} \\
&\quad \text{mthdSgn} = (\text{name}[, \text{PARAMS}, \text{returnDomain}]) \\
\text{PARAMS} &= \{\text{param} : \text{param} = (\text{num}[, \text{name}], \text{domain})\} \\
\text{STRT} &= \{\text{stereotype} : \text{stereotype} = (\text{name})\} \\
E_{cl} &= \{e : e = (v_s, v_e, \text{type}[, \text{info}]); v_s, v_e \in V_{cl}, \text{type} = \text{gen} \mid \text{ass} \mid \text{dep} \mid \text{impl}\} \\
&\quad \text{info} = ([\text{name}, r_s, r_e, m_s, m_e, \text{aggr}_s, \text{aggr}_e, \text{navig}_s, \text{navig}_e]) \\
D_{ob} &= \{V_{ob}, E_{link}\} \\
V_{ob} &= \{v : v = (\text{name}, \text{clName}[, \text{ATTRVAL}, \text{STRT}])\} \\
\text{ATTRVAL} &= \{\text{attrval} : \text{attrval} = (\text{name}, \text{value})\} \\
E_{link} &= \{e : e = (v_s, v_e, \text{name}); v_s, v_e \in V_{ob}\} \\
D_{seq} &= \{V_{cl} \cup V_{ob}, E_{msg}\} \\
E_{msg} &= \{e : e = (v_s, v_e, \text{msgCall}); v_s, v_e \in V_{cl} \cup V_{ob}\} \\
&\quad \text{msgCall} = ([\text{guard}, ]\text{seqnum}, \text{mthdCall}) \\
&\quad \text{mthdCall} = (\text{name}, \text{ARGS}[, \text{returnValue}]) \\
&\quad \text{ARGS} = \{\text{armnt} : \text{armnt} = (\text{num}, \text{value})\} \\
D_{st} &= \{V_{st}, E_{tr}\} \\
V_{st} &= \{v : v = (\text{name}[, \text{entry}, \text{do}, \text{exit}]); \text{entry}, \text{do}, \text{exit} \in \text{mthdCall}\} \\
E_{tr} &= \{e : e = (v_s, v_e, \text{trCall}); v_s, v_e \in V_{st}\} \\
&\quad \text{trCall} = ([\text{guard}, ]\text{mthdCall}).
\end{aligned} \tag{2}$$

An example in Fig. 8 illustrates information stored with some types of vertices and edges.

The relation to be checked should be represented as the first order predicate logic formula. Propositional variables in this formula are the elements of the model above. Such unified approach to formulating the criteria of relations to be checked allows using the only Checker for any sort of relation.

Here is an example of such formula. It describes the fact that if the instance of one class sends the message to the instance of another class in the sequence diagram then the corresponding method should be among the methods of the latter class in the class diagram (Fig. 2 shows an example of this relation not satisfied).

$$\begin{aligned}
&(\forall e \in E_{msg})(\exists v \in \text{implGenPath}(v_e(e))) \\
&(\exists \text{mthd} \in \text{MTHD}(v(e)))(\text{msgCall}(e) \approx \text{mthdSgn}(\text{mthd}))
\end{aligned} \tag{3}$$



**Fig. 8.** Information stored in graph elements for the example in Fig.7

where

$$\begin{aligned}
 implGenPath(v) &= v_1 \dots v_n : \\
 &((v \in V_{cl} \wedge v_1 = v) \vee \\
 &(v \in V_{ob} \wedge (\exists cl \in V_{cl} : clName(v) = name(cl)) \wedge v_1 = cl)) \wedge \\
 &(\forall i = 1, n-1)(\exists e \in E_{cl} : \\
 &(type(e) = gen \vee type(e) = impl) \wedge v_s(e) = v_i \wedge v_e(e) = v_{i+1}
 \end{aligned} \tag{4}$$

is introduced to take into account the fact that the method in question can be inherited along the path in the inheritance tree of the class or be an implementation of an interface method.

The formulas for some relations are quite bulky. Say to check that the destination class of some message in a sequence diagram has the corresponding method with needed visibility modifier in a class diagram one has to use the following formula:

$$\begin{aligned}
 &(\forall e \in E_{msg})(\exists v \in implGenPath(v_e(e))) \\
 &(\exists mthd \in MTHD(v(e)))(msgCall(e) \approx mthdSgn(mthd)) \\
 &(v_e(e) \in V_{ob} \vee scope(mthd) = "classifier") \\
 &((visibility(mthd) = "public") \vee ((visibility(mthd) = "private") \wedge (v_e = v_s)) \vee \\
 &((visibility(mthd) = "protected") \wedge ((v_s(e) \in V_{cl} \wedge v_e(e) \in implGenPath(v_s(e)) \vee \\
 &((v_s(e) \in V_{ob}) \wedge (\exists cl \in V_{cl} : name(cl) = clName(v_s(e))) \wedge (v_e(e) \in implGenPath(cl)))
 \end{aligned} \tag{5}$$

It takes into account the facts that not only the signature of the message call should correspond to the signature of the corresponding method but the visibility modifier of this method can be different depending on the relations between the source and

destination classes of the message. Moreover if the message is sent to a class (not to an instance of a class) then the corresponding method should have static modifier which means that its scope is “classifier”.

## 6 Evaluation Results

The software tool for the proposed method has been developed and tested. As the part of the research we checked inconsistencies in a number of real students' projects of different scope using the developed tool. The results have been compared with experts' evaluation of these projects.

The first project being the biggest one was designed for a university on-line testing system. It allows teachers to create tests including tasks of five types and with different levels of difficulty. A teacher can also define the way each test is estimated. After that a student may pass a test at the time set by a teacher. All students' results are stored by the system and might be viewed by its users. The project's design consists of 4 class diagrams with 95 classes and 15 sequence diagrams with 142 messages in them. Overall number of inconsistencies found in the project is 77.

The second project was designed for a cinema ticket system. It allows cinema cashiers to sell tickets on various sessions. The project's design consists of 2 class diagrams with 22 classes and 4 sequence diagrams with 21 messages in them. Overall number of inconsistencies found in the project is 10.

The third project was designed for an on-line drug store. It allows its clients to order medications via the Internet. The project's design consists of 4 class diagrams with 67 classes and 18 sequence diagrams with 101 messages on them. Overall number of inconsistencies found in the project is 88.

The fourth project was designed for a system supporting a multi-level marketing company selling cosmetics and goods for personal care. The project's design consists of 1 class diagrams with 15 classes and 7 sequence diagrams with 41 messages in them. Overall number of inconsistencies found in the project is 32.

The fifth project was designed for a system of generating reports based on the information taken from a database. A report might include a table with data and charts. The project's design consists of 1 class diagrams with 12 classes and 4 sequence diagrams with 21 messages in them. Overall number of inconsistencies found in the project is 13.

The analysis results show that the most common types of inconsistencies found by the developed tool are of three types: a message call for a nonexistent method, sending a message to an instance of nonexistent class and a message call for a method with an unaccepted visibility modifier.

We asked an expert designer to analyze the same projects for inconsistencies and compared their results with the results given above which were obtained by the developed tool. It may seem unnatural but an expert found about eighty percent of inconsistencies in small projects but only about thirty percent in a project including about one hundred messages and more than sixty classes. The results of the experiment are given in the Table 1.

**Table 1.** The results of experiment

Name of the project	Num. of classes	Num. of msgs	Nonexistent method		Nonexistent class		Unaccepted visibility	
			Tool evaluation	Expert evaluation	Tool	Expert	Tool	Expert
Test system	95	142	42	10	12	3	23	5
Cinema ticket system	22	21	7	6	2	1	3	1
Drug store	67	101	56	18	14	4	18	3
Multi-level marketing company	15	41	26	21	3	2	10	3
Reports' generator	12	21	9	9	0	0	4	2

The results of the experiment show that even an expert can find less than a half of inconsistencies in a middle-size project and the developed tool could assist a designer to avoid not only misprints but also inaccurate assignment of responsibilities between classes.

## 7 Conclusions

This chapter offers a simple method for detecting inconsistencies between different UML diagrams. It was implemented as an Eclipse plug-in and tested on some types of cross-diagram inconsistencies.

Another more general method for checking inconsistencies in UML design is proposed. It uses the unified model with graph representation of the design components and formulae of the first order predicate logic to represent relations which should be satisfied to make the design consistent. This approach can also be used to evaluate the quality of a design and make recommendations on its improvement on account of better use of the main principles of the object-oriented design.

## References

1. Andrews, A., France, R.B., Ghosh, S., Craig, G.: Test Adequacy Criteria for UML Design Models. *Journal of Software Testing, Verification and Reliability* 13(2), 95–127 (2003)
2. Fischer, G., et al.: The Role of Critiquing in Cooperative Problem Solving. *ACM Transactions of Information Systems* 9(3), 123–151 (1999)

3. Briand, L., Labiche, Y.: A UML-based approach to system testing. *Software and System Modeling* 1(1), 10–42 (2004)
4. Souza, C.R.B., et al.: Using Critiquing Systems for Inconsistency Detection in Software Engineering Models. In: *Proceedings of the Fifteenth International Conference on Software Engineering and Knowledge Engineering (SEKE 2003)*, San Francisco Bay, pp. 196–203 (2003)
5. Souza, C.R.B., et al.: A Group Critic System for Object-Oriented Analysis and Design. In: *Proceedings of the 15th IEEE International Conference on Automated Software Engineering (ASE 2000)*, pp. 313–316 (2000)
6. Ghosh, S., France, R.B., Braganza, C., Kawane, N., Andrews, A., Pilskalns, O.: Test Adequacy Assessment for UML Design Model Testing. In: *Proceedings of the International Symposium on Software Reliability Engineering*, pp. 332–343. Denver, Co. (2003)
7. del Mar Gallardo, M., Merino, P., Pimentelis, E.: Debugging UML Designs with Model Checking. *Journal of Object Technology* 1(2), 101–117 (2002)
8. Gogolla, M., Bohling, J., Richters, M.: Validation of UML and OCL Models by Automatic Snapshot Generation. In: Stevens, P., Whittle, J., Booch, G. (eds.) *UML 2003*. LNCS, vol. 2863, pp. 265–279. Springer, Heidelberg (2003)
9. Kawane, N.: Fault Detection Effectiveness of UML Design, Model Test Adequacy Criteria. In: *Supplementary Proceedings of the International Symposium on Software Reliability Engineering*, pp. 327–328. Denver, Co. (2003)
10. Kawane, N.: *EPTUD: An Eclipse plug-in for testing UML design models*. Master’s of science thesis, Colorado State University, Fort Collins, Colorado (2005)
11. Mellor, S., Balcer, M.: *Executable UML: A Foundation for Model Driven Architecture*. Addison Wesley Professional (2002)
12. Dinh-Trong, T., Kawane, N., Ghosh, S., France, R.B., Andrews, A.A.: A Tool-Supported Approach to Testing UML Design Models. In: *Proceedings of 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2005)*, Shanghai, China (2005)
13. Object Management Group: *UML 2.0 Superstructure Specification* (2005), <http://www.uml.org/>
14. Pender, T.: *UML Bible*. Wiley Published Inc. (2003)

# Coursework Peer Reviews Increase Students' Motivation and Quality of Learning

Vadim Ermolayev, Natalya Keberle, and Sergey Borue

Zaporozhye National University, Department of Information Technologies,  
Zhukovskogo st. 66, 69063 Zaporozhye, Ukraine  
vadim@ermolayev.com, nkeberle@gmail.com, bsu@znu.edu.ua

**Abstract.** This chapter reports about a pedagogical experiment at Zaporozhye National University (ZNU) aiming at improving students' motivation and learning quality in our Computer Science Bachelor program. The major novelty in teaching and learning practice introduced in the experiment was the use of peer evaluation for the assessment of coursework reports in two disciplines – one in the second and the other in the fourth year of study. The results were compared to the historical data collected in the previous 3-4 years for which traditional assessment by instructors was applied. Our experiment proved that constructively exploiting students' aspirations for informal leadership and incurred competition is effective and yields some increase in motivation to learn and learning quality. The assessments were also subjectively regarded as more clear and better justified by the students involved in the experiment. A good side effect is also that the students learn the working patterns broadly used by professionals in their field in academia and industry for making qualitative and unbiased peer evaluations.

**Keywords:** Incentive, motivation, learning quality, peer evaluation, Computer Science Bachelor program, coursework.

## 1 Introduction

Recent higher education experience reveals a substantial decrease in popularity of University education and degrees which is reflected for instance in the decrease in degree completion rates [1]. Researchers analyzing the reasons for this decrease point out:

- The rise of pragmatic attitudes to education in life planning among young people
- The trend for devaluation of a University degree as a factor facilitating to employment and career development

As a result and because of the concurrent demographic and economic crises a substantial decrease of interest to quality learning among University students is observed. For example the quantity of students enrolled for the 1-st year is going down. This observation is also supported by the decrease in student grades. Consequently, the employers suffer from a decreased quality of the graduates coming to the labor market.

Academia can not remove or relax demographic or economic factors unfortunately. Hence, the only feasible way of keeping academic performance at a competitive level is focusing on the stimuli for their students based on more social than purely pragmatic basics. For example, exploiting the value of informally assessed professional capability and leadership in student groups may be an effective way of stimulating spending more effort on learning.

The research presented in this chapter aims at finding such stimuli for Computer Science students based on their attitude to informal leadership grounded on professional competencies. The idea behind our pedagogical experiment was to place the subjects in an environment which is similar to a professional one and offer them peer-evaluation of their individual courseworks. Hence, the higher the grades a person gets from his or her peers in such an evaluation – the higher becomes the professional reputation of the person in the class – a professional group of the peers.

In fact the approach we have taken is not new and has been effectively exploited:

- In the academic world in peer evaluation mechanisms
- In social networks for forming communities of interest and building social reputation for the individuals

Such stimuli are qualified as **solidary** (in contrast to **material**) **incentives** [2] i.e. intangible rewards from the act of being a part of a group having coherent interests. In our research we build upon the mechanisms and tool support adopted from the mentioned domains. We involve students in peer evaluation of their individual coursework assignment reports similarly to that of reviewing conference papers. We measure their qualification by: comparing evaluations by peers and instructors – assignment results; and measuring deviations between their individual scorings and the mean values – reviewer competence. Anonymized results are then made available to the group.

We have observed that being an evaluator of the peers' work proved to be a noticeable incentive for the subjects who took part in our pedagogical experiment. Knowing that their results will be assessed by the fellows in the class, on average they worked harder and delivered better elaborated coursework reports – both in terms of tasks coverage and quality. Consequently the degree of active involvement and the quality of individual assignment results have increased noticeably, in particular for the graduate fourth year group of our Bachelor program in Computer Science. This observation is backed up by the results presented in Section 4.

The remainder of the chapter is structured as follows. Section 2 gives an overview of the related work on incentive mechanisms relevant for motivating higher education students to participate more actively in the learning process and provide better quality outputs. Section 3 presents the setting of our pedagogical experiment. Section 4 discusses experimental results. Conclusions are drawn and plans for the future work given in Section 5.

## 2 Related Work

Motivations are denoted as "... reasons individuals have for behaving in a given manner in a given situation" (c.f. [3]). "They exist as part of one's goal structures, one's beliefs about what is important, and they determine whether or not one will engage in

a given pursuit'' (c.f. [4]). In academic settings two types of motivation are distinguished – **intrinsic** and **extrinsic**. Intrinsically motivated subjects learn for their own sake, because they enjoy learning or assess the outcome of the learning process as important for them – e.g. [5]. Extrinsic motivation is driven by a desire of getting rewards – from the others; or avoiding punishment. Students motivated extrinsically focus on receiving approval – like judgements by lecturers and peers – e.g. [4]. Our approach, though welcoming intrinsic motivation, focuses on obtaining the utility of exploiting student's extrinsic stimuli – which proves to become more widespread and influential in the current economic settings.

Many authors stress the importance of a skill to maintain and enhance students' motivation as one of the core capabilities of a University lecturer. "A wide variety of theories of learning and teaching recognises motivation as an essential prerequisite for successful learning. The ability to maintain and enhance student motivation is therefore one of the most important skills ..., and many publications and training programs devote considerable space and time to this matter. Application of this theoretical knowledge to practice, however, remains difficult due to the complexity of the concept and the number of different models of motivation available" (c.f. [6]). Our research is focused exactly on the application of motivation stimuli to practice in a Bachelor level Computer Science program – so the experimental data we have analysed spans across several disciplines taught from the 1-st to 4-th year of the program at Zaporozhye National University (ZNU).

The mainstream of experimental studies in higher education teaching and learning centres around using the methodologies of individual subjective assessment by subjects – based on interviews, questionnaires, etc (e.g. [7] to mention just one of many relevant publications). In contrast to the mainstream methodology, we exploit the collaborative character that is intrinsic to student collectives and base our approach on well known social and peer approaches – this is why peer evaluation is used. Such a method allows us not only to collect and analyse individual judgements, but also rate the subjects by their own cross-judgements and stimulate healthy competition – thus increasing positive stimuli.

A solid body of related work giving insight on relevant ways of incentivising subjects to contribute their creative work for reputation could be found in socio-economic publications on on-line communities or crowd-sourcing.

Forte and Bruckman [8] stress that recognition by peers plays a role for Wikipedia contributors and is mentioned as an important factor for increasing visibility and influence in the community. Basing their analysis on the framework by Latour and Woolgar [9], Forte and Bruckman observe that the incentives for Wikipedia authors to contribute content are very similar to the motivations of scientists spending effort on publications. They conclude that a reason for behaving altruistically is a reputational "credit" in a broad sense obtained indirectly for helping, serving to the community, and sharing knowledge. This credit is often assessed in terms of increased visibility of the author's contributions, or more frequent invitations to provide featured content.

Wasko and Faraj [10] argue that if knowledge is considered a public good, knowledge exchange in on-line communities of practice is motivated by moral obligation and community interest rather than by self-interest. They prove their argument by the analysis of the results from a survey examining why people participate and share knowledge in three electronic communities of practice related to software and IT.



These results indicate that people participate primarily out of community interest, generalized reciprocity, and pro-social behavior.

A good analytical review of different relevant incentive mechanisms for participation in human-driven semantic content creation is [11]. The authors also stress the dominance of extrinsic motivation (building reputation among the peers) for increasing interest to contribute content and share knowledge.

These examples of related work in fact support our intention to exploit extrinsic motivational mechanisms for incentivizing students to take an active part in the educational activity within a social group of peers who cross-evaluate the content provided themselves.

### 3 Experimental Settings

Stimulated by the necessity to seek for a remedy for the decrease of interest in quality learning at Universities, we have planned and further conducted a pedagogical experiment at ZNU. We have focused on the individual coursework as one of the important kinds of students' creative activities in which motivation plays a very important role.

Our major objective was to check the validity of the pedagogical hypothesis:

If students are given an opportunity to act as peer-evaluators of the other students reports, their **extrinsic motivation** to:

(a) Deliver the coursework

(b) Perform as well as they can

– **will be higher** than among those who do individual work in a traditional way and are graded by their instructor only. Furthermore, the **quality** of submitted reports is expected to **be better**, as students informally compete and cross-evaluate their quality.

Finally, the **objectivity of the assessments will be higher**; those will be perceived as fair by the subjects.

For that we have:

- Chosen the disciplines: (a) for which the historical data on the coursework grades existing for several years; (b) for which the complexities of doing coursework assignments were comparable; and (c) so that the coverage of all four years of our Bachelor program was even
- Developed detailed assessment forms for inexperienced evaluators offering a clear procedure and set of explicit metrics for coursework report assessment for each involved discipline
- Chosen the academic student groups comprising the cases when the same group acted as an experimental sample and formerly a control sample; briefed the experimental groups
- Configured the set of software tools to support the experiment and developed written methodological recommendations for the subjects
- Adopted and adapted simple and effective metrics that allowed testing our research hypothesis

### 3.1 Pedagogical and Methodological Setting

The pedagogical setting of our experiment covers: the choice of disciplines; the preparation of the evaluation forms; and subjects' briefing about the evaluation procedure and tools.

First, we have chosen the disciplines with historical data and currently available in our Bachelor program, as summarized in Table 1. The table also shows that:

- More than 3 years' historical data on the coursework assignment grades is available
- The disciplines cover all 4 years of study within the program evenly

The complexity of the assignments, though different per discipline, is comparable as shown in Table 1.

