Formal Aspects of Computing (1991) 3: 326–345

Formal Aspects of Computing

Context Induction: A Proof Principle for Behavioural Abstractions and Algebraic Implementations

Rolf Hennicker

Fakultät für Mathematik und Informatik, Universität Passau, Postfach 2540, D-8390 Passau, Germany

Keywords: Context induction; Behavioural specification; Behavioural theorem; Behavioural implementation; FRI implementation

Abstract. An induction principle, called context induction, is presented which is appropriate for the verification of behavioural properties of abstract data types. The usefulness of the proof principle is documented by several applications: the verification of behavioural theorems over a behavioural specification, the verification of behavioural implementations and the verification of "forget-restrict-identify" implementations.

In particular, it is shown that behavioural implementations and "forget-restrict-identify" implementations (under certain assumptions) can be characterised by the same condition on contexts, i.e. (under the given assumptions) both concepts are equivalent. This leads to the suggestion to use context induction as a uniform proof method for correctness proofs of algebraic implementations.

1. Introduction

Induction proofs play an important role in the verification of properties of programs and data types. Historically, one can distinguish computational and structural induction methods which are based on different paradigms: while computational induction works on an inductively defined set of functions (for proving properties of least fixpoints) structural induction was suggested by [Bur69] for proving properties of recursive programs by induction over the (structure of the) arguments. More generally, induction proofs are appropriate for the verification of assertions over any well founded domain (i.e. over any set on which a Noetherian ordering is defined). Particularly important domains are the (finitely generated) models of algebraic specifications where all objects

Correspondence and offprint requests to: Rolf Hennicker, Fakultät für Mathematik und Informatik, Universität Passau, Postfach 2540, D-8390 Passau, Germany.

can be denoted by a ground term and hence properties of an abstract data type can be proved by induction on the structure of ground terms (called "term induction", cf. [PBB82]), or, more generally, by induction with respect to an arbitrary Noetherian relation on ground terms. Algorithms for proving inductive theorems over data types are implemented, for instance, by Boyer and Moore's theorem prover (cf. [BoM88]) or by the Larch prover (cf. [GaG88]).

This work presents an induction principle, called *context induction,* which is appropriate for proving behavioural properties of data types. In contrast to the classical concepts the principle is *not* based on the assumption that equations (between terms) denote identities between objects rather interpreting equations as behavioural equivalences of objects as in the behavioural approaches to algebraic specifications proposed by [Rei85], [NiO88] and others. The motivation for this conception is given by the fact that from a software user's point of view, internal data representations (of an implementation) are not relevant if they induce the same observable effects, i.e. data objects can be seen as equal if they cannot be distinguished by experiments with observable result. In the framework of algebraic specifications such experiments can be formally represented by *contexts* of observable sort over the signature of a specification where a distinguished subset of its sorts is specified as observable. Thus to show that a certain property is valid for all observable experiments one can formally show this property for all corresponding contexts of observable sort. Since contexts are particular terms (over the signature of the specification) the syntactic subterm ordering defines a Noetherian relation on the set of observable contexts. Hence the proof principle of structural induction induces a proof principle for properties of observable contexts which we call context induction.

After introducing the principle of context induction (Section 3), one important purpose of this paper is to present possible applications for this proof technique. As a first application domain, in Section 4 behavioural specifications in the sense of [Rei85] and [NiO88] are considered. In contrast to the classical semantical concepts (initial, terminal, loose semantics) behavioural specifications admit a more abstract view of the semantics of a specification since equations are interpreted as behavioural equivalences. It is shown that for the behavioural analysis of a specification several properties, like membership of an algebra to the behavioural models of a specification or behavioural validity of theorems, can be expressed by corresponding properties on the set of observable contexts such that context induction provides an appropriate verification method. As an example we consider a behavioural specification of a small imperative programming language and prove by context induction a criterion for the behavioural equivalence of programs which can be easily applied to show particular equivalences.

For the application of formal specifications in the process of program development (e.g. by stepwise refinement) one needs formal implementation notions which describe correct transitions between different abstraction levels. In order to be useful in practice, formal implementation concepts should be supplied by proof methods which support the verification of correct program development steps. A major point of this work is addressed to the development of context criteria which allow the verification of implementation relations by context induction.