**Table 1.** Choice of disciplines and complexities of related coursework assignments

Discipline	Year	2008	2009	2010	2011	Avg
Programming	I	---	100	100	100	100
Algorithms and Data Structures	II	100	150	200	<b>250</b>	175
DataBases and Information Systems	III	---	150	150	150	150
Intro to Logical Programming and AI	IV	---	150	150	<b>250</b>	183

**Legend:** numbers in columns 3-7 are coursework complexities. The cells corresponding to our experiment are shaded gray and have bolded numbers in white.

Grade data for the coursework assignments in Programming (year I) and Databases and Information Systems (year III) form our first and second baseline control datasets respectively.

The complexity of the first year coursework assignment in Programming has been chosen as basic – represented by 100 abstract points. This coursework contains a survey part on a particular topic and a practical assignment to develop a program solving a given simple problem. The complexity of the coursework in this discipline remains without change for all the 3 years of our observations.

The complexity of the third year coursework in Databases and Information Systems is also static within the period of observation. However, it is 1.5 times more complex as it contains several interrelated practical problems in database and IS development using SQL Server software. Another difference is that the subjects for this assignment were the third year students whose motivation and experience differ from those of first year students.

Observations in Algorithms and Data Structures and Introduction to Logical Programming and Artificial Intelligence contain both control and experimental (shaded gray in Table 1) data.

The complexity of the coursework assignment in Algorithms and Data Structures increases from 100 points in 2008 to 250 points in 2011. In 2008 it was very similar in structure to the coursework in Programming – a detailed written presentation of a sorting algorithm studied individually and its practical implementation in a computer program. In 2009 the task of analytically evaluating the computational complexity of the algorithm was added – raising the complexity up to 150 points. In 2010 the task of

experimental measurement of the computational complexity and comparing it to the analytical estimation was added – the complexity has therefore increased to 200 points. In 2011 the coursework has been complicated (up to 250 points) by offering a comparative evaluation exercise – the students were tasked to measure the performance of their program and compare to the performance of a program developed by a peer based on several common datasets containing records of different types.

The complexity of the coursework assignment in Introduction to Logical Programming and Artificial Intelligence increased from 150 points in 2009<sup>1</sup> to 250 points in 2011. In 2009 and 2010 the assignment contained two parts: (i) a survey of the state of the art in a sub-field of Artificial Intelligence; and (ii) development of a micro-ontology describing the major terms in this sub-field of Artificial Intelligence. In 2011 the assignment has been substantially re-thought and re-worked. It has been realized that it also needs to cover a practical work on developing a tiny expert system for solving logical problems in PROLOG. Urban traffic domain has been chosen for this part of the coursework. A student is tasked to develop a rule-based expert system in PROLOG that answers the questions taken from the driver license examination questionnaire<sup>2</sup>. For that a pattern PROLOG program is offered and a student has to:

- Check the validity of conclusions provided by the given program in several possible traffic situations on a typical T-shaped crossroad
- Refine the program by adapting it to a particular crossroad configuration of the given variant<sup>2</sup> by adding statements reflecting particular constraints
- Evaluate the refined program by offering all possible traffic situations as inputs
- Document the results in the coursework report

This addition to coursework assignment has added 100 points in complexity – which raised the overall complexity of the coursework up to 250 points.

Secondly, we have developed the evaluation forms for coursework reports in both disciplines. An example of a fragment of an evaluation form is pictured in Fig. 1. The forms are in fact structured questionnaires covering all the sections of the report and suggesting several weighted Likert scale [12] based metrics covering several aspects that were different for each section. Tables 2 and 3 contain the lists of the report sections and evaluation questions for both disciplines. Weights represent the importance of a given evaluation aspect in the Section – given that the sum of the weights per sections equals 100 per cent. Sections also have different importance indicated by the maximal number of grade points given in brackets.

It has been decided that the overall grade for a coursework report of maximum 20 points is divided in two parts:

- Coursework grade (0 – 15 points) computed as a mean of the three assessments done by two peers and one instructor
- Evaluation grade (0 – 5 points) computed as 5 minus the mean of the absolute values of the deviations of the subject's evaluation scores from the corresponding mean scores. So, the closer an individual scoring is to the mean scoring in all the

---

<sup>1</sup> In its current format the course has been given for the first time in 2009.

<sup>2</sup> <http://www.gai.ru/voditelskoe-udostoverenie/examen-pdd-online/>  
last accessed on 11.08.2012

evaluation assignments – the higher the resulting evaluation grade is. This method of computing evaluation grades may be considered as unfair at first look. Indeed, if there were two student reviewers A and B, A gave the same grade as the instructor whilst B's grade was very different, then A would have been penalized because of B's scoring. This interpretation would be correct only if B made a mistake. However it is possible that both A's and instructor's decisions were wrong, though less likely. So, our method penalizes both students but proportionally to the likelihood of their mistake.

**Table 2.** Evaluation aspects covered by review forms, and scoring weights. Discipline: **Algorithms and Data Structures.**

Aspect to Evaluate	Weight %
<b>Section 1: Sorting method and algorithm</b> (0-2 points)	
1.1 Is the description of the family of sorting methods given?	25
1.2 Is the algorithm described sufficiently completely and clearly?	50
1.3 Is algorithm stability analyzed?	25
<b>Section 2: Software implementation</b> (0-3 points)	
2.1 Is the source code provided?	10
2.2 Does the implementation comply with the algorithm described in Section 1?	15
2.3 Are the implementation decisions described sufficiently fully and clearly?	30
2.4 Are the constraints (absence of) wrt input data explained and justified?	10
2.5 Does the provided software work?	25
2.6 Is the source code reasonably commented?	10
<b>Section 3: Theoretical estimation of computational complexity</b> (0-2 points)	
3.1 Is the estimation the computational complexity given?	50
3.2 Is the graphical illustration of the computational complexity given?	50
<b>Section 4: Computational experiment – comparison with other algorithms</b> (0-3 points)	
4.1 Are the data sets chosen correctly	50
4.2 Are the other algorithms for comparison chosen reasonably?	50
<b>Section 5: Experimental assessment of computational complexity</b> (0-3 points)	
5.1 Are the rules and programming solution for measuring computational complexity described?	30
5.2 Is the comparative analysis of the computational complexity given?	30
5.3 Is the experimental assessment compared with the theoretical estimation?	30
5.4 Is the graphical illustration of the computational experiment results given?	10
<b>Section 6: Conclusions</b> (0-3 points)	
6.1 Do the conclusions reflect the results obtained?	50
6.2 Are the references to the adopted components given?	30
6.3 Is the report compliant to the template (abstract, contents, references, appendices)?	20

**Table 3.** Evaluation aspects covered by review forms, and scoring weights. Discipline: **Introduction to Logical Programming and Artificial Intelligence.**

Aspect to Evaluate	Weight %
<b>Section 1: Survey (0-6 points)</b>	
1.1 Is the graph of the basic notions elaborated?	20
1.2 Are the basic terms denoted correctly and completely?	40
1.3 Are the major problems in the field covered?	40
1.3.1 Are the problem statements given?	
1.3.2 Is the actuality of these problems explained?	
1.3.3 Are the descriptions of solution methods given?	
1.3.4 Are the surveyed solution methods compared?	
<b>Section 2: Solving Problems in Visual PROLOG (0-7 points)</b>	
2.1 Is task 2 solved?	10
2.2 Is task 3 (traffic expert system design) solved?	30
2.2.1 Are schematic descriptions of situations at T-shaped crossroad given?	
2.2.2 Are predicates and goals for situations fully described?	
2.2.3 Are predicates and goals compliant to the schematic descriptions?	
2.3 Is task 4 (traffic expert system implementation) solved?	30
2.3.1 Are predicates and goals specified correctly?	
2.3.2 Does the developed expert system work?	
2.3.3 Are the implementation decisions documented?	
2.4 Is task 5 (traffic expert system refinement) solved?	30
2.4.1 Are predicates and goals specified correctly?	
2.4.2 Is the sense of the predicates and structures explained?	
2.4.3 Are all possible traffic situations described?	
2.4.4 Does the refined Expert System work?	
<b>Section 3: Conclusions (0-2 points)</b>	
3.1 Do the conclusions reflect the results obtained?	50
3.2 Are the references to the adopted components given?	30
3.3 Is the report compliant to the template (abstract, contents, references, appendices)?	20

### 3.2 Experimental and Control Groups

Two experimental groups in the second and fourth year of study were selected so that the historical coursework grade data was available for them. For comparison, the control data about the grades in the other groups of different years of study and in all four chosen disciplines was taken into account. The groups, for which the control data was taken into account, were further treated as control groups. Table 4 depicts the

distribution of our control and experimental groups over the years of study. As it could be seen in Table 4, the experimental groups are also control groups but in different disciplines and years of study. So, different ways of comparing subjects' activity and performance in fulfilling coursework assignments arise:

- The same group in different years
- The same group in different disciplines
- The same group as an experimental one and doing the work in a traditional way; etc.

**Table 4.** Experimental and control groups

Group No	2008		2009		2010		2011	
	Year of Study	Role	Year of Study	Role	Year of Study	Role	Year of Study	Role
8216			IV	C				
4327	II	C	III	C	IV	C		
<b>4328</b>			II	C	III	C	IV	E
4329			I	C	II	C	III	C
<b>4320</b>					I	C	II	E
4321							I	C

**Legend:** C – control group; E – experimental group.

At the beginning of the experiment the subjects of our two experimental groups were briefed about: the deadlines; the objectives of peer evaluation; the structure and the content of the evaluation forms; the grades that would be assigned for the reports and for the reviews; the tools they would use in the peer evaluation process.

### 3.3 Instrumental Setting

Two procedures have been chosen for evaluation that differed in the used tools. For the experiment with the second year students the workflow based on e-mail exchange and manual supervision has been adopted. For the fourth year students we have introduced the EasyChair Conference Management System<sup>3</sup> as a tool to manage the process, final ranking and grading. In both cases structured evaluation forms have been offered to the subjects to be filled out using MS Excel.

## 4 Experimental Results and Discussion

The evaluation process was organized and executed similarly to the peer evaluation of conference papers by program committee members. Students were invited to serve on the evaluation panel and the review assignments have been made by the instructors

<sup>3</sup> The service is persistently available at <http://www.easychair.org/>

who acted as program chairs. The results of evaluation have been collected and processed using two different patterns:

- For second year students – collected by e-mail and processed manually using Excel spread sheet as shown in Fig. 2
- For fourth year students – collected and processed using the EasyChair installation which provided data for a very similar score table as the one in Fig. 2

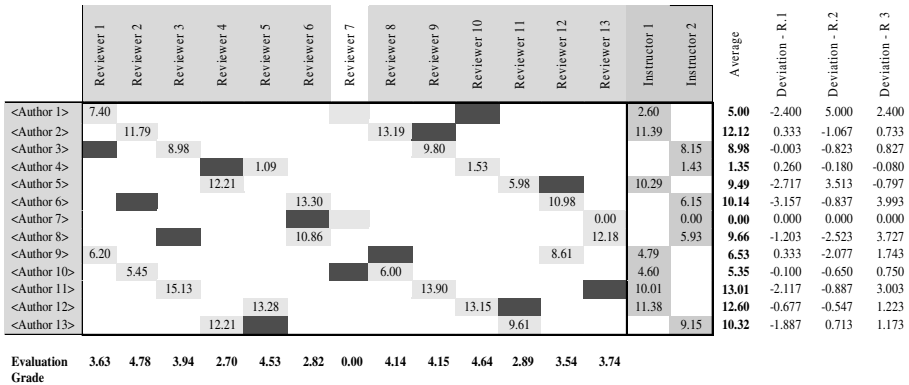


Fig. 2. Review results visible to instructors (anonymized for the publication)

The authors of the coursework reports were further notified by e-mail about their individual results and rating position in the overall list (Fig. 3). Again, the notifications to the second year students were manually communicated by e-mail; and the fourth year students were notified by e-mails sent automatically by the EasyChair system.

#### 4.1 Additional Effort for Tutors

As experienced, the additional instructors’ effort for organizing and managing the peer review process was substantial.

The major part of their additional work could be qualified as the set-up effort:

- Developing review forms (Fig. 2)
- Compiling briefing manuals for the student reviewers
- Preparing management tables;
- and Configuring the software tools

The result of this effort may however be re-used quite substantially. So the start-up effort may be regarded as an initial investment and neglected in further considerations of the tutor effort overheads in the context of a discipline.

Following two different workflows for the second and fourth year students implied different management efforts because of using different toolsets. Overall, using EasyChair Conference Management System appeared to require 3 times less effort than using just e-mail and MS Excel.

	Reviewer 1	Reviewer 2	Instructor 1	Instructor 2	Average	Review Points	Total
<Author 1>	7.4		2.6		5.00	4.64	10
<Author 2>	11.79	13.19	11.39		12.12	4.15	16
<Author 3>	8.98	9.8		8.15	8.98	3.63	13
<Author 4>	1.09	1.53		1.43	1.35	2.70	4
<Author 5>	12.21	5.98	10.29		9.49	3.54	13
<Author 6>	13.3	10.98		6.15	10.14	4.78	15
<Author 7>		0		0	0.00	2.82	3
<Author 8>	10.86	12.18		5.93	9.66	3.94	14
<Author 9>	6.2	8.61	4.79		6.53	4.14	11
<Author 10>	5.45	6	4.6		5.35	0.00	5
<Author 11>	15.13	13.9	10.01		13.01	3.74	17
<Author 12>	13.28	13.15	11.38		12.60	2.89	15
<Author 13>	12.21	9.61		9.15	10.32	4.53	15

(a) Score table for the second year class

No	Author, Title	Scores (0-15)	Average	Decision
1	<Author 1>.LPAI CourseWork Report	13(4),13(4),13(3)	13	ACCEPT
13	<Author 13>.LPAI CourseWork Report	11(3),13(4),13(2),13(4)	12.5	ACCEPT
15	<Author 15>.LPAI CourseWork Report	11(3),13(2),13(4)	12.3	ACCEPT
23	<Author 23>.LPAI CourseWork Report	11(4),7(2),11(4)	10.2	accept?
16	<Author 16>.LPAI CourseWork Report	11(3),11(1),9(3)	10.1	accept?
26	<Author 26>.LPAI CourseWork Report	9(1),9(3),11(4)	10	accept?
17	<Author 17>.LPAI CourseWork Report	9(3),9(2),11(4)	9.9	accept?
18	<Author 18>.LPAI CourseWork Report	9(3),11(4),7(1)	9.8	accept?
25	<Author 25>.LPAI CourseWork Report	9(3),11(3),9(2)	9.8	accept?
14	<Author 14>.LPAI CourseWork Report	7(2),7(3),11(4)	8.8	borderline
27	<Author 27>.LPAI CourseWork Report	9(4),11(4),3(2)	8.6	borderline
22	<Author 22>.LPAI CourseWork Report	7(2),7(3),11(3)	8.5	borderline
24	<Author 24>.LPAI CourseWork Report	9(2),7(3),9(4)	8.3	borderline
21	<Author 21>.LPAI CourseWork Report	7(2),11(3),5(2)	8.1	borderline
28	<Author 28>.LPAI CourseWork Report	9(3),7(4),7(3)	7.6	borderline
3	<Author 3>.LPAI CourseWork Report	3(4)	3	reject
5	<Author 5>.LPAI CourseWork Report	3(4)	3	reject
10	<Author 10>.LPAI CourseWork Report	3(4)	3	reject

(b) Score and rating table for the fourth year class. Numbers in brackets in column 3 indicate self-assessments of how confident the reviewers were in their evaluation judgments.

Fig. 3. Anonymized resulting score and rating tables



**Table 5.** Experimental Results

Discipline	Year		Group No	Average Score (0-20)	Average Submission Ratio			Average Score among Submitted	
	of Study	Calendar			No of Submissions	Total Students	Ratio	Factual (0-20)	Aligned by Complexity
Programming (PR)	I	2009	4329	5.00	9	23	0.39	12.78	12.78
		2010	4320	10.33	9	15	0.60	17.22	17.22
		2011	4321	4.06	7	16	0.44	9.29	9.29
Algorithms and Data Structures (ADS)	II	2008	4327	9.68	21	31	0.68	14.29	14.29
		2009	4328	12.03	21	29	0.72	16.62	24.93
		2010	4329	8.50	10	19	0.53	15.30	30.60
		2011	4320	10.13	13	15	0.87	11.69	29.23
DataBases and Information Systems (DBIS)	II	2009	4327	11.70	14	23	0.61	19.21	28.82
		2010	4328	12.00	18	33	0.55	18.67	28.00
		2011	4329	11.00	12	19	0.63	18.33	27.50
Introduction to Logic Programming and Artificial Intelligence (LPAI)	IV	2009	8216	4.47	10	36	0.28	16.10	24.15
		2010	4327	4.76	6	21	0.29	16.67	25.00
		2011	4328	7.14	15	28	0.54	13.33	33.33

## 4.2 Interpretation of Experimental Results

Table 5 contains the summary of our experimental findings and is structured as follows:

- Broad horizontal sections correspond to the data related to one discipline. Two of them are baseline (as explained in Section 3.1) – Programming and Databases and Information Systems. The other two contain both control and experimental data – Algorithms and Data Structures (second year) and Introduction to Logic Programming and Artificial Intelligence (fourth year). Experimental data is in the rows that are shaded gray.
- The **Year** column informs about the timing attribution of data (years of study and calendar years)
- The **Group No** column associates the rows to the academic groups. Group numbers may be found similar in several cases – reflecting the availability of both

control and experimental measurements for several groups in different years and disciplines.

- The **Average Scores** are in fact based on the total number of students in a group which makes it different to the scores in the last two columns computed based on the number of submitted reports.
- **Average Submission Ratio** is in fact the measure that reflects the motivation of our students to submit their work
- The **Factual Average Scores** are the averages for the submitted reports. but without balancing them by coursework complexity
- Finally, the rightmost column contains the score averages multiplied by the complexity scaling factors provided in Table 1

Let us explain now how the results given in Table 5 and further interpreted graphically in Fig. 4 prove our research hypothesis. The Y-values in Fig. 4(a) are the numbers from the Submission Ratio column of Table 5; while the Y-values in Fig. 4(b) are taken from the Aligned by Complexity column of this table.

**Firstly**, we expected that the introduction of peer reviews as a non-traditional way of teaching will **increase** students' **extrinsic motivation**. This expectation was valid as pictured by the values of the submission ratio. Indeed, the ratio of coursework submission in our experiment with the second year students reached the global maximum of 0.87 across all the disciplines. The next lower value was 0.72 which is 15 per cent lower. For the fourth year subjects the increase in motivation was not that significantly high overall, though very substantial within their year of study. Indeed the reached submission ratio of 0.54 is 1.86 times better than the next lower value of 0.29 in 2010.

**Secondly**, the **quality of submitted reports** may be interpreted as quite average in our experiments: 11.69 in the second year and 13.33 in the fourth. The registered decrease in scores, compared to the previous year, is: 23.96 per cent for the second year; and 20.03 per cent for the fourth year. An explanation for that decrease in quality is twofold:

- (i) As the ratio of submissions increased the proportion of the best students (who always submit their work) decreased – so did the average scores. For the second year the ratio increase was 15 percent versus a 23.96 decrease in scores<sup>4</sup>. However, for the fourth year the increase in submission ratio (86 per cent) substantially outperformed the decrease in average score (20.03 per cent). Hence, it could be concluded that our approach proved to be effective for the final year students of our Bachelor program.
- (ii) The observed decrease in scores is to some extent explained by the increase of coursework complexity. Indeed, the maximal values of the average scores have been reached in the cases with substantially less complicated coursework assignments – as

---

<sup>4</sup> It has to be noted that the decrease in scores should not be regarded as dramatic. If the grades of only the best halves of the groups have been taken into account – to compensate the negative effect of the increase in submission ratio – their averages would have been 15.14 for the experimental and 15.30 for the control samples. The difference would have been only 0.16 points (0.8 per cent).

explained in Table 1. For example, the global maximum of 19.21 corresponds to the assignment weighted 150 points. It is ‘outperformed’ by the score of 13.33 in our fourth year experiment because the complexity of the experimental coursework is 250 points. As pictured in Fig. 4(b), this imbalance is corrected by the values shown in the Aligned by Complexity column of Table 5.

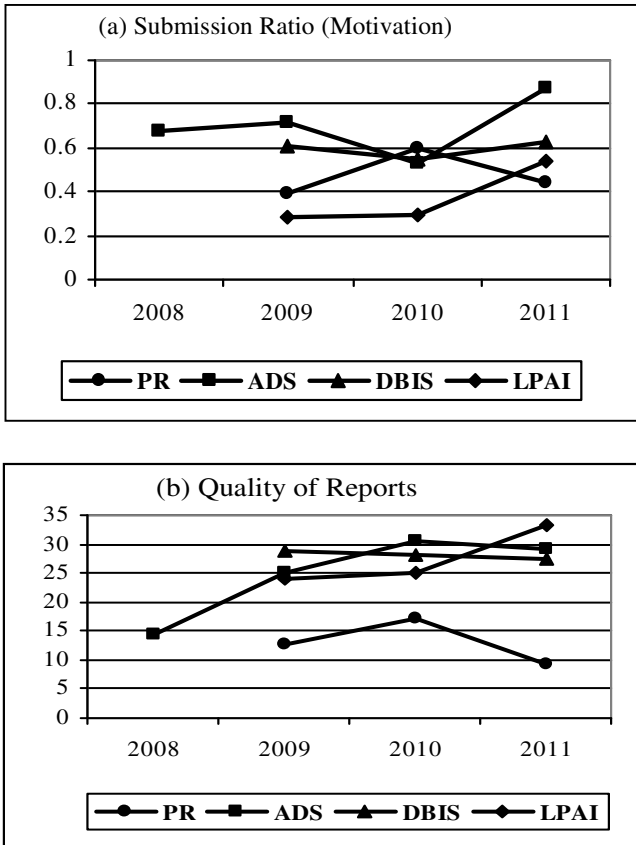
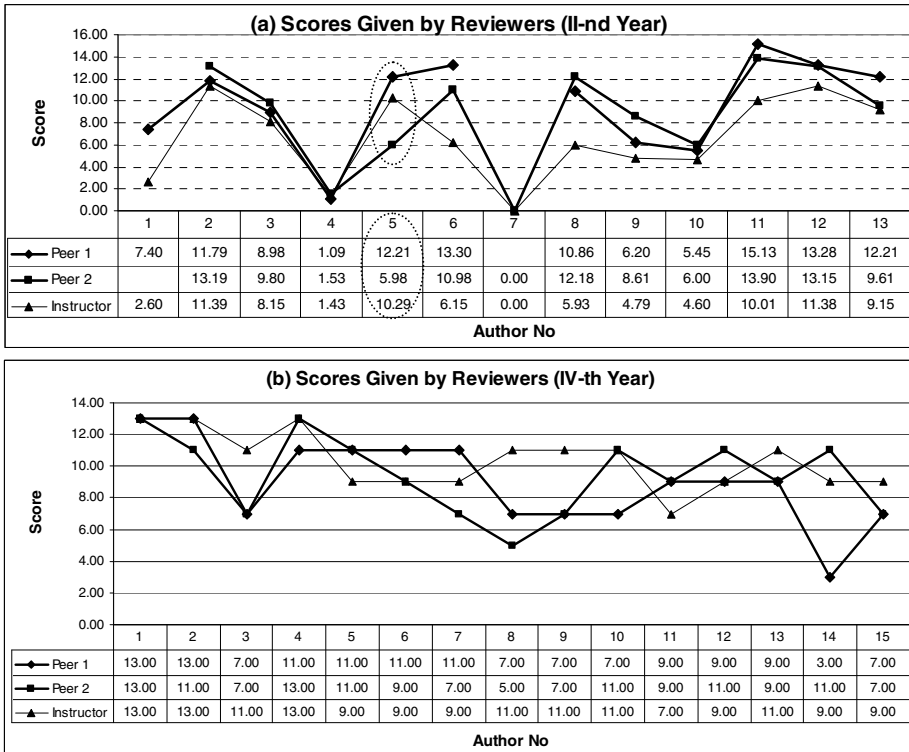


Fig. 4. Graphical interpretation of the increase in motivation and quality of work

Thirdly, the analysis of the **quality of evaluation results**, which has direct implications on subjective perceptions of fairness by students, reveals some interesting findings.

While analyzing the quality of reviews we first looked at how similar or deviated the assessments were. The grades in the class of Algorithms and Data Structures given by student peer reviewers and instructors are quite well aligned – as could be seen in Fig. 5 (a). Overall the instructors were noticeably harsher, though the proportion of the grades between different coursework reports follows the overall tendency demonstrated also by the student reviewers. The only fluctuation is for the report (author) No 5, where one of the peer reviewers gave the lowest grade. Interestingly, the picture for the fourth year class in the Introduction to Logic Programming and Artificial Intelligence

is absolutely different – the grades by different peer reviewers and also instructors are uncorrelated. Generally, the instructors were more generous in scoring, perhaps resolving doubts in favor of victims. The students were harsher in the majority of cases, with quite substantial deviations in scores sometimes (for example the scores for the reports No 14, 7, 10).



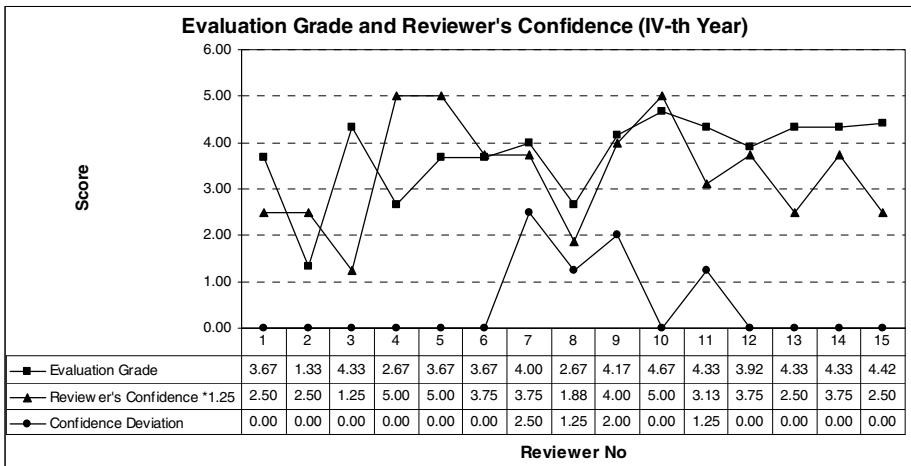
**Fig. 5.** The distribution of coursework grades in the second year and fourth year classes

There are two possible explanations for this difference. Firstly, the evaluation criteria for LPAI courseworks were not perhaps clear enough for inexperienced student reviewers, which allowed too much freedom in interpretation. This fact suggests that the review forms for this discipline have to be better elaborated and structured. Secondly, and with less probability in our opinion, the fourth year students perhaps treated the exercise as more serious and competitive for them as they were in the final year of study.

Consequently, we checked if good students are also good reviewers – i.e. are the evaluation and coursework grades well correlated. If so, there is a good chance that the reviews were fair and were accepted as fair by the coursework authors. The correlations are shown in Fig. 6. In fact the grades are correlated quite well, though there are some imperfections in the graphs outlined by dotted ellipses. Reviewer No 7 in the second year class simply did not submit reviews – so got 0 points for evaluation. Reviewers 11 and 13 in the second year class just behaved contrary to being graceful to their fellow peers (as the majority did)



These interviews revealed that the students treat their grades as more clear and fair compared to the previous experience, even if the scores were lower both individually and on average (column Factual of Table 5).



**Fig. 7.** The correlation between evaluation grades and reviewers’ self-confidence in their assessments. Reviewers’ confidence values (0-4) are multiplied by the factor of 1.25 to be evenly based with evaluation grades (0-5).

### 5 Concluding Remarks and Future Work

This chapter reported about our pedagogical experiment undertaken for seeking a way of improving extrinsic motivation and learning quality of Computer Science students in our Bachelor program at ZNU. The increase in motivation has been proven convincingly. Exploiting students’ aspirations for informal leadership and incurred competition constructively is effective and attracts people to learning. A gain in the quality of learning was a little bit over-estimated. Indeed, having involved more students in a creative learning activity does not guarantee that the quality of their work increases dramatically by miracle. However, the increase in motivation helped increasing also the quality to some degree – as shown in the previous section.

A good side effect is also that the students learn the working patterns broadly used by the professionals in their field in academia and industry for making qualitative and unbiased peer evaluations.

The results discussed in Section 4 appeared to be positive also for the other colleagues at the department of IT at our University. So, we plan to extend the experiment by covering more disciplines and collecting a broader sample of results in the near future. Among other things, this will allow us to base our work on a statistically representative set of subjects for making the results statistically valid. Finally, we plan to undertake an evaluation of the objectivity of the scoring in our settings.

**Acknowledgement.** The results reported in this chapter have been obtained in the pedagogical experiment conducted at the Department of Information Technologies of Zaporozhye National University. The authors are grateful to all the colleagues at the Department who took part in discussions and helped in setting and performing the experiment. The authors would also like to thank the anonymous reviewers whose comments and suggestions helped improve the chapter substantially.

## References

1. Bound, J., Lovenheim, M.F., Turner, S.: Why Have College Completion Rates Declined? An Analysis of Changing Student Preparation and Collegiate Resources. *Am. Econ. J. Appl. Econ.* 2(3), 129–157 (2010), doi:10.1257/app.2.3.129
2. Clark, P.B., Wilson, J.Q.: Incentive Systems: A Theory of Organizations. *Administrative Science Quarterly* 6(2), 129–166 (1961)
3. Middleton, J.A., Photini, A.: Spanias: Motivation for Achievement in Mathematics: Findings, Generalizations, and Criticisms of the Research. *J. Research in Mathematics Education* 30(1), 65–88 (1999)
4. Ames, C.: Classrooms: Goals, structures, and student motivation. *J. Educational Psychology* 84, 261–271 (1992)
5. Middleton, J.A.: A Study of Intrinsic Motivation in the Mathematics Classroom: A Personal Constructs Approach. *J. Research in Mathematics Education* 26, 254–279 (1995)
6. Ramirez-Iguez, R., Canton, U.: Understanding Motivation in Large Groups of Engineering and Computing Students. In: *Proc. of Engineering Education Inspiring the Next Generation of Engineers* (2010)
7. Wong, S.H.S.: Motivating Students to Learn through Good and Helpful Coursework Feedback. In: *Proc. Engineering Education Inspiring the Next Generation of Engineers* (2010)
8. Forte, A., Bruckman, A.: Why do People Write for Wikipedia? Incentives to Contribute to Open-Content Publishing. In: *Proc. 41st Annual Hawaii Int. Conf. on System Sciences, HICSS* (2008)
9. Latour, B., Woolgar, S.: *Laboratory Life: the Construction of Scientific Facts*. Princeton University Press, Princeton (1986)
10. Wasko, M., Faraj, S.: It is what one does: why people participate and help others in electronic communities of practice. *J. Strategic Information Systems* 9, 155–173 (2000)
11. Cuel, R., Morozova, O., Rohde, M., Simperl, E., Siorpaes, K., Tokarchuk, O., Wiedenhofer, T., Yetim, F., Zamarian, M.: Motivation Mechanisms for Participation in Human-Driven Semantic Content Creation. *Int. J. of Knowledge Engineering and Data Mining* 1(4), 331–349 (2011)
12. Likert, R.: A Technique for the Measurement of Attitudes. *Archives of Psychology* 140, 1–55 (1932)

# Influence of Music Art Multimedia Production on Professional Competence of the Future Music Teachers

Lyudmila Gavrilova

DSPU, Donbass State Pedagogical University Street General Batyuk 19,  
84112 Slavyansk, Ukraine  
lusjamuz@mail.ru

**Abstract.** The chapter examines the urgent problem of higher musical education, which concerns the impact of multimedia technologies on formation of future music teachers' professional competence. It provides the analysis of existing multimedia production useful in training of music students. The authors present the electronic textbook Russian Music: from Ancient Times to Early 20th Century, define pedagogical objectives determining the necessity to introduce the textbook to the study of the history of music. Consequently, the structure of the manual and various possibilities of its use in teaching music students of both pedagogical and professional specialties are being analyzed.

**Keywords:** professional competence of a music teacher, multimedia musical production, multimedia textbook, history of music.

## 1 Introduction

In recent years, training of music teachers is considered to be determined by two main trends, one of which is competence-based approach, and the other is the process of informatization of education. Formation of music teachers' professional competence became an object of thorough scientific and theoretical investigation, aimed also at the modes of application of information and communication facilities in teaching practice.

In the latest investigations professional competence is defined as an ability to teach music, essential for further development of the teacher's skills. M. Mykhas'kova infers that music teacher's professional competence is an integrative complex unity of such components as musical taste and abilities, practical skills of singing, playing a musical instrument, choir leading, creative artistic experience [4]. I. Poluboyaryna states that structure of music teachers' professional competence includes three components: personal component, communicative component and activity component, each embracing specified kinds of competence. For instance activity component includes methodological, didactic, musical, scientific, artistic, managerial and organizational kinds of competence [6].



Some researchers (e.g. N. Murovana) see the essence of music teacher's proficiency in the form obtained by certain specific skills, the others (e.g. S. Svitaylo) differentiate specialty competence from professional competence as a particular type of the latter. O. Shcholokova defines professional competence in the field of art education as a personal and professional characteristics, including cognitive (professional knowledge), operational (special skills and abilities), axiological (interiorated system of artistic values) components [10].

However, none of the aforementioned definitions renders information and communication component. As we are witnessing the becoming of information society, where information and knowledge form the core of production, this fact indicates an essential contradiction between the urgent needs and the real state of music education. Thus, the up-to-date interpretation of music teacher's proficiency integrates musical and pedagogical (special) knowledge, skills, aesthetic experience and value orientations of teacher's personality and mastery of ICT. Music teacher's ICT proficiency is a component of general culture of a teacher, it also indicates the level of teacher's professional skills and their relevance to international standards in higher education.

It must be noted that practice proves the advantage of computer and multimedia technologies use over traditional teaching techniques. Multimedia technologies provide the possibility to combine different forms of information presentation (text, sound, graphics, animation, audio and video). In recent years they became widely used for music lessons in elementary school, as evidenced by the experience of teachers presented in articles of O. Huminskaya [3], L. Stolyarchuk [7], N. Novikova [5], and others. This experience concerns methodological guidance for creation of multimedia Microsoft Power Point presentations and use of STT (Software Teaching Tools) on music lessons.

Scientific and theoretical consideration is being given to the problems of using computer technology on music lessons at school. This is explicated by scientific exploration of V. Shtepa [9], who identifies such domains of computer use on music lessons as representation of historical and theoretical material, listening and music analysis, composing music, obtaining various musical information on Internet. N. Byelyavina [1] proves that computer multimedia technology provides progressive alternative forms of representation of content (auditory, visual and motor-tactile). O. Chaykovskaya [8] examines different aspects of multimedia use in the process of obtaining musical knowledge in primary school.

The system of higher special and pedagogical music education also becomes an object of critical analysis. I. Gaidenko introduces a special term "music computer technology" to define various computer tools for composing, music recording and editing, represented in the form of audiovisual media production, tape records, printed production, etc. Many Ukrainian universities provided such disciplines as "Music Informatics", "Composition, Sound Synthesis and Processing by Computer", "Computer Analysis of Musical Text", "Modern Music Technologies". Some pedagogical universities provide the course "Computer Music Printing" as a sequence of the discipline "New Information Technologies and Technical Teaching Aids". When there is a total decrease of musical literature publishing, music students get the opportunity to professionally notate their own compositions, song accompaniments, choral scores. Mastering special software conditions acquiring skills of musical notation,

orchestration and arrangement, listening to each voice separately or several voices in any combination, listening to all parties together, and use of different timbres.

Thus, the course "Musical Computer Technologies" is a part of curriculum offered to the students of the Faculty of Arts, in Sumy State Pedagogical University, and belongs to the set of optional disciplines for specialty 7.010103 - "Pedagogics and Methods of Secondary Education. Music". This course aims at obtaining and enriching theoretical knowledge and practical skills of using music computer technologies.

Since 1999 Zhytomyr State University offers a special course "Modern computer arrangement" to the students of specialty 7.02020401 - "Music Art", specialization - "Culture". This academic discipline comprises theoretical knowledge and skills concerning the basic components of multimedia facilities and their classification, MIDI-technologies, notation, music software, and the foundations of sound directing, acoustics of musical instruments.

These are only few examples of how music information and computer technologies can be introduced to higher music education in Ukraine. Incidentally, similar processes occur in the Russian education. Russian State Pedagogical University, for example, developed a new system of higher music education based on music computer technologies (Bachelor, Master). To sustain the project there was founded the educational laboratory "Music computer technology". G. Tarayeva, Professor of music theory and composition department at Rostov State Conservatory, works over the introduction of computer technology into the practice of music education. She is also the author of the book "Computer and innovation in music education. Technology of creation of electronic teaching resources".

One way to introduce modern ICT into higher musical education is to create and develop new courses, aimed at mastering music computer technologies. It is equally important to use these technologies in teaching such cultural, historical and pedagogical courses as "History of Music", "Music Theory", "Music Analysis", "Methodics of Music Teaching", as well as to create electronic teaching aids, multimedia textbooks, etc.

The objective of this research is to analyze the multimedia facilities on music art, reliable for training of music teacher's, as well as to present the multimedia textbook *Russian Music: From Ancient Times to the Early 20th Century*.

The structure of the research comprises three sections:

- Review of multimedia encyclopedias on music art, which can be efficient in forming music teachers' professional competence
- Analysis of multimedia production for children, revealing the possible ways of its use on music lessons as a factor of proficiency formation
- Presentation of the multimedia textbook *Russian Music: From Ancient Times to the Early 20th Century*

## 2 Review of Multimedia Encyclopedias on Music Art

The market of current multimedia editions offers a great number of various production, which can be differentiate into certain groups in accordance with the objectives and the modes of its use in educational practice. These are primarily multimedia encyclopedias, those editions which contain a significant amount of scientific data

and information from a wide range of issues. Among them, special attention should be paid to the following ones:

1. Encyclopedia "Masterpieces of Music" (developed by "Cyril and Methodius", 1997) – a multimediareference book that provides various information on music, its content includes an overview of musical genres (opera, piano music, cantatas and oratorio, symphonic music, ballet, instrumental concert), historical styles and traditions of music art (baroque, romanticism, impressionism, modernism, etc.). The material is given in the form of lectures including the story narration combined with corresponding slides and music fragments. Textual material of the encyclopedias is subdivided into four sections: glossary, musical instruments (about 80 articles), music compositions (stories about the most famous operas, ballets, instrumental compositions, etc.), composers (stories of life and work of 40 famous artists). To test knowledge of classical music, one can choose the section of "Test Game". The program lets the user print articles, use the search function of a required term or name.

2. Multimedia Collection "Sonata: World Culture in the Mirror of Music" (by L. Zaleski and "Three Sisters", 2004 - 2008.) is a guide of European and Russian classical music, jazz and popular music in high-quality performance, which has an extensive structure including such sections as "Not only classics" (articles about more than 100 composers, arranged alphabetically in accordance with historical epochs and genres), "Music for All" (compilation of articles about music and art, e.g. "Bach and the Baroque", "Classics and Classicism", "Music and Language"), the section "Musical Instruments", dealing with different groups of instruments (modern, vintage, ethnic, electronic ones), "Outstanding Performers" (a compilation of articles on the most outstanding representatives of instrumental and vocal performance), "Gallery", which presents relevant collections of paintings, semiserious "Quiz" which offers simple questions about music and composers, glossary of music terms.

The encyclopedia "Sonata: World Culture in the Mirror Music" provides resources for work on school lessons in music, literature, history, world culture, encourages learning, opens opportunities for interactive activities in the classroom, lets stimulate interest in learning music. Therefore this book can be used for training future music teachers.

3. Multimedia Encyclopedia "Musical Instruments" (KorAx, 2002) – a reference book where information about different musical instruments is given in four sections: "Instruments from A to Z," "Groups of instruments", "Musical ensembles", "World Musical Instruments". Each instrument is given a precise description, sustained with still images of the instrument, taken from different angles. There are also examples of sound eliciting and sound effects inherent to instrumental performance, enriched with fragments of instruments sounding solo and in various ensembles and orchestras.