In Section 5, an implementation notion for behavioural specifications is

defined which formalises the intuitive idea that an implementation is correct if it produces correct observable output. Formally, a behavioural specification SP1 is called *behavioural implementation* of SP if all behavioural models of SP1 (after appropriate restriction) are behavioural models of SP as well. It is shown that the behavioural implementation relation can be characterised by a property on the set of observable contexts and hence context induction can be used for the verification of behavioural implementations. As a concrete example an implementation of a specification of states (i.e. environments of a set of identifiers with values in the natural numbers) by a "full-memory" representation of states is proved by context induction.

Section 6 deals with the well-known "forget-restrict-identify" approach to algebraic implementations (cf. e.g. [EKM82], [BMP86] and several others). The main step in those concepts is the identification of "concrete" objects which represent the same "abstract" objects e.g. by means of a congruence relation or an abstraction function. Following the approach of [BMP86] it is shown that also for "forget-restrict-identify" implementations ("FRI implementations" for short) a context criterion can be formulated and hence context induction provides an appropriate proof method also in this case. (This is not surprising since the identification of concrete representations corresponds to the behavioural equivalence of objects). As a consequence of the context criteria we show that under certain conditions FRI implementations and behavioural implementations are equivalent which leads to the suggestion to use context induction as a uniform proof technique for the verification of implementation relations in the process of formal program development.

In Section 7, a general scheme for proofs by context induction is discussed and finally, in Section 8, some further aspects are considered.

2. Basic Notions

In this section we briefly review the basic notions of algebraic specifications (for more details see e.g. [EhM85]). A (many sorted) *signature* Σ is a pair (S, F) where S is a set of *sorts* and F is a set of *function symbols*. To every function symbol $f \in F$ a functionality $s_1 \times \cdots \times s_n \to s$ with $s_1, \ldots, s_n \in S$ is associated. If $n = 0$ then f is called *constant* of sort s.

A *total* Σ*-algebra* $A = ((A_s)_{s \in S}, (f^A)_{f \in F})$ consists of a family of carrier sets $(A_s)_{s \in S}$ and a family of (total) functions $(f^A)_{f \in F}$ such that $f^A: A_{s_1} \times \cdots \times A_{s_n} \to A_s$ if f has functionality $s_1 \times \cdots \times s_n \to s$ (if $n = 0$ then f^A denotes a constant object of A_s). In this presentation we assume that $A_s \neq \emptyset$ for all $s \in S$ (for a discussion of empty carrier sets see [GoM82], [PaW84]).

A total Σ-algebra B is called Σ*-subalgebra* of A if $B_s \subseteq A_s$ for all $s \in S$ and $f^A|_B = f^B$ for all function symbols $f \in F$ where $f^A|_B$ denotes the restriction of f^A to the elements of the carrier sets of B.

A signature $\Sigma' = (S', F')$ is called *subsignature* of Σ if $S' \subseteq S$ and $F' \subseteq F$. The *restriction* of a total Σ-algebra A to Σ' is the Σ'-algebra $A|_{\Sigma'} = ((A_s)_{s \in S'}, (f^A)_{f \in F'})$.

The *term algebra* $W_\Sigma(X)$ over an S-sorted family $X = (X_s)_{s \in S}$ of sets of variables of sort s has as carriers the sets $W_\Sigma(X)_s$ of *terms* of sort s. For $(f: s_1 \times \cdots \times s_n \to s) \in F$ the corresponding function $f^{W_\Sigma(X)}$ is defined by $f^{W_\Sigma(X)}(t_1, \ldots, t_n) =_{\text{def}} f(t_1, \ldots, t_n)$. If $X = \emptyset$ then $W_\Sigma(\emptyset)$ is denoted by W_Σ and W_Σ is called *ground term algebra*.

A *substitution* $\sigma: X \to W_\Sigma(X)$ is a family of mappings $(\sigma_s: X_s \to W_\Sigma(X)_s)_{s \in S}$. For any term $t \in W_\Sigma(X)$, the *instantiation* $\sigma(t) =_{\text{def}} t[\sigma(x_1)/x_1, \ldots, \sigma(x_n)/x_n]$ is defined by replacing all variables $x_1, \ldots, x_n \in X$ occurring in t by the terms $\sigma(x_1), \ldots, \sigma(x_n)$.