4. "Virtual Museum of Musical Instruments "TerraMusicalis" (HyperMethod, 2000) presents a unique collection of St. Petersburg Museum of Musical Instruments. The encyclopedia is designed as a virtual excursion. In "Exhibition" hall one encounters detailed descriptions and illustrations of 350 instruments, supplemented with samples of their sounding, and synthesizer of all items with the virtual keyboard, and video of an artist playing the instrument), a tour through the halls of the museum is in the 3D format (panorama mode); "Excursions" hall is a globe with indicated zones of instruments' use, art gallery and the history of collecting of musical instruments;

"Masterpieces" hall exhibits the history of the most valuable instruments; "Workshop" hall represents drawings, x-ray and tomography images of instruments.

The multimedia products mentioned above can be recommended as tutorials and reference books for such special courses as "History of Music: Russian Music," "The History of Foreign Music," as well as for comprehensive subjects including "Culture", "Aesthetics", "History of Culture".

To sustain music lessons in seventh and eighth grades, when the pupils are to understand the correlation between classical music and trends of modern art, it is appropriate to use such encyclopedic multimedia books as the following:

- "Great Encyclopedia of Jazz" (DELTA-MM Corp., 2003.), which is an electronic version of the encyclopedic handbook by V. Feyyertaha "Jazz. Twentieth century" (St. Petersburg, 2001), includes more than 1,000 articles about jazz musicians (biography, stages of creative work, illustrations and discography), some articles are accompanied by the most famous songs in the format mp.3.
- "Encyclopedia of Popular Music" ("Cyril and Methodius", 2004) - multimedia edition of the main directions of modern rock, pop and jazz, represents extensive information about the musical style of the so-called "light music" group musicians, performers (with full discography), as well as composers, DJs, producers and others. The book includes numerous illustrations (portraits, photographs, slides), video and audio fragments, glossary of popular music.
- Multimedia encyclopaedias "Ballet" (2008), "Music Encyclopedia" (2006) are electronic versions of paper publications, produced in collaboration of DyrektMedia with "Great Russian Encyclopedia" They contain scientific articles on musical and choreographic arts, concerning aesthetics, music theory and choreography, historical aspects of the arts. These books clarify the artistic vocabulary and increase the thesaurus of future teachers.

Unfortunately there are no Ukrainian production of that kind. All multimedia music encyclopedias, mentioned above, are made in Russia and the possibility to use it in Ukrainian universities is quite limited.

Domestic production of such character is represented by CD Set "Treasures of Ukrainian culture" (Infodisk, 2005), a multimedia encyclopedia on various arts, created as a joint project of the international agency "Ukraine-Art", the Ministry of Culture and Arts of Ukraine, Ukrainian Association "Prosvita" and "World Music Ltd". It includes discs "Cultural Heritage of Ukraine" (information about national museums, parks, libraries, historical monuments and architecture), "Visual Arts" (review of the types of Ukrainian, creative work of famous artists with a special focus on Ukrainian avant-garde, non-figurative painting, sculpture, graphics, etc.). Three CDs of "Treasures of Ukrainian Culture" contain information about musical art. "Classical Music" disk tells about the most famous Ukrainian theaters and concert halls, the leading Ukrainian musicians, and fragments of music plays. "Folk Music" CD presents folk choirs and orchestras, ensembles and soloists from different regions of Ukraine. "Pop" CD gives a review of Ukrainian rock music with information about rock musicians, rock festivals, includes a photo gallery and a great selection of songs. The multimedia encyclopedia can be recommended for music students as an additional reference book for such courses as "History of Ukraine", "History of Ukrainian Culture", "Ukrainian Music", "Folklore", as well as music lessons at school.

### 3 Analysis of Multimedia Production for Children

It must be noted that a particular group of multimedia products are created for musical development of children. These are interesting computer music and game facilities and multimedia products of educational nature, which have to be exposed to the music students in the course of "Methodics of Teaching Music".

Multimedia products for children can be classified as follows:

1. Educational programs that relate to the basic concepts of musical knowledge, mastering the basics of playing musical instruments;
2. Developing games, aimed to musical abilities (sense of timbre, rhythm, musical auditory imagination). Moreover, we can differentiate musical gaming editions for beginners and from the ones of advanced level.

Let us consider multimedia products for children of educational and developing character.

Interactive games for beginning musicians, who are about to enter the world of music, are electronic resources designed for children of preschool and primary school age. They introduce students to the world of sounds that surround them, the basics of musical literacy and creativity, elementary musical instruments. The information about these games is given in the Table 1.

**Table 1.** Multimedia music games for beginners

<b>Name of the game, the authors, the editors</b>	<b>Characteristics of the game</b>
1. "School of Music. Magic Disc of Brownie Bu" is a musical educational game from the series "Play" (BiznesSoft Ltd, 2007) for the 4 or 6 years old children.	Brownie Bu invites a child to the music studio, where they encounter musical sounds of varying pitch and duration. They also learn the basics of musical literacy, perform rhythmic exercises, collect a real ensemble of musical instruments, learn to read the scores and create a CD of tunes.
2. "Clifford. Guess the melody" (Scholastic Inc. and New CD, 2003) for the 4 - 7 years old children.	This educational and developmental musical game trip lets a child join a friendly dog-giant Clifford in learning the basics of music literacy, including information about musical instruments, get an idea of composition, rhythm, musical styles, etc.. It also teaches singing and dancing. The game creates conditions for solving logical and creative tasks, training in mathematics and logic, helps to develop memory, creative thinking, and musical ear. Together with Clifford, the child can play different instruments in the style of jazz, rock and roll, country, waltz, etc., learn a new song or a certain dance.
3. "Murzilka. Lost Melody" (IDCOMPANY and New CD, 2009) for the children older than 6 years	A developing game that allows the child to dive into the realm of Russian folk tales and learn more about the world of music together with Murzilka, while attempting to cheer Princess Nesmiyana, the child makes a real musical journey, finds the lost magical melody. The game promotes musical abilities and skills, it also improves imagination, memory, and creativity of children.

**Table 1.** (Continued)

<b>Name of the game, the authors, the editors</b>	<b>Characteristics of the game</b>
4. "Sesame Street: Let's make music" (Akella, 2003), originally named SesameStreet: MusicMaker for children of 3 years age and older	An electronic facility that combines elements of musical simulator and educational and developmental game. The characters of the famous cartoon serial "Sesame Street" invite a child to the concert, where the child would create music himself. Such Sesame Street characters as Elmo, Ernie, Gruver and Cookie give the user introductory music lessons, teach how to distinguish high and low sounds, tell about musical instruments and show how they sound.
5. "Music School of Princess Lilifi" (Tivola and Buka, 2009)	A popular series of developing games with Princess Lilifi, an iconic character that has come into the world of computer games from the modern German literature for children. The Princess invites a child to her music school, full of such fascinating instruments as a magic harp, star xylophone, and magic piano. One can repeat melodies after Lilifi's friends or create one's own melody, and play funny games.
6. "Mary and Musical Hare" DerezaProduction and "Rus-sobit-M» / GFI ("Bestway Ltd"), 2011	A musical game. I order to teach the Hare to sing, Mary wants to build a real music school, make music instruments (flute, drum, maracas, cymbals, guitar, etc.) and invite a famous singer Nightingale. Children will learn about some musical instruments, their simple construction, and listen to their sound.

Multimedia products for more knowledgeable users, who already acquired basic knowledge of music and developed their musical abilities, is characterized by more complex structure and content. Table 2 contains the list of such aids.

The critical review of educational and developing multimedia music-gaming products, created for children of primary school age, evidences the variety of methodological approaches to the problem of early musical development.

So, the above-mentioned programs contain enough material to encourage pupils to listening to music, and increase their interest in fates of artists. The possibility to listen to the music fragments is another important feature of the music gaming facilities. Producers of the series "Playing a Game with Music" specially invited instrumental ensembles and orchestras for recording:

- "The Magic Flute. Playing Music of Mozart" is accompanied by sound fragments of the same opera (the popular arias of the Queen of the Night, Papageno and music of the overture to the opera), in the "Magic Box" children are introduced to other well-known works by Mozart (A Little Night Music, Symphony number 40, Concerto for Violin and Orchestra, Part 1, Concerto for Clarinet and Orchestra, Part 2);
- "Alice and Four Seasons. Playing the Music of Vivaldi" introduces instrumental music by Vivaldi, music of the well-known cycle is divided into twelve fragments according to the number of months in a year;
- "Nutcracker. Playing the Music of Tchaikovsky" gives children an opportunity to listen to the most famous melodies from the ballet by Tchaikovsky and miniatures from his "Children's Album" for piano, arranged for different instrumental ensembles.

**Table 2.** Multimedia facilities of advanced level

<b>Name of the game, the authors, the editors</b>	<b>Characteristics of the game</b>
1. "Drakosha in the World of Music" (GOGGames and Media Services, 2000) is an educational game program for 5 year olds	The character from the cartoon enters the world of music and learn about the musical instruments given to Drakosha (harp, balalaika, drums, violin, guitar, trumpet, piano, etc.), and the basics of musical literacy (notes, keys, duration). Deeper knowledge is offered in "Harmony Lesson" (beat, pause, tempo, dynamic shades). "Music Literature" lets the child learn more about famous composers, there are short stories about J. Bach, L. Beethoven, G. Bizet, R. Wagner, M. Mussorgsky, P. Tchaikovsky, S. Prokofiev, accompanied with fragments of their works. At the end of each unit there are tests to generalize the information received.
2. "Music Class" ("NewMediaGeneration", 1997, 2004) for 5 year olds	This is a pioneer program of primary musical education via computer, actualizing different musical activities, representing the basics of musical notation and theory. "Music Theory" section offers to know about basics of music knowledge (musical sound, meter and rhythm, signs of alteration, etc.). In "The History of Musical Instruments" section children are introduced to the history and development of musical instruments, with their construction described, and the specifics of their sound and features noted. "Computer Piano" section gives children the opportunity to improvise and play timbres of ten different instruments. "Cybersynthesizer" can be used to compile pieces of music in different genres. The games "Musical Tic-tac-toe", "Musical Dictation" and "Music Cubes" offer to test knowledge of music theory and develop a musical ear.
3. "The Magic Flute. Playing the Music of Mozart" (Quaim Interactive and Media House, 2003) for 5 year olds	An educational game program from the series "Playing a Game with Music ", it is based on the music of the Mozart's opera, and combines games, quizzes, fabulous adventures and musical encyclopaedia. Gaming tasks are accompanied with fragments from Mozart's "The Magic Flute" and some other famous works of the composer.
4. "Alice and Four Seasons. Playing the Music of Vivaldi" (Music Games International and Media House, 2005) for 5 year olds	An interactive music game from the series "Playing a Game with Music" based on the works of L. Carroll's classic "Alice in Wonderland" and A. Vivaldi's "Four Seasons". The program develops musical abilities, hearing, introduces the composer's musical legacy. "Alice and Four Seasons" contains 12 music games that teach children to listen to music, to distinguish sounds of musical instruments by ear. The structure of the game contains music encyclopedia that provides information about music instruments of symphonic orchestra.

**Table 2.** (Continued)

<b>Name of the game, the authors, the editors</b>	<b>Characteristics of the game</b>
5. "The Nutcracker. Playing the Music of Tchaikovsky" (Music Games International, Guaint Interactive, Media House, 2002 - 2004) for 5 year olds	A developing program that combines computer game, music encyclopedia and fabulous musical adventure. The structure of the program contains the following components: 9 games to develop musical hearing and abilities; musical masterpieces of Tchaikovsky, written for children, including ballet "The Nutcracker" and "Children's Album"; Music Encyclopedia, which introduces a biography of the composer, ballet libretto and the history of its creation.
6. "Children's Collection: Famous World Composers" (Garbuz.Studio and AtlanticRecords, 2006) for the children of preschool and primary school age	Ukrainian training program with elements of the game. There are stories and interesting facts about the composer's life adapted for children: "Violin, Vivaldi and Paganini," "Create blindly. Bach and Handel "," The Mystery of Mozart and Salieri ", "Beethoven and Schubert", "Piano in the life of Chopin and Liszt", "Maurice Ravel ", "Peter Tchaikovsky", "Myhaylo Verbytsky "and others.
7. "Entering School. Develop Your Musical Abilities "(IDCOMPANY,2009) for children from the youngest age to 7	An educational entertaining program that opens the world of scores, tells interesting stories about the life of famous composers, introduces musical instruments and their sounds. Music classes are differentiated by age groups. The program sections represent the following items: all about the music (stories of musical genres), notation for kids, music theory exercises and development of musical hearing, who "makes" music (stories about famous composers), all about musical instruments, musical stories. It also offers recommendations for parents.
8. "Camille Saint-Saëns. Carnival of Animals "(Alisa Studio and New CD, 2005) for 5 year olds	A training and developing computer program which represents Carnival of Animals by Camille Saint-Saëns. It combines information about animals, cartoons, tasks to develop skills of listening classical music, facilities for painting and child's creative work. The program includes 14 pieces of "Carnival of Animals". The narrator tells us about the meaning of the music. Each fragment is supplemented by a corresponding animated video illustrating the narration. The section "Tasks" allows us to reveal the child's reaction to C. Saint-Saëns' music and the narrator's explanations andand offers the means to express the child's feelings in paint.

We have to add that the music analysis on the very lesson requires sufficient knowledge of means of musical expression and musical instruments with their specific sound. For instance, the interactive games "The Magic Flute. Playing Music of Mozart " and "The Nutcracker. Playing Music of Tchaikovsky" contain encyclopedic



sections on musical instruments, presenting instruments for small orchestra: violin, viola, cello, bass, piano, harpsichord, flute, oboe, clarinet, bassoon, horn, trumpet, trombone, drums and more. Music for these instruments in various ensemble combinations is also provided by the gaming programs.

"Alice and Four Seasons" also includes an encyclopedia of musical instruments: besides the instruments of small orchestra there are articles on vibraphone, cymbals, vintage mandolin strings, theorbo, lute, supplemented with short stories and examples sounding instruments.

In our opinion, the quality of learning musical instruments in primary school can be improved through the use of specified educational and entertaining programs that include not only sections of informative character, but also provide game tasks, checking knowledge about instruments to their sounds. "The Magic Flute. Playing Music of Mozart", for example, includes such games as "Guess the Musical Instrument" (the task is to click on an instrument that sounds), "Guess the melody" (the task is to choose the quartet that performs a particular work of the composer), "Riddles of the Queen of the Night" (the task is to guess distorted tones) and others.

The interactive music game "Alice and Four Seasons" also contains many games and tasks that check out whether the child can distinguish sounds of different instruments aurally: the game "Orchestra" (the task is to identify the instrument that reproduces some sound among the ones depicted on the game screen); the game "Music Card" (the task is to find instruments sounding similarly); the game "Two orchestras" (the task is to divide musicians into two groups in accordance with piece of music they play).

Various game tasks for knowledge of musical instruments and their timbers are provided by the "Nutmacker. Playing the Music of Tchaikovsky". It includes the game "Orchestra Pit" (the task is to identify musical instruments by timbre and drag them from the pit to the stage), the game "Find me the Tree" (the task is to look for the voice-tone of each character), the game "Mad House" (the task is to guess the distorted melody), the game "Children's Album" (the task is to listen to orchestrated pieces from the collection of Tchaikovsky and determine which instruments sound in each fragment).

Thus, multimedia music gaming aids of the series "Playing the Music" raise pupils' interest in musical instruments, and are helpful for development of listening skills.

Musical training program "Children's Collection: Famous World Composers" from the series "Wolf Panas' School" is designed to represent life events and famous works of composers from different countries and eras. The elements of the game are nothing but the steps to follow the fate of the artist (J. Bach, Handel, Vivaldi, Paganini, Mozart, Beethoven, Chopin, Liszt, Ravel, Tchaikovsky, Verbytsky). It also should be said that this is the only resource in the Ukrainian language.

The other multimedia facilities can be partially used on the music lessons. The game "Entering School. Develop Your Musical Abilities" is much less interesting for children, since it is not animated, which decrease the children's attention to the theory, and makes it look rather like a studybook than a game.

The music multimedia games for the beginners can be applied to the music lessons as additional tasks and informational aids, helpful for gaining musical literacy and specifics of musical instruments. Such games as "Clifford. Guess the melody", "Murzilka. Lost Melody", and "Mary and Musical Hare" are useful in individual work of

pupils. The homeworks can be enriched through involvement of the possibilities provided by "Music School of Princess Lilifi", "Sesame Street: Let's make music", or "School of Music. Magic Disc of Brownie Bu".

"Murzilka. Lost Melody" is distinct for the use of Russian folk orchestra's accompaniment, and contains the information about traditional instruments of Russia, Spain, Turkey, and Ireland.

Multimedia means can also be used to develop the skills of musical creative work. Almost every game, reference book or musical simulator offers the task of composing a melody of certain fragments, improvising on musical instruments (usually the result of children's creativity can be saved, printed, and even recorded).

Future music teachers, trying to respond the current requirements of education, determined by informatization and computerization of educational process, ought to follow the production of multimedia teaching and developing music-gaming means and be able to introduce them to music lessons in primary school.

Thorough analysis of all musical multimedia educational game products for children should be one of the tasks for independent students' work. Knowledge of each resource is one of the conditions for the use of their educational and developmental opportunities for music lessons at school. The ability to design lessons with multimedia educational game support, using games, tests, video narratives, and music sounding, the skill of enriching the lessons with certain sections of electronic training and developing aids, transforms now into an integral component of practical training in methodics of teaching music in pedagogical universities.

#### **4 Presentation of the Multimedia Textbook Russian Music: From Ancient Times to the Early 20th Century.**

Yet, there is another aspect of the problem to be examined in the article, which concerns the idea that involvement of future teachers to information and communication technologies, including multimedia, should be provided by modern electronic teaching aids and textbooks for high school. At the present moment, there are certain achievements in Ukrainian higher education, represented by educational electronic products in computer science, physics, economics, social sciences and foreign languages. The corresponding issues in artistic education are basically limited to the electronic versions of the most popular textbooks on musico and culture studies (scanning paper editions), and separate references guides to electronic libraries, encyclopedias, or galleries.

In recent years, there have been some changes in the situation. The teachers of Melitopol State Pedagogical University developed electronic training complexes in artistic disciplines for undergraduate students and masters: "History of Western Music" (Horemchyn A.I., 2006), "Musical Culture of Ukraine of the 20th and the beginning of 21st century" (Martyniuk A.K., Kornyshev V.V., Osadchy V.V., 2008), "Choral Culture" (Martyniuk A.K., Kornyshev V.V., Osadchy V.V., 2009). All resources are available for free in online library MSPU (<http://lib.mdpu.org.ua/kat.html>).

An interesting approach to creating an electronic textbook "Music Analysis" with Microsoft Front Page is worked out by M. Dyadchenko, a lecturer in the history and theory of music at Taganrog Pedagogical Institute .

Taking into consideration the experience of colleagues, we created a multimedia textbook on the history of music, which is based on scientifically sound and normalized requirements to the content of electronic textbooks. In what follows we are going to highlight the structure of the textbook, its technical characteristics and the possibilities of its use in teaching music students.

The textbook on the history of musical art Russian Music: from Ancient Times to the Early 20th Century was created as the basic studybook for the course History of Russian Music, obligatory for the students of Slavyansk state Pedagogical University (specialty 6.010102 – Primary Education, specialization: Music). The textbook can be used not only in the system of art education in pedagogical universities, but also in professional music education, since similar courses are offered in music schools and conservatories.



**Fig. 1.** The cover of the textbook

The multimedia training product on the history of musical art Russian Music: from Ancient Times to the Early 20th Century is an electronic textbook which in contrast to the traditional printed editions contains visual and audio components. The textbook is meant to be used at university, so its structure and content follow the requirements for the design of electronic textbooks for high school and include:

- A management system, providing means of structuring the educational material, tests and feedback
- Methods that accelerate the learning process, such as hypertext and hypermedia
- Graphical tools that provide effective use of visual aids
- Test system that enables to control students' knowledge

Textbook materials are presented on 2 discs: the first DVD-ROM contains the textbook itself and the second is a compilation of music for individual listening. We should also mention the hardware requirements needed to install the tutorial, which are as follows:

- Processor 1000Mhz (recommended 2000 Mhz)
- Memory: 512 Mb (1024 Mb)
- Hard Disk 3 Gb of free memory
- Operating system: WindowsXP, Vista (32/64) Windows 7(32/64-bit)

It is important to consider the structure of the textbook and the possibility of its use in training future music teachers.

Installation of the textbook is traditional: one should insert the CD into the CD drive then select "autorun.exe" in the menu, which will lead to a startup tutorial.

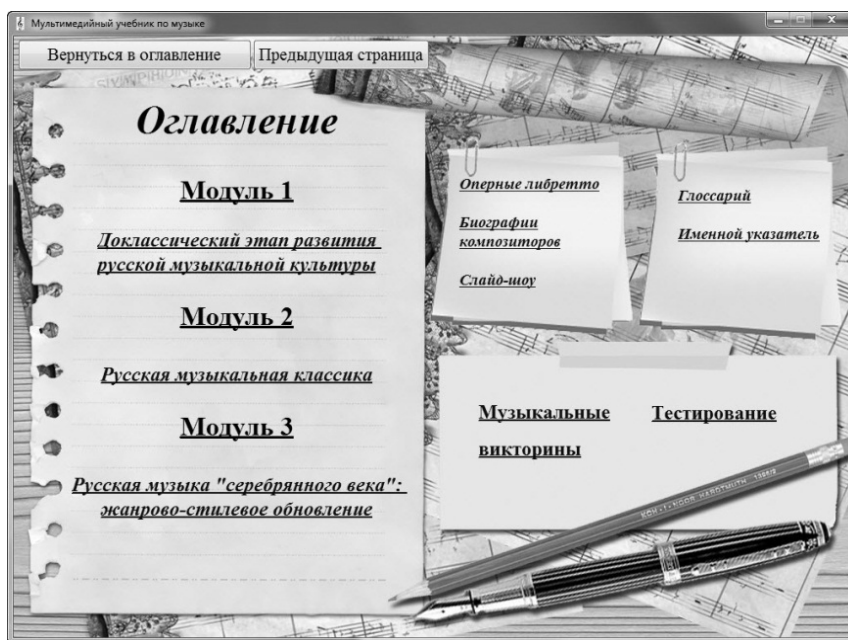


Fig. 2. Menu

The course of the history of Russian music covers the period from ancient times to the early 20th century that is from the early Middle Ages to the so called Silver Age of music in Russian culture. The textbook is opening with the Menu and it is structured as follows:

Lecture (a text with hyperlinks to the glossary, index of names, biographies, opera librettos);

- Multimedia slide show
- Glossary (dictionary of musical terms)
- Index of names
- Opera librettos
- Biographies of composers
- Musical quiz in two variants that is aimed at controlling students' knowledge of certain compositions
- Tests in two variants

Music for individual listening is presented on a separate mp.3 disk. Students are offered to listen to the following works by Russian composers:

- A sufficiently large collection of Russian sacred music (from ancient chants to the works by anonymous composers of Synodal School of the late 19th century, and the finest examples of sacred music of Sergei Rachmaninoff)
- Masterpieces of Russian chamber and vocal music
- Works of Russian musical theater: the most famous operas and ballets that are traditionally studied in the course of the history of Russian music (Glinka's *Ruslan and Lyudmila*, Ivan Susanin, Dargomyzhsky's *Rusalka*, Mussorgsky's *Boris Godunov*, Borodin's *Prince Igor*, Tchaikovsky's *Eugene Onegin*, *Queen of Spades*, Stravinsky's *Petrushka* and *The Rite of Spring*)
- The best examples of instrumental music of Russian composers (symphonic works by Glinka, Borodin, and Scriabin, Tchaikovsky's symphonies, Rachmaninoff's piano concertos, separate instrumental miniatures and cycles)

The Menu opens navigation to any section of the textbook. Lectures are set in the textbook according to the historical principle. Culturological material is divided into 3 modules and 7 units.

Module 1. Pre-classical stage of development of Russian music: from ancient times to the late 18th century).

Module 2. Russian classical music ( from Glinka to Tchaikovsky).

Module 3. Russian music of Silver Age: genre and style transformations (Rachmaninoff, Stravinsky, Scriabin, and others).

Texts of lectures, modules and clusters were composed with the reference to modern printed books on the history of Russian music (I. Rapatsky *History of Russian Music: From Ancient Rus' to the Silver Age*, Moscow (2001) (In Russian). In addition we used other books (E. Orlova *Lectures on the History of Russian Music*, Moscow (1979) (In Russian); B. Asafiev *Russian Music of 19th and 20th Centuries*, Leningrad (1978) (In Russian); T. Levaya *Russian Musical Culture of the Early 20th Century in the Context of Artistic Trends of the Era*, Moscow (1991) (In Russian)), as well as multi-volume history of Russian Music edited by A. Kandinsky, and Y. Keldysh, Internet sources (such sites as [mus-info.ru](http://mus-info.ru), [classic-music.ru](http://classic-music.ru), [belcanto.ru](http://belcanto.ru), etc.). Glossary that includes 262 items and Index of names including 335 items have interesting

selection of concepts from theory of music, music history and general cultural context. Informative sections of the book are supplemented with the biographies of 25 Russian composers, and 11 of the most famous opera librettos.

Slide show serves as a multimedia application to every topic, including widely available photographic materials, reproductions of paintings, audio fragments of music compositions and video clips of operas, ballets, concerts.

Special attention is given to the system of means aimed at controlling students' knowledge of theory and musical compositions. Traditionally while studying the course of history of music, students are supposed to penetrate the world of music through listening to numerous pieces of music. The multimedia textbook provides students with an opportunity to listen to the set of compositions, and use quizzes to test their knowledge:

- To identify certain audio fragment, to name the author, the composition and the part the fragment was taken from (an act in opera, a part of symphony)
- To choose the right answer among the given variants

Passing each stage of the quiz, students can examine the record of their answers, compare them with the right options and find out the level of their knowledge. In addition, the textbook contains an actual test at the end of each unit, students are offered two variants of closed test, so they have to choose one or more correct answers.

The introduction of electronic textbook to the process of studying the history of music contributes to the solution of the following educational objectives:

- To study Russian musical culture as a whole which absorbed spiritual experience of the nation and reflected religious, philosophical, aesthetic and moral principles of certain historical periods
- To learn the main styles and trends in the Russian music of the period from ancient times to Silver Age, to get knowledge of the origins and the main stages of the evolution of Russian culture
- To specify the features of different musical styles and trends in Russian music on the basis of certain compositions by the great Russian musicians
- To develop feelings, emotions, imagination, and artistic abilities of students
- To refine their artistic and aesthetic tastes, to bring up the need in personal assessment of values
- To enlarge the scope of their thinking, to make them create the cultural space of their own

The multimedia textbook on the history of music *History of Russian Music: from Ancient Times to the Early 20th Century* can be used in various spheres of higher education.

First, it can be used for lectures on the history of Russian music, when the lecturer's narration is followed by presentations of audio and video clips, photographs, reproductions of paintings, etc. Multimedia slide show can accompany the given facts of the composer's life and work, it can illustrate information about the features of the musical culture of certain age, exemplify specific traits of certain musical trend, etc. The lecturer has an opportunity to select the necessary information

in accordance with the audience comprehension. The textbook can be used to introduce the new material as well as to fix and repeat the items already known to the students, it can be useful on practical classes, and at the stage of assessment of students' knowledge.

Students can work with the textbook in the classroom, the computers being provided, and independently as well. The textbook can be used:

- As a source of the following information: lectures, slides, audio and video clips, glossary, index of names, biographies of Russian composers
- As a visual aid: the variety of multimedia components helps to get acquainted with the numerous masterpieces of art, sculpture, architecture, as well as to listen to the episodes of musical compositions, to watch the fragments of operas and ballets
- To exercise in self-listening, and preparation to the quiz
- To write papers and reports
- To get ready for the practical classes
- To reduce the gaps of knowledge in the history of Russian music
- To estimate the level of knowledge

The textbook can be used in a classroom with one computer and a projector, as well as in the computer lab.

Use of multimedia provides additional possibilities of presenting information on the history of music:

- The possibility to see the visual details of paintings, historical documents, old editions of music, etc.
- The possibility to use hypertext and hypermedia links that simplifies the coordination of student's independent work
- Free navigation through the content, the direct access to the menu of the textbook

We began to introduce the multimedia textbook on the history of music in the 2009 at the music department of primary school faculty of Slovyansk (since 2012 Donbass) State Pedagogical University. The textbook was approved by the Ministry of Education and Science, Youth and Sport of Ukraine from 13.04.2012 № 1/11-5116 for production, classified as "Recommended by MESYS of Ukraine".

## 5 Conclusions

Information and communication technologies, especially computer multimedia, at the present stage of development of society is an important factor in improving the professional competence of future teachers. To possess skills of ICT use, to know the current multimedia publications (encyclopedias, web-resources developed for adults, and educational developing game programs for children, etc.), as well as to create their own e-products, and to apply them music teaching are essential conditions for becoming a really modern music teacher.

## References

1. Belyavina, N.: Pedagogical Conditions of Computer Technologies at Primary Stage of Music Education: Abstract of the thesis for Candidate degree. Kyiv State University of Culture and Arts, Kyiv (1999) (in Ukrainian)
2. Volynsky, V., Krasovsky, O., Chornous, O., Yakushina, T.: Structure and Content of Electronic Textbooks: Cognitive and Behavioral Aspects. In: Computer at School and Family, pp. 44–49 (2011) (in Ukrainian)
3. Huminska, O.: Multimedia Use as the Means of Innovation of Teaching Arts. *Art and Education* 3(53), 4(54), 21–25, 19–22 (2009) (in Ukrainian)
4. Mykhas'kova, M.: Formation of Professional Competence of the Future Teacher of Music: Abstract of the dissertation for obtaining the scientific degree of the candidate of pedagogical sciences in specialty 13.00.02 – theory and methodics of teaching music and musical upbringing. M.P. Dragomanov National Pedagogical University, Kyiv (2007) (in Ukrainian)
5. Novikova, N.: Computer Presentation on Music Lessons. *Art and Education* 3, 25–29 (2010) (in Ukrainian)
6. Poluboyarina, I.: Forming of the Professional Competence of Future Music Teachers at a Teachers' Training College. Abstract of the thesis on getting the scientific degree of a candidate of pedagogical sciences in specialty 13.00.04 – theory of professional education. State Ivan Franko University of Zhitomir, Zhitomir (2008) (in Ukrainian)
7. Stolyarchuk, L.: Computer Technologies Use on Music Lessons (in Ukrainian), <http://intkonf.org/stolyarchuk-li-vikoristannya-kompyuternih-tehnologiy-na-urokah-muziki/>
8. Tchaikovska, O.: The Formation of Junior School Students Music Knowledge by Means of Multimedia Training Techniques. Abstract of the dissertation on competing a scientific degree of the candidate of pedagogical sciences by specialty 13.00.02 – theory and methods of musical education. Kyiv National University of Culture and Arts, Kyiv (2002) (in Ukrainian)
9. Shtepa, V.: Computer Educational Programs for Music Lessons. *Computer for School and Family* 3, 33–37 (2011) (in Ukrainian)
10. Shcholokova, O.: Principles of Professional Artistic Training of Future Teacher Kyiv (1996) (in Ukrainian)



# General Disciplines and Tools for E-Learning Software Engineering

Ekaterina Lavrischeva and Alexei Ostrovski

Institute of Software Systems, NAS of Ukraine, Akademika Glushkova str.,  
40, Kiev, Ukraine

{lavryscheva,ostrovski.alex}@gmail.com

**Abstract.** This chapter presents new theoretical aspects of software engineering, which are oriented towards usage of technological lines for building applied systems and software product families from readymade reusable components. These aspects include: the theory of component programming; models of variability and interoperability; theory for building applied systems and SPFs using algebraic transformation of data types having different formats; principles for implementing certain theoretical aspects as lines for developing new components in the instrumental and technological complex; interoperability between programming systems and environments; developing structure of certain domains using VS.NET DSL Tools; product line-based experimental programs factory, developed by students at Kiev National University; approach to e-learning new theories and technologies, C#, Java, Visual Basic programming languages, and the “Software Engineering” discipline.

**Keywords:** software engineering, interoperability, variability, software industry, reusable components, applied systems, education, e-learning.

## 1 Introduction

The goal of the fundamental project for Institute of Software Systems (ISoftS), NAS of Ukraine, was to develop new scientific and applied aspects of software engineering, directed towards the advances in software product industry from the readymade software resources (reuses, aspects, services, etc.). We studied and took into account modern facilities and advances in the domain of software engineering, such as object-component programming, generative, compositional, and service-oriented programming [1–3], as well as peculiarities of modern operating environments and systems (Microsoft .NET, CORBA, Java, IBM, Eclipse, Protégé and others). This was done in order to implement the industrial technologies of software engineering on the basis of the reuse techniques, including engineering, economic, management, production disciplines and education. As a result of efforts, we elaborated the new theoretical foundation in producing applied systems (AS) and software product families (SPF), and developed the instrumental and technological complex (ITC) [4–6].

## 2 New Disciplines for Building Applied Systems from Reusable Components

Software engineering is a system of methods, means and disciplines for designing, developing, computing and supporting ready-to-adopt software. It provides means for the software development for multipurpose applied and informational systems.

The main principles of software engineering are productivity, industry, and quality, according to the corresponding body of knowledge – SWEBOK, which was developed in 2001–2004 by the international committee, formed by ACM and IEEE.

Ten areas of SWEBOK knowledge include the two directions:

1. Development: requirement engineering, designing, developing, testing, and maintaining software
2. Management: managing project, configuration, quality, methods and means of software engineering

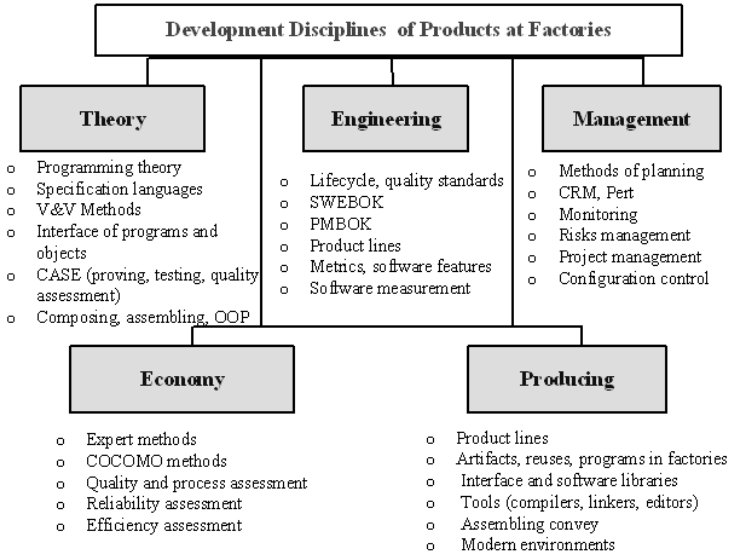
These areas are based on the lifecycle processes of ISO/IEC 12207 and 15539 standards.

SWEBOK does not include software engineering target objects (applied systems, software product families, domains, business applications), and technologies for developing such objects.

According to the approach towards teaching computer sciences in the United States, software engineering plays a prominent role in the informational disciplines. The area of software engineering comprises the entire hardware level, as well as technological and organizational levels of informational sciences. As a discipline, software engineering covers systematic software development of software system families, domains, and other scalable software projects. The main goal of software engineering is developing systematic models, reusable readymade components, and reliable methods for producing high-quality software products. This goal encompasses theories, concepts, paradigms, including both development structures for reliable computational software systems and management aspects of the designing process to meet customers' needs.

We proposed a new classification of software engineering disciplines (Fig. 1), which is necessary in industrial production of applied systems and software product families [6, 8]. The basic goals of SE disciplines according to this classification are as follows:

- *Scientific discipline* consists of the classic sciences (theory of algorithms, set theory, logic theory, proofs, and so on), lifecycle standards, theory of integration, theory of programming and the corresponding language tools for creating abstract models and architectures of the specified objects, etc.
- *Engineering discipline* is a set of technical means and methods for software development by using standard lifecycle models; software analysis methods; requirement, application and domain engineering with the help of product lines; software support, modification and adaptation to other platforms and environments



**Fig. 1.** Classification of SE disciplines

- *Management discipline* contains the general management theory, adapted to team-based software development, including job schedules and their supervision, risk management, software versioning and configuration
- *Economy discipline* is a collection of the expert, qualitative and quantitative evaluation methods and techniques of the interim artifacts and the final result of product lines, and the economic methods of calculating duration, size, quality, and cost of software development
- *Product discipline* consists of product lines, utilizing software resources (re-usable components, services, aspects, and so on), taken from libraries and Internet-based software repositories; it also contains assembling, configuring and assessing quality of software products

These disciplines are built on the basis of software engineering, modern approaches, and the scientific fundament; they are used in developing product lines and producing applied software products with these lines.

## 2.1 Software Industry in SE. New Concepts for Software Industry

Software industry uses modern methods, means and tools for mass production of software products. Kinds of software products include: systems, applied systems, distributed applied systems, business applications, real-time and on-board systems, etc. All of them, as a rule, are developed from readymade resources.

Basic principles of applied systems industry are as follows:

- Assembly line by V. Glushkov (1975), which uses readymade modules [5]
- Conveyor by K. Czarnecki and U. Eisenecker [11]
- Software factories by J. Greenfield, K. Short et al.
- Continuous integration by Martin Fowler
- EPAM assembly line (Belarus) for building various types of software, improving software quality and reducing risk
- Compositional programming by E. Lavrischeva for developing software products from reuses, services, and so on

The notion of assembly conveyor, originally proposed by Glushkov, has been elaborated for many years; nowadays we have created the experimental program factory at Kiev National University based on product lines for reusable components [9, 10]. Interfaces of these components contain a standardized description in a certain programming language, as well as a communication interface to enable interoperability with other objects.

## 2.2 Theory for Building Applied Systems Using Components

To model applied domains for business applications and applied systems with readymade components, the theory of compositional programming has been developed [5, 12]. Its main postulates are as follows.

**Proposition 1.** *The applied domain, which is modeled with a set of components  $C$ , is a component itself.*

**Proposition 2.** *The applied domain that is modeled may be a separate component  $C$  in another domain.*

Based on the theory of sets, if  $C = \{C_0, C_1, \dots, C_n\}$  is the set of components and  $C_0$  corresponds to the whole domain, then the following equation holds for  $C$  elements:

$$\forall i \quad (i > 0) \rightarrow (C_i \in C_0). \quad (1)$$

Each component is represented as an element in the set. In this case, the expression (I) is transformed into

$$\forall i \exists j \quad (i > 0) \wedge (j \geq 0) \wedge (i \neq j) \wedge (C_i \in C_j). \quad (2)$$

For each component of the set  $C$  (except  $C_0$ ), algebraic operations from the set theory may be applied, such as the “part-whole” relation, aggregation, and so on.

**Proposition 3.** *If  $C = \{C_1, C_2, \dots, C_n\}$  is a set of components, and  $P = \{P_1, P_2, \dots, P_r\}$  is a set of unary predicates associated with the properties of elements of  $C$ , then the component  $C_i$  corresponds to a set of statements and predicates of  $P$ , and for a certain set of pairs  $(i, k)$ ,  $P_k(C_i) = \text{true}$ .*

Building component-based applied systems and SPFs from the elements of these sets is performed by assembling reusable components using their interface data, which corresponds to the defined properties and description of the  $C$  elements, stored in interface and implementation repositories.