The *interpretation* of a ground term $t \in W_\Sigma$ in a Σ-algebra A is denoted by t^A. If all objects of A can be denoted by a ground term then A is called *term generated* (or *finitely generated*). The *term generated subalgebra* of a Σ-algebra A is denoted by $\langle A \rangle$.

If A and B are Σ-algebras then a Σ-*homomorphism* $\phi: A \to B$ is a family of mappings $(\phi_s: A_s \to B_s)_{s \in S}$ such that for all $f \in F$ with functionality $s_1 \times \cdots \times s_n \to s$ and for all $a_1 \in A_{s_1}, \ldots, a_n \in A_{s_n}$: $\phi_s(f^A(a_1, \ldots, a_n)) = f^B(\phi_{s_1}(a_1), \ldots, \phi_{s_n}(a_n))$.

3. Context Induction

In this section we present the proof principle of *context induction* which has proved to be a powerful tool for the verification of behavioural properties of data structures and their specification. Roughly speaking, behavioural properties are obtained by forgetting unnecessary information of a data type. For example one may derive behavioural identities if one abstracts from particular data representations and identifies all objects which cannot be distinguished by experiments with observable result. In the framework of algebraic specifications such experiments can be formally represented by *contexts* of observable sort over the signature of the specification. Thus for showing that a certain property is valid for all observable experiments one can formally reason about all contexts of observable sort.

Definition 3.1. Let $\Sigma = (S, F)$ be a signature and let $Z = \{z_s \mid s \in S\}$ be an S-sorted set of variables. A term $c \in W_\Sigma(Z)$ is called *context* over Σ (or Σ-context), if c contains exactly one variable $z_s \in Z$. To indicate the variable occurring in c we often write $c[z_s]$ instead of c. The application of a context $c[z_s]$ to a term $t \in W_\Sigma$ of sort s is defined by the substitution of z_s by t. Instead of $c[t/z_s]$ we also write briefly $c[t]$. □

In the following we consider not all contexts over a given signature but restrict to those contexts with result sort belonging to a distinguished subset $S_0 \subseteq S$ of the sorts of the signature. Particularly important examples for the subset S_0 are the set of observable sorts of a behavioural specification or the set of primitive sorts of a hierarchical specification (see next sections). The contexts of observable sort, also called *observable contexts*, represent all possible experiments with observable result.

Formally, let $\Sigma = (S, F)$ be a signature and $S_0 \subseteq S$ be a subset of its sorts. The syntactic subterm ordering defines a Noetherian relation on the set of contexts $c \in W_\Sigma(Z)$ of sort $s \in S_0$. Hence the principle of structural induction (cf. [Bur69]) induces a proof principle for properties of contexts of sort $s \in S_0$, called *context induction*.

For showing that a property $P(c)$ is valid for all contexts $c \in W_\Sigma(Z)$ of sort $s \in S_0$ it is sufficient to prove the following conditions:

1. $P(z_s)$ is valid for all sorts $s \in S_0$.
2. For all contexts of the form $f(t_1, \ldots, t_{i-1}, c, t_{i+1}, \ldots, t_n)$ with a function symbol $f \in F$, $f: s_1 \times \cdots \times s_n \to s$, $s \in S_0$, terms $t_1, \ldots, t_n \in W_\Sigma$ and

a context $c \in W_\Sigma(Z)$ of sort s_i the following holds:
If $P(c')$ is valid for all subcontexts c' of c with sort $s \in S_0$, then $P(f(t_1, \ldots, t_{i-1}, c, t_{i+1}, \ldots, t_n))$ is valid. (In particular, the validity of $P(c)$ can be assumed if the sort s_i of c belongs to S_0.)

Proposition 3.1 (*context induction*). Let $\Sigma = (S, F)$ be a signature and $S_0 \subseteq S$. A property $P(c)$ is valid for all contexts $c \in W_\Sigma(Z)$ of sort $s \in S_0$ if the conditions (1) and (2) from above are satisfied.

Proof. The proof is a direct consequence of the principle of structural induction. The ordering on the set of contexts is defined by the syntactic subterm ordering. □

In the following sections we provide detailed examples for the application of context induction.

4. Behavioural Validity

As a first example for an application domain of context induction we consider the theory of behavioural specifications (cf. [Rei85] and [NiO88]). We show that for a given behavioural specification certain properties can be expressed by properties on the set of contexts of observable sort and hence context induction provides an appropriate tool for the behavioural analysis of a specification.