**Definition 1.** *Component is understood by us as a PL-independent self-reliant software product that provides execution for a certain set of applied functions and services of the applied domain, which can be only accessed with remote calls.*

Component model is a corollary of generic typical solutions or functions for the applied domain, which are represented by the component and its interface in the following form:

$$C = (CNa, CIn, CFa, CIm, CSe), \quad (3)$$

where

- $CNa$  is a name of the component
- $CIn = CIn^{(in)} \cup CIn^{(out)}$  is a set of input and output interfaces
- $CFa$  is a set of methods and instances of components
- $CIm$  is the interface implementation
- $CSe$  is the system service

Input interfaces  $CIn^{(in)}$  correspond to the component implementation, while output interfaces  $CIn^{(out)}$  match another component from the set  $C$ . Input interface  $CIn^i \in CIn$  has the model

$$CIn^i = (InNa^i, InFu^i, InSp^i), \quad (4)$$

while interface implementation  $CIm^i \in CIm$  has the following model:

$$CIm^i = (ImNa^i, ImFu^i, ImSp^i), \quad (5)$$

where suffix  $\cdot Na$  denotes the name of an interface or an implementation,  $\cdot Fu$  denotes a list of its functions, and  $\cdot Sp$  is its specification.

The necessary condition for the component  $C_j \in C$  to exist is its integrity:

$$\forall CIn^i \in CIn \exists CIm^j \in CIm \quad \text{Pr}(CIn^i) \subseteq CIm^j, \quad (6)$$

where  $\text{Pr}(CIn^i)$  denotes functionality that provides implementation of the interface methods from  $CIn^i$ .

**Proposition 4.** *To assemble two heterogeneous components  $C_1$  and  $C_2$ , the following condition must be satisfied: if  $CIn_1^i \in CIn_1^{(out)}$ , then there must exist  $CIn_2^k \in CIn_2^{(in)}$  that satisfies*

$$(\text{Sign}(CIn_1^i) = \text{Sign}(CIn_2^k)) \wedge (\text{Pr}(CIn_1^i) \subseteq CIm_2^j), \quad (7)$$

where  $\text{Sign}(\cdot)$  is the signature of the corresponding interface.

The model of the component environment has the following form:

$$CE = (CNa, InRep, ImRep, CSe, CSeIm), \quad (8)$$

where

- $CNa = \{CNa^m\}$  is a set of component names
- $InRep = \{InRep^i\}$  is the repository for component interfaces
- $ImRep = \{ImRep^j\}$  is the repository for component implementations
- $CSe = \{CSe^r\}$  is the system services interface
- $CSeIm = \{CSeIm^r\}$  is a set of implementations of system services

Each element in  $InRep$  is described by the pair  $(CIn^i, CNa^m)$ , with  $CIn^i$  the component interface (4),  $CNa^m$  the name of the implementing component. By analogy, each element in  $ImRep$  is described by  $(CIm^j, CNa^m)$ , where  $CIm^j$  is the implementation interface (5).

These theoretical statements, as well as the theory for transforming data types in heterogeneous components [12, 19, 20], are implemented in the component environment of the ITC [14].

### 2.3 Reusable Components

The component model for component-based development has the following specification [6]:

$$C = (T, I, F, Im, S), \quad (9)$$

where  $T$  denotes type,  $I$  – interface,  $F$  – functionality,  $Im$  – implementation,  $S$  – interoperability service for work with other components and the environment.

Basic operations upon components are:

- Specifying components and their interfaces (pre- and post-conditions, which must be satisfied by caller components) in such languages as IDL, API, WSDL, etc.
- Maintenance of components and reuses in the component repository for their future integration in AS
- Application engineering, domain engineering, software product family engineering, executing lifecycle processes for applied systems, etc.

### 2.4 Notion of Technological Lines

Technological line (TL) is created at the stage of technological preparation, which precedes the production of artifacts and includes designing and developing the architecture of the line. Processing components of the applied system is then performed using the corresponding technological modules or programming systems [20].

The main requirement imposed on the technological line development is thorough selection of lifecycle processes, standard tools of the operational environment, as well as developing a certain set of normative documents. Bearing in mind the specifics of the chosen applied domain, one can choose appropriate reusable components, generative and implementation tools, as well as the route for the product line processes.

## 2.5 Notion of Product Lines

Product line and software product family are defined in the ISO/IEC FDIS 24765:2009(E) – Systems and Software Engineering Vocabulary as “a group of products or services having the common manageable set of properties which meets the requirements of a certain market segment”.

Models for representing processes of SPF development, according to Software Engineering Institute, are the engineering and the process model.

*The engineering model* is created at SEI and corresponds to the three-step production:

- Developing reusable components
- Merging them into software systems
- Managing components and applied systems

The development process encompasses defining the SPF domain, designing the manufacturing process for a set of components, allowing for its context of use, restrictions and production strategy. The merge process includes designing implementations of each applied system on the base of the manufactured resources and reusable components. The management process is oriented towards process coordination.

*The process model* sorts out a set of processes that run on the two levels: domain engineering level, which is also referred to as the development “for reuse”, and applied systems engineering level, referred to as the development “with reuse”. Assembly lines use readymade components, which improve the time of development and are able to support the whole manufacturing cycle according to the specific requirements and needs. Product lines for manufacturing certain elements of software products are developed within the fundamental project for ISoftS, NAS Ukraine, and implemented in the instrumental and technological complex [14], as a web site (<http://sestudy.edu-ua.net>). ITC is oriented towards developing variable and interoperable components, applied systems, and SPFs.

## 2.6 Concept of SPF Variability

**Definition 2.** *Variability is the ability of a software system or artifact in software product line to be extended, changed, customized or configured for use in a specific context with the proper quality characteristics to mitigate its current limitations.*

Variability serves to increase reusability and sensitivity of application to the changes in business processes and environmental conditions by expanding the context of its applicability. At the same time it increases the complexity of the integrated development environment (IDE) and, consequently, the duration and cost of application production cycle. Compliance with the balance between expected benefits and costs of implementation leads to variability assessment and management in IDE [15, 21].

**Definition 3.** *Variability model in SPF is a pair of mutually agreed models:*

$$VM = (SV, AV), \quad (10)$$

where  $SV$  is the SPF structure model;  $AV$  is the development process assets model.

$VM$  model is used to:

- increase the variability level in SPFs continuously
- provide variants of variability in SPF products
- decrease cost and time spent on producing applications

Variability includes management processes for SPF versions for:

- imposing requirements on the SPF by solving scheduling tasks
- adapting to the new operating conditions and changing certain functions and programs
- widening the context of usage for applied systems, when processes or components of the object domain are changing

Basics of variability are mechanisms (model, metrics, and management process), which result in optimization of the structure of software systems built using reusable components.

## 2.7 Interoperability between Programs, Systems, and Environments

**Definition 4.** *Interoperability is the ability of components or systems to interact with each other and exchange common information.*

Formally, interoperability model is understood as the representation of parameters and relationships between different components of software or informational systems. The model reflects the relationship system and the designing process for software product development. The relationships may be described with mathematical means, such as abstract algebras, set theory, and so on [18–22].

Migration of connective components and systems into new environments is based on using interfaces and network communication protocols. Theoretical foundation of system interoperability is based on the OSI standard model and includes:

- Model and mechanisms for describing system interfaces
- Mechanisms and operations for transmitting data via networks from local and global storages in heterogeneous environments

System interconnection model  $M_{\text{inter}}$  covers interconnection between systems, developed in a certain heterogeneous environment, to another one, extending their limits [19]. It has the following generic structure:

$$M_{\text{inter}} = \{M_{\text{pro}}, M_{\text{sys}}, M_{\text{env}}\}, \quad (11)$$

where



- $M_{pro} = (C, Int, Pr)$  is a program model,  $C$  being a component,  $Int$  – an interface,  $Pr$  – a program;
- $M_{sys} = (SS, Int, Prot)$  is a software system model for the system  $SS$ ,  $Int$  is an interface,  $Prot$  is the data transmission protocol;
- $M_{env} = (Env, Int, Prot)$  is an environment model, in which  $Int$ ,  $Prot$  contain the set of external interfaces and remote calls that transmit data between programs via networks.

The basic parameters for the interconnection model  $M_{inter}$  are program, interface and message or protocol. Figure 2 represents the data flow between the modern programming environments.

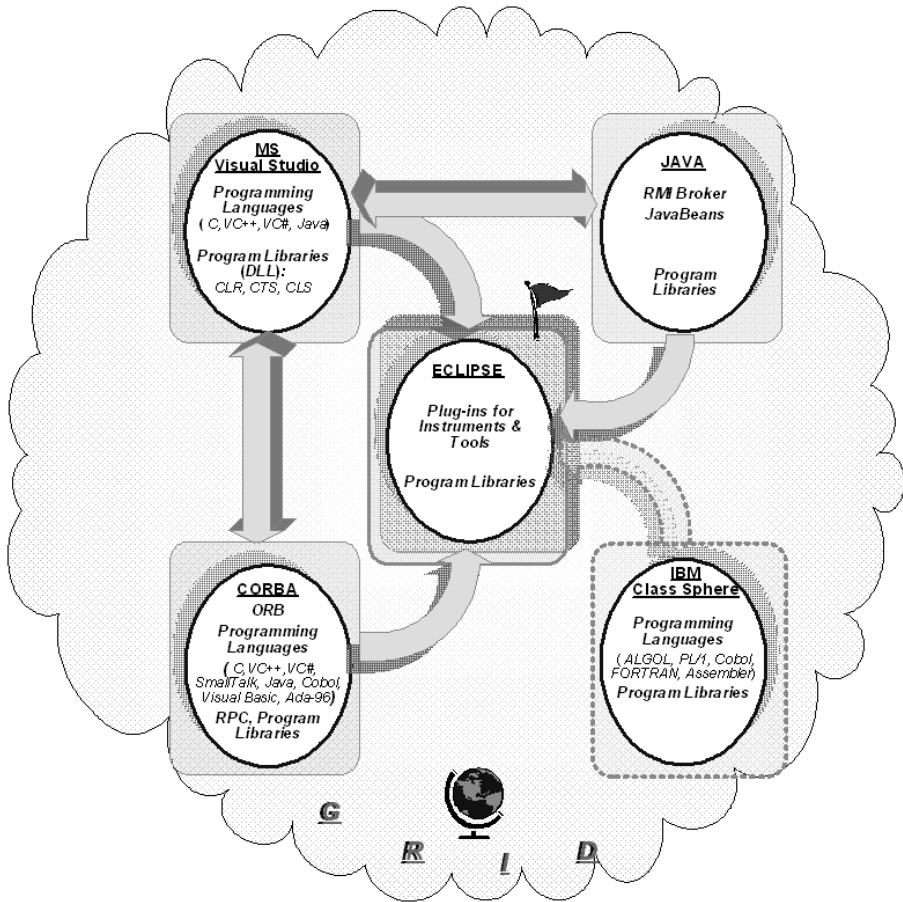


Fig. 2. Interconnection between modern programming environments

The applied models for system interconnection via Eclipse IDE implement three relationships: VS.NET ↔ Eclipse, CORBA ↔ Java, and IBM Web Sphere ↔ Eclipse.

### 3 Structure of the ITC

The instrumental and technological complex is primarily designed for learning basic aspects of software engineering:

- Product lines for manufacturing applied systems from heterogeneous reusable components
- Techniques for assembling readymade components from repositories into new software products using technological lines
- Means and tools for supplying interoperability between systems and environments
- Technological methods, verification, testing and assessment, described in the e-textbook “Software Engineering” [6]
- Designing ontology-based models
- Measuring the characteristics of quality, cost, and capacity

One of the core aspects of the complex is its technological lines, which automate the following processes:

- Development of components, reuses and services and their maintenance in Internet repositories
- Using DSLs for creating domain descriptions (for software lifecycle and computational geometry) with Eclipse-DSL environment and Protégé
- Configuring reuses and systems into SPFs with Workflows tool in VS.NET
- Translating general data types (GDT) into fundamental data types (FDT) according to ISO/IEC 11404 standard, marshalling data and transforming nonequivalent data in programming environments
- Programming techniques in C#, Visual Basic and other programming languages
- Using service tools within the complex

The scientifically important results of our long-term research and development within the complex are as follows:

- Software development, adaptation theory, generative programming [1]; a new ideology, models (interoperability, variability, livability), and methods for supplying correctness (expert, V&V, metric analysis, testing) [4]
- Developing product lines from repository components and services; developing applied systems from multi-language reuses [5]; configuring AS and domains; methods of supplying correctness (V&V); assessing software systems (quality, cost, capacity, reliability) [6]
- Software development using the programs factory (<http://programsfactory.univ.kiev.ua>) at Shevchenko Kiev National University [9, 10]
- Methods and means described in the electronic textbook by E. Lavrishcheva, available on <http://www.intuit.ru>

Judging by the list of several criteria, such as the wide coverage of various software engineering topics and supplying product lines with detailed descriptions and examples, the ITC has no known counterparts with free access in the ex-USSR domain of the Internet. The introduced modular architecture of the complex allows widening its functionality with ease by adding new product lines or elaborating the existing ones.

## 4 Approach to E-Learning Basics of Software Engineering

Teaching students the aspects of industry at Ukrainian universities is currently at its initial stage; to solve some education problems, we have introduced a new approach to e-learning students various aspects of SE, which assists in acquiring knowledge on software industry in general. We suggest learning foundation of SE with the help of the ITC web site.

### 4.1 Functions and Structure of the ITC Web Site

The site in question was developed as a collection of tools for software engineering and at the same time was displayed during lectures on SE at Kiev National University. This drove the authors to orient the complex towards teaching students and graduate students the basics of software engineering, including various tools and means of its support, along the following aspects: developing programs, reusable components, and services; assembling software systems and their families from reusable components, as well as developing, generating, interoperability, and ontological modeling techniques for object domains (software lifecycle, computing geometry, etc.).

Taking the above into account, we have chosen a strategy of teaching various aspects of industry-compliant SE. In order to gradually and consistently implement this strategy within the ITC, we utilized the Internet-based methods and modern programming systems that support different aspects of software development, namely:

- Protégé system to model object domain ontologies
- Eclipse as a tool to embed different programming and system components into the ITC by using its plug-ins
- Microsoft Visual Studio .NET as a multifunctional tool to organize team development of the new systems, including developing software via Internet using various programming languages, OOP, UML, and cloud computing frameworks, such as Azure, SkyDriven, Amazon, etc.
- CORBA system that has a universal object request broker providing interoperability between programs, written in different languages, by using stub/skeleton mechanisms
- Eclipse-DSL and Microsoft DSL Tools to support DSL tools with a graphical user interface for designing systems, domains, applications, and software product families

The start page of the web site features a list of implemented sections and subsections concerning software engineering. The sections in question are: Main Page, Technologies, Interoperability, Tools, Presentations, and Learning (<http://www.sestudy.edu-ua.net>), Fig. 3).

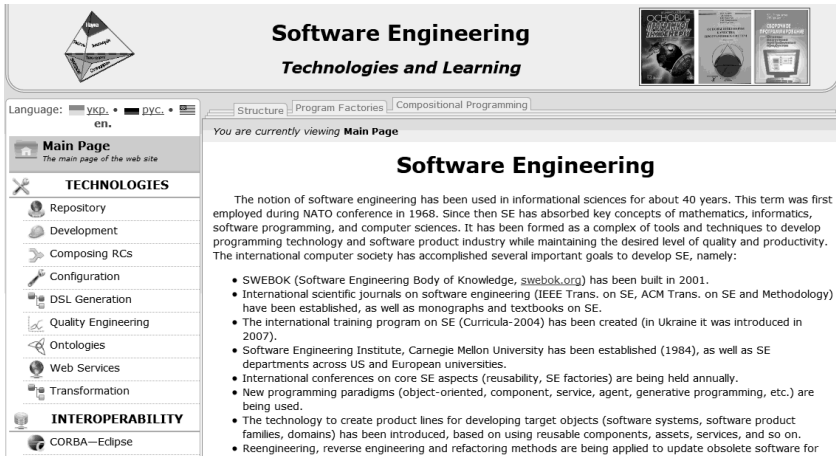


Fig. 3. Main page of the ITC web site

Each section contains subsections with keywords that specify the names of product lines (10 altogether). The keywords, divided into the aforementioned main categories, are as follows:

1. Technologies
  - (a) Component repository
  - (b) Component development
  - (c) Composing reuses
  - (d) Component configuration
  - (e) DSL generation
  - (f) Quality engineering
  - (g) Ontologies
  - (h) Web services
  - (i) Data types transformation
2. Interoperability
  - (a) CORBA ↔ Eclipse
  - (b) Visual Studio .NET ↔ Eclipse
  - (c) Visual Basic ↔ Visual C++
3. Tools
  - (a) Eclipse
  - (b) Protégé
4. Presentations
  - (a) Applied systems

- (b) Software engineering and program factories
  - (c) Software industry
5. Education
- (a) C# and Microsoft .NET framework
  - (b) Java
  - (c) Software engineering

All sections and subsections include standardized pages, such as an overall theoretical description, an example that illustrates the concerned topic (developed with one of the workbench programming environments, in most cases), a thorough description of the example, and so on. Each page displaying an article on one of the topics of the complex is built using the same template that contains the following main components:

1. The unified header, which contains the site banner and the title
2. The current location string
3. The main menu including the language panel and links for navigation
4. The navigation panel, which contains links to various subsections of the current section
5. The content of the article
6. The footer that includes information about the site authors and developers

During the course of choosing product lines for their inclusion into the complex, the following main criteria were taken into account (with the order reflecting their priority, from the most important to the least significant ones):

1. Relevance of the topic in question within the software engineering discipline, as well as its applicability for solving present-day real-world problems
2. Theoretical basis supporting the technology
3. Relations between the technology and techniques behind other product lines
4. Simplicity and accessibility of both theoretical and applied aspects of the technology for a sufficiently wide audience, including students and lecturers of Ukrainian universities
5. Availability of an illustrative example to explain the main concepts behind the technology in question, preferably with applicability to certain real-world problems

Employees of the software engineering department at ISoftS and students of KNU and MIPT in course of writing their thematic and graduate papers implemented the web site and several product lines of software development on the basis of readymade resources and components. Particularly, they have developed an experimental program factory, means of interoperability support between programs and systems, a domain description in DSL using Protégé environment, and an applied system for registering academic missions in the institutes of NAS of Ukraine.

## 4.2 E-Learning Concept of Technological Lines

The ITC web site features several basic technological lines for component development, namely:

- The program factory, which contains the specification of reusable components and courses on basic MS .NET programming and software engineering, and the students' program factory, developed at the cybernetics department of Kiev National University
- The repository of reusable components, which is an integral part of the aforementioned factory
- Assembling multi-language programs and components into a software system by converting incompatible data types
- Configuring reusable components in a system with complex structure that possesses points of possible modifications in some subprograms according to customer's needs (variability points), designed with MS .NET Workflow environment
- Describing applied domains in DSL with an example of the lifecycle domain with graphical and textual representations, created with Eclipse-DSL environment
- Quality and cost engineering with the help of Softest application, designed to estimate labor expenditures and the cost of software development
- Designing domain ontology with Protégé environment, with an example of the applied domain of computational geometry
- Constructing software product families by merging components that use different programming platforms with the help of web services
- Translating general and fundamental data types (GDT and FDT) according to ISO/IEC 11404 standard and GRID system programming practices, by example of the primitive library
- Generating software resources and merging them into programs, software products, and their families with the configurator, as specified by the variability model
- Testing programs in order to obtain a correct software product and to collect data about faults and errors, required in assessing its operational reliability

These lines are intended primarily for e-learning, but nevertheless are capable to be used as building blocks in program factories development.

## 4.3 E-Learning Concept of Interoperability

The principles of interaction between systems and environments were studied by us by examining conditions and capabilities of modern operating systems and development environments (VS.NET, Java, CORBA, Eclipse; see Fig. 2). The objects of the study were components, applied systems, and SPFs. The main question of system adaptation across different environments was transforming different formats in the application data and usage of compiled code. In their master works, students of KNU and MIPT proposed applied interoperability

models and implemented them for pairs of systems: Visual Basic and Visual C++, Java and Microsoft .NET, Microsoft Visual Studio and Eclipse.

Parameters of the interconnection model are used to exchange messages between systems and environments. Interaction between systems is based on the notion of interface, which is specified in IDL and contains definitions of data types and remote call operations.