Following the approaches of [Rei85] and similarly of [NiO88] we first briefly summarise the basic notions of behavioural specifications. A *behavioural specification* $\mathrm{SP} = (\Sigma, \mathrm{Obs}, E)$ consists of a signature $\Sigma = (S, F)$, a subset $\mathrm{Obs} \subseteq S$ of *observable sorts* and a set E of *axioms* (here equations $t = r$ with terms $t, r \in W_\Sigma(X)$). For example the following behavioural specification STATE describes environments (also called states) of a set of identifiers with values in the natural numbers where the sorts *nat* and *bool* are specified as observable.

spec STATE = **enrich** BOOL, NAT, ID **by**
sorts: state
obs-sorts: nat, bool
functs: init: → state
update: id × nat × state → state
lookup: id × state → nat
ifstate · then · else · fi: bool × state × state → state
axioms:
lookup(x, init) = 0
lookup(x, update(y, n, s)) = ifnat eq-id(x, y)
then n else lookup(x, s) fi
update(x, n, update(y, m, s)) = ifstate eq-id(x, y)
then update(x, n, s) else update(y, m, update(x, n, s)) fi
ifstate true then s_1 else s_2 fi = s_1
ifstate false then s_1 else s_2 fi = s_2

(The constant *init* denotes the initial state, the operation *update* assigns a value to an identifier and the operation *lookup* delivers the current value of an identifier. The notation "**enrich** · · · **by**" means that STATE is a (syntactic)

enrichment of given specifications ID for the identifiers (with equality test *eq-id*), NAT for the natural numbers, and BOOL for the truth values.)

For the definition of the behavioural semantics of a specification [Rei85] and [NiO88] use the notion of *behavioural satisfaction* which is based on the idea that non-observable data objects are behaviourally equivalent if they cannot be distinguished by operations with observable result. Formally, given a signature $\Sigma = (S, F)$ and a distinguished subset $\text{Obs} \subseteq S$ of observable sorts, a term generated Σ-algebra A *satisfies behaviourally* an equation $t = r$ (written $A \models_{Obs} t = r$) if and only if for all Σ-contexts $c[z_s]$ (where s is the sort of t) of observable sort, $A \models c[t] = c[r]$ holds w.r.t. the usual satisfaction relation "$\models$". (Here only term generated algebras are considered. Hence the slight difference in the definitions of [Rei85] and [NiO88] is not relevant here.)

As a standard example the characteristic set equations $\text{add}(x, \text{add}(x, s)) = s$ and $\text{add}(x, \text{add}(y, s)) = \text{add}(y, \text{add}(x, s))$ (where x, y are variables for elements, s is a variable for sets and add is the operation which adds an element to a set) are behaviourally satisfied but not identically satisfied by the algebra of finite sequences if only the sort *bool* and hence all results of membership tests $x \in s$ are observable.

The *behaviour class* Beh(SP) of a behavioural specification SP consists of all *behavioural models* of SP, i.e. of all term generated Σ-algebras which behaviourally satisfy all axioms of SP. The *behavioural theory* BTh(SP) of SP consists of all equations $t = r$ which are behaviourally satisfied by all behavioural models of SP. From the definitions follows (see also Proposition 2.1.12 in [NiO88]) that an equation $t = r$ belongs to the behavioural theory BTh(SP) if and only if for all observable Σ-contexts $c[z_s]$ (where s is the sort of t) and for all ground substitutions σ: $X \to W_\Sigma$, $\text{SP} \vdash c[\sigma(t)] = c[\sigma(r)]$ holds (i.e. is deducible from the axioms of SP by the axioms and rules of the equational calculus, cf. [EhM85]). For example, the associativity law for the sequential composition of programs is a behavioural theorem over the specification PROG (cf. Example 4.1 below) but it is not a theorem over PROG in the classical sense.

The above discussion shows that in concrete examples the verification of behavioural properties, like behavioural satisfaction, may be a non-trivial task since (in general) one has to reason about infinitely many observable contexts. In particular, we suggest that even the restriction of a behavioural theory to ground equations is (in general) not recursively enumberable. Hence we are interested in proof methods which support the solution e.g. of the following standard problems:

1. Does a given Σ-algebra A behaviourally satisfy an equation $t = r$?
2. Is a given Σ-algebra A a behavioural model of a specification?
3. Does a given equation $t = r$ belong to the behavioural theory of a specification?