The ITC web site features three examples of interoperability between systems, which are used to demonstrate different levels of interconnection in software products:

- Interoperability between programs created in Visual Basic and Visual C++, provided by the interface layer in form of a library, which transmits data from one program to another and transforms incompatible data types, when necessary
- Interoperability between Java and Microsoft .NET programming platforms, implemented by utilizing the CORBA object request broker and using interface definition language (IDL) to describe interfaces in these platforms
- Interoperability between Microsoft Visual Studio and Eclipse integrated development environments, provided by transmitting application data of a program, developed with Visual Studio, into the Eclipse repository, utilizing Eclipse plug-in capabilities

#### 4.4 KNU Students' Experimental Program Factory

During learning SE disciplines and informational systems [7] in universities, students perform scientific and applied work to develop their first programs or new artifacts. Sometimes their knowledge and know-hows will be required by other students. Therefore their artifacts may be used by others in solving different problems, associated with a similar artifact, as well as in development of new applied systems. Author, Prof. Lavrischeva has proposed to create an experimental programs factory to accumulate knowledge about students' artifacts and programs.

To allow students participate in the development of research artifacts for mass use, the web site <http://programsfactory.univ.kiev.ua> has been developed (Fig. 4). The site is a part of the 'Technologies' line of the ITC. To develop reusable components and programs, technological lines 1, 2, 3, 4 have been built. These technological lines are:

- Learning C# programming language in Visual Studio .NET environment
- Selecting reusable components from the repository to meet market demands in special-purpose software
- Configuring components to create complex program structures
- Basic fragments of knowledge (knowledge domains) concerning SE discipline for students provided with a dedicated e-textbook by one of authors, Prof. E. Lavrischeva

The line for repository maintenance includes mechanisms for providing uniform documentation of stored components using their interfaces, and various tools for

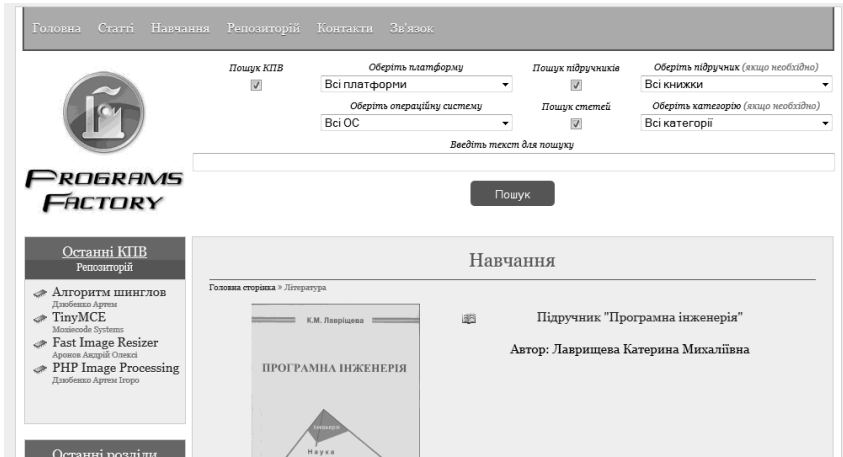


Fig. 4. Main page of the programs factory web site

readymade components and programs. Technological lines of the ITC are used for quality assessment of components or programs, as well as cost assessment using COCOMO model. The data obtained with these estimations and the general description of the component are the elements of the certificate of software products.

The line for building applied systems at the factory includes tools for engineering specific required components with various programming languages, programs and artifacts.

The factory has been fully implemented; since January 2011 it was visited by more than 4000 users.

## 5 Conclusion

We have implemented the following theoretical foundations of applied systems development, necessary for teaching students:

- Classification of disciplines in SE and software industry
- Concepts for developing AS lines based on conveyor principles
- New models of interoperability and variability
- Configuring components, testing, evaluation of quality and cost of the product
- Software development in C#, Java, Basic programming languages with Visual Studio .NET and Eclipse
- Ontological representation of knowledge on new disciplines for e-learning (software lifecycle and computational geometry)
- ITC complex, supported by the two web sites: <http://sestudy.edu-ua.net> and <http://programsfactory.univ.kiev.ua>



## References

1. Czarnecki, K., Eisenecker, U.: *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Boston (2000)
2. Greenfield, J., Short, K., Cook, S., Kent, S.: *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley, Hoboken (2004)
3. Duval, P., Matyas, S., Glover, A.: *Continuous integration. Improving Software Quality and Reducing Risk*. Addison-Wesley (2007)
4. Lavrischeva, E., Koval, G., Babenko, L., Slabospitska, O., Ignatenko, P.: *New Theoretical Foundations of Production Methods of Software Systems in Generative Programming Context*. Electronic monograph. In: UK 2011, vol. 67. *Akademperiodika*, Kiev (2011) (in Ukrainian)
5. Lavrischeva, E., Grischenko, V.: *Assembly Programming*, 2nd edn. *Basics of Software Industry*. Naukova Dumka, Kiev (2009) (in Russian)
6. Lavrischeva, E.: *Software Engineering*. *Akademperiodika*, Kiev (2008) (in Ukrainian)
7. Lavrischeva, E.: *Cybernetics, informatics and SE: evolution aspects*. In: *Problems in Programming*, vol. 1, pp. 3–14. *Akademperiodika*, Kiev (2010) (in Russian)
8. Lavrischeva, E.: *Classification of Software Engineering Disciplines*. *Cybernetics and Systems Analysis* 44(6), 791–796 (2008)
9. Anisimov, A., Lavrischeva, E., Shevchenko, V.: *On Scientific Software Industry*. Technical report, Conf. Theoretical and Applied Aspects of Cybernetics (2011) (in Ukrainian)
10. Aronov, A., Dzubenko, A.: *Approach to Development of the Students' Program Factory*. In: *Problems in Programming*, vol. 3, pp. 42–49. *Akademperiodika*, Kiev (2011) (in Ukrainian)
11. *Framework for Software Product Line Practice*, version 5, <http://www.sei.cmu.edu/productlines/index.html>
12. Lavrischeva, E.: *Formation and Development of the Modular-Component Software Engineering in Ukraine*, 31 p. Glushkov Institute of Cybernetics, Kiev (2008)
13. Andon, P., Lavrischeva, E.: *Development of Program Factories in the Informational World*. In: *Bulletin of NAS of Ukraine*, vol. 10, pp. 15–41. *Akademperiodika*, Kiev (2010)
14. Lavrischeva, E.: *Instrumental and Technological Complex for Developing and Learning Aspects of Software System Development*. In: *Bulletin of NAS of Ukraine*, vol. 3, pp. 17–27. *Akademperiodika*, Kiev (2012) (in Ukrainian)
15. Lavrischeva, E., Slabospitska, O., Koval, G., Kolesnik, A.: *Theoretical Aspects of Variability Management in Software Product Families*. In: *KNU Bulletin. Physics and Mathematics Series*, vol. 1, pp. 151–158. KNU, Kiev (2011) (in Ukrainian)
16. Ostrovski, A.: *Approach to Interconnection Support between Java and MS.NET Programming Environments*. In: *Problems in Programming*, vol. 2, pp. 37–44. *Akademperiodika*, Kiev (2011) (in Russian)
17. Radetskyi, I.: *One of Approaches to Maintenance Interconnection Environments Visual Studio and Eclipse*. In: *Problems in Programming*, vol. 2, pp. 45–52. *Akademperiodika*, Kiev (2011) (in Ukrainian)
18. Lavrischeva, E.: *Generative Programming of Software Products and Their Families*. In: *Problems in Programming*, vol. 1, pp. 3–16. *Akademperiodika*, Kiev (2009) (in Ukrainian)
19. Lavrischeva, E.: *Interaction Models of Programs, Systems, and Operational Environments*. In: *Problems in Programming*, vol. 3, pp. 13–24. *Akademperiodika*, Kiev (2011) (in Ukrainian)

20. Lavrischeva, E.: Concept of Scientific Software Industry and Approach to Calculation of Scientific Problems. In: Problems in Programming, vol. 1, pp. 3–17. Akadempriodika, Kiev (2011) (in Ukrainian)
21. Lavrischeva, E.: Problem of Interoperability between Heterogeneous Objects, Components, and Systems. Approach to Solve It. In: 7th International Programming Conference, UkrProg 2010, pp. 28–41. Akadempriodika, Kiev (2010)
22. Lavrischeva, E., Ostrovski, A., Radetskyi, I.: Approach to E-Learning Fundamental Aspects of Software Engineering. In: Proceedings of ICTERI 2012, Aachen. CEUR Workshop Proceedings, pp. 176–187 (2012)

# Formation of Digital Competence of Future Teachers of Elementary School

Nataliya Kushnir and Anna Manzhula

Kherson State University, 27, 40 roktiv Zhovtnya St., Kherson, Ukraine 73000  
kushnir@ksu.ks.ua, ilovetrees@mail.ru

**Abstract.** The chapter presents the experience of forming the ICT-competencies of future teachers of elementary school and early education. The course is developed on the Moodle platform for distance education. The course objective is training of future teachers to use ICT in their pedagogical activities, understanding the role of ICT, its opportunities and perspectives for improving educational process.

**Keywords:** ICT-competencies, future teachers of elementary school, distance learning, Moodle.

## 1 Introduction

There is no doubt that it is the first time in the history of human kind when the generation of things, processes and ideas changes faster than generation of people [1]. All these changes were caused by the transmission and processing of large amounts of information by experts in a short time.

The world has changed so much since the 70th years of the XX century, that now we face completely different reality. We live in the overloaded of information world, it requires new skills, new lifestyle and new vision; information society presents fundamentally new requirements to the entire education system in particular to the training of teachers.

## 2 Related Work

Among the trends of society that significantly affect education it is necessary to emphasize the active development of mobile technologies, the creation of open electronic content, the appearance of virtual educational game technologies, the usage of social networks for studying and so on.

Nowadays informal education outstrips the formal one. Because we (parents and teachers) cannot prevent or stop the active usage of ICT by children, we must lead and guide this process for children of all ages. "... it is not necessary any more to prove that ICT matters in early childhood education. New digital technologies have entered every aspect of our reality, including families and lives of young people. They have already affected preschool children's play and learning" [2].

Modern children were born in the century of digital technologies and perceive information in rather different way. Mark Prensky called them digital natives [3]. Neil How & William Shtrauss in their paper Theory of generations described this phenomenon as Generation Y. At the same time majority of modern teachers are digital emigrants so such imbalance prevents the development of adequate interaction between teachers and children [4].

Alan Kay in 1968 described the Dynabook concept as a portable interactive personal computer, as accessible as a book'. The Dynabook would be linked to a network and offer users a synthesis of text, visuals, animation and audio. Kay drew an initial pen and ink sketch of this device, which is widely considered the prototype for the notebook computer, but in fact is much closer to today's iPad [2].

Today, people can use massive and free internet services that allow them creation of their own content, freely sharing it, and active participation in the communities and social networks. Most students already use these opportunities while the process of education at schools still remains traditional. Moreover, some internet services, such as social networks are not supported there. As a result, there is a gap between the level of communication and presentation of information on Social Web and at school, where the educational process excluding home assignments is usually interrupted as soon as the student leaves the class.

Informatics education, due to its relationship to technical devices, is bound to act swiftly in response to societal trends. To meet modern requirements and needs of students and their parents schools have to overcome merging gaps and provide more personalized process of education that focuses on student's individuality and ensure continuity of studying.

It's known that changes in educational system follow social and economical ones. A new Ukrainian National standard of Primary education that was approved in April 2011 has already included Informatics as required course for all pupils starting from the second year.

The goal of Educational system is therefore to facilitate increasing the quantity of digital natives among the teachers. An important part of teacher's skills is related to the competences and abilities to design, develop, conduct, facilitate and access teaching and learning processes aimed at acquisition of productive "soft skills" enhanced by Information and Communication Technologies. These skills include: knowledge presentation, working on projects, problem solving, and communication skills. The term "ICT-enhanced skills" is a concept coined to denote the synergy between soft skills and ICT skills [5].

Changing the level of teachers' ICT-skills is so important that it become a point of attention of many large commercial and noncommercial organizations. UNESCO worked out ICT-competency Standards for Teachers [6]. The company Microsoft realizes project Partnership in Education, the company Intel develops the project Education for Future and so on. As a result of the project Innovative Teacher (I \* Teach) the most essential skills were identified for teachers in the participating countries, such as Bulgaria, Germany, Italy, Lithuania, Netherlands, Poland and Romania (as ICT-enhanced skills for which there was a biggest need in the countries involved):

- Searching and selecting information
- Presenting information
- Working on a project
- Working in a team

Similar skills are the most essential either for Ukrainian school teachers.

Unfortunately, the system of education is quite inert. Despite the fact a new Ukrainian national standard was approved in April last year there weren't done any changes in the syllabus of future teachers' training till the 1st of September, 2012.

Therefore, the authors of this chapter have made an attempt within former discipline "New Information Technologies and Technical Facilities of Education" (NIT and TFE) to develop a new curriculum and organize educational process based on the usage of distance learning technologies which helps build ICT skills and digital competence of future teachers.

Digital competence means confident usage of electronic media for work, leisure and communication. Within this course is focused on the usage of technologies in educational process and in future professional activities. This competence is based on logical thinking, the high level of proficiency in information management and highly developed communication skills. At a basic level of ICT skills includes utility of modern IT (computers, multimedia, Internet, electronic media, ATMs, mobile phones, etc.) for search, access, storage, production, presentation and exchange of information, as well as communication between people and work on the Internet [1].

Other authors use the term "digital literacy" that includes the following abilities [7]:

- Using ICT skills to create and share information
- Searching, sifting, scanning, and sorting information
- Navigating through screens of information
- Locating and evaluating information
- Using ICT to research and solve problems
- Making multimedia presentations
- Retrieving, organizing, managing, and creating information
- Sending and receiving messages

The clusters of students' skills that are need in the digital age of the 21<sup>st</sup> century are described by different terms as digital literacy (or in the plural digital literacies), e-literacy, new literacies, screen literacy, multimedia literacy, information literacy, ICT literacies [7].

If we analyze the employers' requirements, the most essential skills for them are developed analytical thinking, communication skills, the willingness to experiment, the ability to work in teams and so on. Of course, a teacher who doesn't have these skills will not be able to help students to form them. Formation of Digital competence requires expansion of didactic goals of typical tasks for students. Therefore a new goal "forming ICT-competencies" emerges beside the traditional goal "learning concrete material by students".

We asked ourselves the following questions before upgrading the course:

Who will teach Informatics at Primary school?

What will be simpler and more effective:

- empowering a teacher of primary school knowledge and skills in Informatics; or
- empowering a teacher of Informatics knowledge and skills in Didactics and Psychology of children

To answer these questions let us remember Ukrainian experience that is similar to the situation with adding Informatics as required discipline in Primary school. Ten years ago foreign languages has become a required course for all pupils starting from the second year. It showed that a teacher is used to work with high school children and doesn't know Didactics and best practices of working with 6-10 year old children. As a result many children had negative attitude to this discipline. From our point of view the level of professional competence of primary school teachers, particularly in the area of ICT, should be such that enable them to teach computer science.

And the next common question: What should be learnt by future teachers in ICT field and how can we realize these didactic purposes?

At the Conference "Informatics in Secondary School - Evolution and Perspectives" (ISSEP, 2005 in Klagenfurt, Austria and in 2006 in Vilnius, Lithuania) questions about what exactly teachers need to be trained in terms of ICT proficiency were discussed. The one of the topic attracted much attention was about European Computer Driving License (ECDL) that had been propagated since the late 1990s and implemented at schools at this time already on a broad scale. Teachers, parents, as well as pupils had welcomed such approach to education in ICT. Firstly, teachers had material that had a clear scope and was easy to teach and examine. Secondly, parents were sure that their children learn "modern and relevant staff", moreover, work with computers was attractive to kids. Some colleagues already proposed to weave conceptual knowledge in the strictly application-focused instruction of how to handle computers; and how to handle widely used general application software [5].

We have experience in introducing ECDL at our training center at Kherson State University that has been established as the international project (Tempus/Tacis Joint European Project "ECDL4UA"). In the research reported in this chapter we used this background experience and extended it on educational process of future teachers of primary school [8]. According to ECDL students must be familiar with the basic concepts of information technologies and must be able to use a personal computer and basic applications, but the study of concrete applications is not so important now. It's consequence of the fact that people started using the Internet not only as professional resource but also privately as an information resource as well as communication media with relatives and friends. There are new free services that allow each user to create own space on the network and fill it with their content. E-learning becomes more relevant and is positioned as "learning for future", but there is a lack of sufficiently competent educators for e-learning. It was initially pushed into the domain of those teachers who can handle computers, i.e., teachers of informatics [5]. However, every teacher needs to know features that modern information and communication technologies provide for educational process, which of them are appropriate in a

particular situation, teachers need to have the skills of mastering new programs or services by themselves. It is important for teachers to understand the necessity of improving their ICT skills during the life and have a desire to do it.

There is a world tendency of shifting teacher's role from a source of knowledge to facilitator and ICT can be core means for this process [10]. The approval of a new educational standard of elementary school, which includes Informatics as required course for all pupils starting from the second year, led to increasing the priority of the course NIT and TFE for professional training of future elementary school teachers.

Now lecturers of this course observe such tendency: students use ICT more often and have better skills of working with different application in comparison with the previous years. The results of analyzing of data, gotten from the entrance poll, have confirmed it. Really, there is no sense to repeat the school course of Informatics: 89% of students have their own computer or laptop; 70% of respondents have free access to the Internet outside university; 78% of students have stated they are skilled in using Office programs, Internet, printers, scanners; 51% of respondents have experience of self-reliant learning of software. 49% of students feel comfortable with working at the computer, 33% – feel confident [13]. So, there is a need to shift the emphasize from the learning keystrokes in Informatics to understanding ICT creative potential for teaching and applying it in future pedagogical activities. As we have seen before the authors identified a need to investigate how goals and tasks, content, methods and forms of teaching this course should be appropriately changed, peculiarities of its realization and analysis of received results.

The course “NIT and TFE” is the only ICT-related discipline for undergraduate students in our Computer Science Program., except students studying at specialization “Basics of Informatics”. The course “NIT and TFE” is a base for studying the discipline “New information technologies in preschool education” (8 hours of lectures and 12 hours of practical classes) for undergraduate students of the specialty “Preschool education”. The overall objective of these disciplines is to train future teachers to use ICT in their pedagogical activities, to understand the role of ICT, its opportunities and perspectives for improving educational process.

The objective of the course implemented such educational and training tasks:

- To uncover the importance of role, opportunities and perspectives of the ICT in preschool and elementary education, train the skills of studying with the ICT and reasonable using these technologies in future professional activities
- To familiarize the students with the basic facilities and methods of the modern information technologies, their theoretical and technical basis
- To give knowledge to the students, abilities and skills, which necessary for their further self-improvement and self-education for the effective usage of ICT in their future professional work
- To organize students' creative activities in making their own computer programs for teaching
- To give an opportunity to each student to realize his educational trajectory in a way of differentiation and increasing the quantity of creative tasks

- To encourage students to make a collection of e-resources for learning
- To form the basics of information culture of the students

The importance of the educational and training tasks listed above requires a priority status of this and similar discipline in the curriculum of pedagogical specialties. In our opinion, it would be reasonable to extend this course and add a lot of academic hours by the variation part of the curriculum. For example, at the Department of Preschool and Elementary Education of Kherson State University (DPEE and ES KSU) there is only 18 academic hours for work in auditorium: 4 hours for lectures and 14 hours for practical lessons, while the quantity of time for independent work is 36 hours. It requires making clear tasks, forms and terms of the students' work. It was confirmed by the results of final poll in which fair quantity of students had expressed an opinion to add a lot of academic hours and extend this course.

The actual level of the development of Internet-services lets an educator write an electronic summary (e.g. in form of presentation), regularly change content without any expenses for re-edition. Students can use such electronic resources any time, if they missed a lesson or wish to revise the material. Students can get distant recommendations from the educator or other students about different questions concerning their tasks. Students can also load the files with the executed work to the site for control. Students have the opportunity to evaluate the course and quality of teaching and in this way the administration can commit monitoring. This type of organization of studying has already become a standard in many leading universities all over the world, e.g. The University of Glasgow (<http://moodle.gla.ac.uk/>), University of Nottingham (<http://moodle.nottingham.ac.uk/login/index.php>), Humboldt University of Berlin (<http://moodle.hu-berlin.de/?lang=en>), European University Cyprus (<http://moodleold.euc.ac.cy/login/index.php>), Saint Petersburg University of Technology (<http://95.163.77.122/en/>), First Moscow State Medical University (<http://moodle.spbstu.ru/>).

In most cases, information and communication technologies were used to maintain educational process at university, to supplement and intensify studying in the classrooms with a radical change in didactics, for example utility of presentations at lectures, internet access to website with lecture material. The appearance of technologies such as the Internet, computer multimedia and World Wide Web, Web2.0 has led to a number of significant changes in teaching and learning process.

Our course satisfies all these norms and the Moodle platform for distant learning helps even the beginner to realize it conveniently and easily (see <http://ksuonline.ksu.ks.ua>) [11].

### **3 Didactical and Instrumental Setting**

The main webpage visualizes the structure of the course; it consists of the annotation and the blocks with hyperlinks to each topic. The course contains the entrance poll and testing, which give more information about the audience, identify the level of



residual knowledge of Informatics of the students that gives an opportunity to optimize educational process. According to the results of the entrance test students are subdivided in two levels of difficulty of the tasks: intermediate Level A and advanced Level B. The course includes also the final poll. Despite the fact the students spend a lot of time at the computer, they use it mainly for entertainments. So the level of residual knowledge of the school Informatics course is low and 79% of students studied at level A, respectively 21% performed the tasks of level B. The latter level we proposed to the students, which showed high results of the entrance testing: a quantity of correct answers had to amount more than 75 %.

The content of lectures includes the basic definitions of ICT, approaches to using ICT in education and classification of pedagogical software, special issues of using ICT by children, new opportunities for teachers that are offered by technology Web 2.0. The theoretical part of the course also contains modern requirements to elementary teacher's ICT-competency. The priority of the course is its professional focus, so the lectures include peculiarities and approaches of using ICT, sanitary and hygienic norms of working at computer in kindergartens and elementary schools, recommendations for choosing pedagogical software.

We formulated the following requirements to the practical tasks for students on the ground of methodological literature analysis:

1. Using ICT in the educational process, on the one hand, facilitates to the systematization of student's knowledge in this sphere, on the other hand, reduces the level of creative activities. Therefore, the need to increase amount of creative tasks appears to be the compensation for such ICT influence.
2. Tasks should be directed to the forming self-education skills that is one of the factors of the further professional development.
3. Tasks should raise internal motivation for education, particularly for studying opportunities of using ICT in the future teaching practice.
4. Tasks should be oriented to the future professional activities.
5. Tasks should form understanding of modern tendencies of the ICT development and using it in the educational process.

Practical tasks were structured in four blocks: "Information and communications", "Creating and using an educational presentation", "Text documents for a teacher", "Using of Excel program in pedagogical practice" and distinguished between two levels: intermediate Level A and advanced Level B. Students are subdivided on two groups according to the results of the entrance test (see Table 1). We considered starting the course with the topic "Information and communications" to be principally important, because not every student has an experience of distance learning and unfortunately a lot of students even don't have e-mail or don't use it. Therefore, we had to teach students to use the Moodle platform for distant learning and give tutor's consultations in a case of difficulties or technical problems with electronic correspondence.

**Table 1.** The plan of the course “NIT and TFE”

The academic hours	Theme blocks	Auditorium tasks			Individual work	
		The content of the tasks		The forms of control	Content of the tasks	The forms of control
		A (intermediate)	B (advanced)			
2/1	Information and communications	Search of the information on the theme: “Educational programs for children”, adding the web-pages to the “Favorite”, filling the table following the example for 5 sites.	Creating the site by the tools of Google sites, filling the main page and the page of the useful recourses (min5).	Loading the file to the site. The hyperlink to the blog on the educator’s e-mail box.	The acquaintance with the rules of network etiquette. Formulating 5-10 rules of using the Internet for children of primary school age.	The answers in a text form.
2/2		Forums and polls. Searching the forums of the professional themes, registration and participation in one of them. Creating a poll.	Creating the poll on the free service and allocation it on the own site.	The hyperlink to the site on the educator’s e-mail box.	The acquaintance with the document “ICT competency standards for teachers”.	Test on KSU Online. Loading the file to the site.
2/5	Creating and using an educational presentation	Producing “film-strip” tale with music support	Creating an interactive visual aid.	Loading the file to the site.	The acquaintance with the document “Design of educational products in MS PowerPoint”. Making test with triggers.	Loading the file to the site.
2/4		Creating didactical game with the hyperlinks.	Creating didactical game with the triggers.	Loading the file to the site.	Making the design of all presentations.	Loading the file to the site.
2/2	Text documents for a teacher	Making the letter of commendation by merging the documents.	Creating labels for a student’s exercise book.	Loading the file to the site.	The “It would be interesting to know”	Loading the file to the site.
2/1	Using of the Excel program in pedagogical practice	Statistical data analysis using MS Excel.	Statistical data using MS Excel.	Loading the file to the site.	Drawing the graphics and diagrams.	Loading the file to the site.
2	Final lesson	“Children and the computer”: search of information, statistical data, its graphic presentation in MS PowerPoint.		Loading the file to the site.		Final complex test. Test. Conclusive poll.

Among practical tasks there are such ones as creating and editing a site by the tools of Google Sites, filling its content; communicating in a blog or forum on the professional topics; producing “film-strip” tale with music support; creating interactive visual aids and didactical games by means of MS PowerPoint; creating labels for student’s exercise book; making certificate of good work and conduct by merging documents in MS Word; statistical data analysis using MS Excel etc.

We’ve got interesting results of anonymous final students’ poll. They noticed that the most difficult creative tasks are the most attractive and useful for them. The analysis of data showed 64% of respondents answering the question “What task was the most difficult one?” Chose following answers: “Creating a didactical game with the hyperlinks”, “Making test with triggers”, “Producing “film-strip” tale”. As an answer to the question “What task do you consider to be the most useful for you?” 79% of students have selected variants “Producing “film-strip” tale”, “Creating a didactical game with hyperlinks”, “Certificate of good work and conduct by merging the documents”. About 67% of students gave the following answers: “Producing “film-strip” tale”, “Creating a didactical game with hyperlinks”, “Making poster “It would be interesting to know”, “Making certificate of good work and conduct by merging the documents” on the question “What of the tasks did arouse the greatest interest?”.

We have to mention that statistic data presented earlier characterize answers of students, who studied at Level A (79% of the whole group). Approximately such statistical data describe poll’s results in the group of Level B.

The practical tasks have unified structure: a title, an objective, necessary software, criterion of assessment, examples of executed task, step-by-step instruction with illustrations. Also we have developed video-lessons and put hyperlinks to subsidiary web-resources as collateral relief. Observing the network etiquette was the important requirement to the students’ work.

The best works by our students are made publicity retrievable at the site “Virtual gallery of creative multimedia works of DPEE KSU students” <https://sites.google.com/site/museumfdpo>, which shows the real example of using free internet services in educational aims.

Among the most original products in gallery we wish to point to next works of our students: “In the vegetable garden” (Irina Govrishchenko), “The games” (Tetyana Kydrevs’ka) and “Compare numbers!” (Svitlana Shrub) [12].

This site motivates students increase quality of creating multimedia educational products and cultivates respective attitude to copyright. Furthermore, the students’ participation in creating of the open collection of multimedia projects promotes further collaboration, exchange of experience among other teachers in the future.

Tasks for individual work are not differentiated and do not require the high level skills of ICT using. They induce generalization and deepening Informatics knowledge, understanding of the role and opportunity of using ICT in future professional activities. Students have the considerable duration of time (36 hours) for the individual work that predetermined its clear organization (statement or problem, forms of control, execution terms etc.). Moodle Platform provides a possibility for an educator to regulate the time for execution of the tasks (for example, the access to the function “loading files” will be closed for students after the deadline).

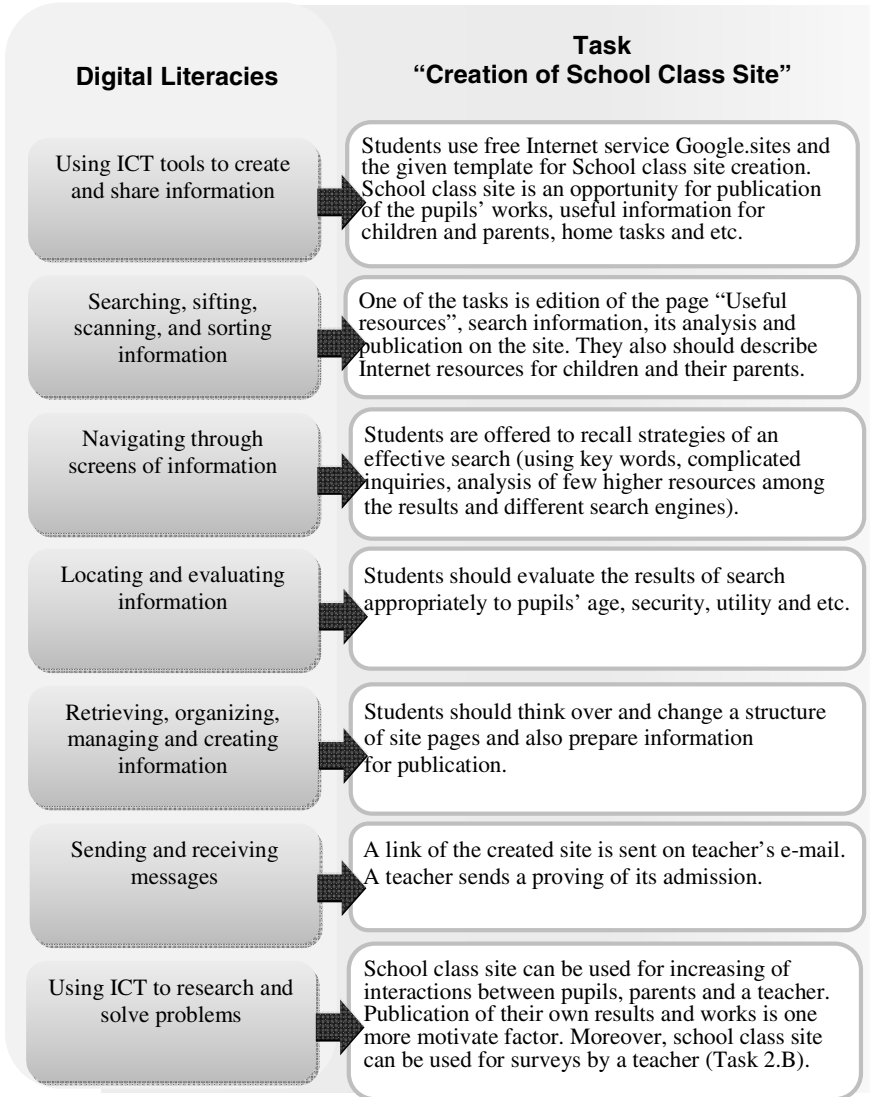
It is well known that principle “effort here and now, but result – later” reduces the motivation. Therefore it’s important that all tasks should be checked and marked before the next lesson and if students get some educator’s comments or recommendations, they can improve their product and get a better mark. Students submit a result of their work in a form determined previously by educator like “submit file on the site”, “send hyperlink to tutor’s e-mail”, “test on the site”, etc.

The authors added a section “Recourses for You” as an additional support for students, which contain the presentations of lectures, “Useful resources” – structured collating of Internet-resources, “Main definitions” (glossary). ‘Guide to algorithms” contains instructions illustrating some operation of using MS Office. These reference materials decrease the task description because elements of algorithms are not duplicated. The structure of such tasks is convenient both for “novice” and for “experienced” software users: it gives step-by-step illustrated instructions for “novices”, and it does not draw the advanced users’ attention to unnecessary detailed description that let them proceed quickly and independently [11].

Hence, we practically realized our vision of the extension of didactical purposes by including more elements of digital literacy in each course task (see Fig.1).

Digital literacies \ Tasks of course	Using ICT skills to create and share information	Searching, sifting, scanning and sorting information	Navigating through screens of information	Locating and evaluating information	Using ICT to research and solve problems	Making presentations	Retrieving, organizing, managing, and creating information	Sending and receiving messages
Creating the site of class								
Forums and polls								
Producing “film-strip” tale								
Creating didactical game								
Making a poster “It’s interesting to know”								
Final task								

Fig. 1. Including Elements of Forming Digital Literacy in the Course Tasks



**Fig. 2.** Forming Digital Literacy during an Execution Task “Creation of School Class Site”

As an example, forming elements of digital literacy during an execution of the task “Creation of School class site” are in detail depicted in Fig. 2. There is one absent element of digital literacy in Fig.2; it’s “Creation of multimedia presentation”. Students shouldn’t create a presentation in this task, though they’ll public their presentations on this site during next topic.

Evidently, not only elements of digital literacy but professional competency are formed during an execution of this task. Hence, they should evaluate searched internet-

resources not only appropriately its reliability, relevance but also check an adequacy to pupils' age, possible benefit for children and their parents and etc. Previously known criteria of evaluating the results of this task are one of the mechanisms of formative assessment and develop critical attitude to own work.

The final block of the webpage helps an educator in organizing the last lesson that includes the complex task (55 minutes) and the final test (20 minutes). Students must show skills of using at least three of four program tools offered to them at the practical lessons and abilities to search and analyze information, identify a problem question, well-reasoned presentation of the certain point of view and graphical visualization of data, particularly statistical one. So we chose broad theme of the final task "Children and computers" not to limit students' creative freedom.

The aim of the final test is to check the main learnt statements of the lecture material. Test includes questions of different form with different quantities of the right answers. Every student has only one attempt to take the test. The aim of the conclusive poll is to detect the level of student's satisfaction by quality of methodical support of the course, reveal the most interesting and useful topics for them, identify new strategies of improving the course.

Thereby developed course "NIT and TFE" fully corresponds to the modern level of educational process organization at universities. The course materials could be "disseminated" at other educational institutions without attracting the authors of this course to teaching and organizing lessons. Moreover the electronic form of this course is open for modification and improvement.

## 4 Educational Experiment and Results

The discipline has been taught in the fall semester of 2011 to the third year students in the Bachelor program Preschool and Elementary Education. 109 students attended the class and accomplished the course.

The results of final poll evidence students' understanding the role of ICT in future professional activities. So, to the question "Do you consider the skills of working at the computer to be an integral part of your teacher's activity?" 47% students gave an answer "Yes" and 42% – "Likely yes than no".

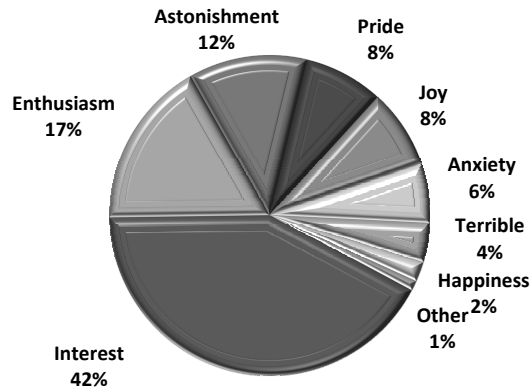
Students have estimated the quality of the course materials "NIT and TFE" as 9.29 (arithmetical mean) of 10 and novelty of lecture information for them – 7.59 respectively, practical tasks – 7.64 according to the ten-point scale.

The students described their emotional state during studying the course "NIT and TFE" as follows: 42% students felt interest, 17% – enthusiasm, 12% – astonishment, 8% – pride, 8% – joy, 6% – anxiety, 4% – terrible and 2% – happiness (see Fig.3).

Consequently, the results of our educational experiment have confirmed positive results of our work.

As for further improvement of substantive component of the course we can also follow the student's wishes expressed in conclusive poll. Summarizing students' replies the future course may include studying of other software and studying applications as MS PowerPoint (triggers), MS Word, MS Excel, using Internet, creating sites,

etc. Also students mentioned their desire to study methods of ICT using in professional activities and a need to add the hours of auditorium lessons.



**Fig. 3.** The Results of the Final Poll: Students' Emotional State

## 5 Conclusions and Future Work

Detailed and clear structure of the course is useful for supporting equal conditions and criteria (excluding double standards) and makes the process of estimation of students' academic achievements independent from the educator. We want to note, that such organization of pedagogical process chasten both students and tutors: requires their regular attention and higher level of activity. Students note when they had problems, they got help from a lecturer during the lesson – 54% and during the individual consultations – 9% (10% of students didn't have any difficulties at all and 25% got the help from their colleagues).

The participation of students in this project has become the factor of enriching their experience of effective ICT using in education. Such system and style of pedagogical work makes possible for a student to act as a subject of educational process: it also influences on further improving the course and can access through the poll developers and educators work.

Moodle distance learning platform makes it possible to present a course visualized and structured. It makes an educator's work extremely organized and clear. The use of Moodle led to positive student's attitude to the course, particularly because students can do some tasks distantly. The experience of development of this course by means of Moodle is useful for educators of any other discipline as well.

The main principle for future teachers of elementary school should be the realization that while teaching the basics of informatics they should show not only how computer operates but mainly how to use computer as a tool for solving various problems, for communication, organization of activities, particularly research activity, that causes a significant change of educational methodology and moves accents.

## References

1. Burmakina, V.F., Zelman, M., Falina, I.N.: Great Seven (G7). Information-communication-technological competence. Methodological preparation to testing teacher. The International Bank for Reconstruction and Development, Moscow (2007) (in Russian)
2. Kalaš, I.: Recognizing the potential of ICT in early childhood education. Analytical survey. UNESCO Institute for Information Technologies in Education, UNESCO IITE, Moscow (2010)
3. Prensky, M.: Digital Natives, Digital Immigrants. In: On the Horizon, vol. 9(5). NCB University Press, Lincoln (2001)
4. How, N., Strauss, W.: Millennials Rising: The Next Great Generation. Vintage Books, New York (2000)
5. Mittermeir, R.T., Sysło, M.M. (eds.): ISSEP 2008. LNCS, vol. 5090. Springer, Heidelberg (2008)
6. ICT Competency Standards for Teachers,  
<http://www.unesco.org/en/competency-standards-teachers>
7. Anderson, J.: ICT Transforming Education. A Regional Guide. UNESCO, Bangkok (2010)
8. Petuhova, L.E., Osipova, N.V., Kushnir, N.O.: Actual problems of the implementation the course ECDL in the system of training future teachers. In: Spivakovskiy, O., Kravtsov, G. (eds.) Information Technology in Education, vol. 8, pp. 17–22. KSU, Kherson (2010) (in Ukrainian)
9. Anderson, J., van Weert, T. (eds.): Information and Communication Technology in Education: A Curriculum for Schools and Programme of Teacher Development. UNESCO, Paris (2002)
10. Kashevarova, A.: Role of teacher: tutor and facilitator. Monologue psychologist. E-magazine Schoolpsychologist,  
<http://psy.1september.ru/chapterf.php?ID=200700115>
11. The course New information technologies and technical facilities of education,  
<http://ksuonline.ksu.ks.ua/course/view.php?id=10>
12. The site Virtual gallery of creative multimedia works of students DPEE KSU,  
<https://sites.google.com/site/museumfdpo/%20>
13. Kushnir, N.O., Manzhula, A.M.: The practical tasks of the course “NIT and TFE” for students of pedagogical specialties. Scientific Pedagogical Journal Parus 4, 54–60 (2012) (in Ukrainian)



# Author Index

- Alferov, Eugene 60  
Alferova, Lyudmila 60  
Alobaidi, Mizal 19
- Baklanova, Nadezhda 1  
Batyiv, Andriy 19  
Borue, Sergey 177
- Djeddai, Selma 131  
Doroshenko, Anatoliy 39
- Ermolayev, Vadim 177
- Gavrilova, Lyudmila 195
- Ivanov, Ievgen 111
- Keberle, Natalya 177  
Kulankhina, Oleksandra 165  
Kushnir, Nataliya 230
- Lavrishcheva, Ekaterina 212  
Letichevsky, Alexander 149  
Letychevskiy, Olexander 149
- Manzhula, Anna 230  
Mezghiche, Mohamed 131  
Mykhailenko, Hlib 165
- Nagy, Michal 72  
Nikitchenko, Mykola S. 89
- Ostrovski, Alexei 212
- Peschanenko, Vladimir 149
- Spivakovsky, Alexander 60  
Strecker, Martin 1, 131
- Tymofieiev, Valentyn G. 89
- Zaretska, Iryna 165  
Zhereb, Kostiantyn 39  
Zholtkevych, Grygoriy 19