According to the above definitions and facts each of the three problems can be formally expressed by the validity of a property $P(c)$ for all contexts c of observable sort:

Definition 4.1. Let $\text{SP} = (\Sigma, \text{Obs}, E)$ be a behavioural specification, let A be a Σ-algebra and let $t, r \in W_\Sigma(X)$ be terms of the same sort. Then for any Σ-context $c[z_s]$ we define:

1. $P_{A,t=r}(c) = \text{true} \Leftrightarrow_{\text{def}}$ if t is of sort s then $A \models c[t] = c[r]$ holds,

2. $P_{A,\mathrm{SP}}(c) = \mathrm{true} \Leftrightarrow_{\mathrm{def}}$ for all $(t = r) \in E$, if t is of sort s then $A \models c[t] = c[r]$ holds,
3. $P_{t=r,\mathrm{SP}}(c) = \mathrm{true} \Leftrightarrow_{\mathrm{def}}$ if t is of sort s then for all ground substitutions $\sigma: X \to W_\Sigma$, $\mathrm{SP} \vdash c[\sigma(t)] = c[\sigma(r)]$ holds. □

With these definitions we can formulate the following fact:

Fact 4.1. Let SP, A and t, r be as in Definition 4.1.

1. $A \models_{\mathrm{Obs}} t = r$ iff for all Σ-contexts c of observable sort $P_{A,t=r}(c)$ is valid.
2. $A \in \mathrm{Beh(SP)}$ iff for all Σ-contexts c of observable sort $P_{A,\mathrm{SP}}(c)$ is valid.
3. $(t = r) \in \mathrm{BTh(SP)}$ iff for all Σ-contexts c of observable sort $P_{t=r,\mathrm{SP}}(c)$ is valid. □

For proving in concrete examples behavioural satisfaction, membership to a behaviour class or membership to a behavioural theory the principle of context induction (cf. Proposition 3.1) can be applied.

As an example we consider a behavioural specification PROG of a simple imperative programming language and give a criterion for the behavioural equivalence of programs. The specification PROG admits usual basic constructs for imperative programs: the empty statement *nop*, the sequential composition ";" of programs, the assignment ":=" of an expression to an identifier, the conditional statement *if · then · else · fi*, and the repetitive statement *for* which repeats a statement n times (for some natural number n). Based on the specification STATE from above the semantics of programs is specified by the state transition function *trans* which determines for a program p and an "old" state s the "new" state after execution of p. The function *value* computes for a given program p and a (result) expression e the evaluation of e under the final state after execution of the program. The results of such evaluations are observable since *nat* is an observable sort (of STATE and hence also of PROG).

spec PROG = **enrich** EXP **by**
  sorts: prog
  functs: nop: → prog
  .;. : prog × prog → prog
  .:=. : id × exp → prog
  if · then · else · fi : exp × prog × prog → prog
  for : nat × prog → prog
  trans : prog × state → state
  value : prog × exp → nat
  axioms:
  trans(nop, s) = s
  trans(p_1; p_2, s) = trans(p_2, trans(p_1, s))
  trans(x := e, s) = update(x, eval(e, s), s)
  trans(if e then p_1 else p_2 fi, s) = ifstate(eval(e, s) = 0)
  then trans(p_1, s) else trans(p_2, s) fi
  trans(for(n, p), s) = trans(if natexp(n) then nop
  else p; for($n - 1$, p) fi, s)
  value(p, e) = eval(e, trans(p, init))

where

spec EXP = **enrich** STATE **by**
sorts: exp
functs: natexp: nat → exp
idexp: id → exp
plus: exp × exp → exp
mult: exp × exp → exp
eval: exp × state → nat
axioms:
eval(natexp(n), s) = n
eval(idexp(x), s) = lookup(x, s)
eval(plus(e_1, e_2), s) = eval(e_1, s) + eval(e_2, s)
eval(mult(e_1, e_2), s) = eval(e_1, s) * *eval*(e_2, s)

PROG gives a behavioural specification of our simple imperative programming language where the effects of a program p w.r.t. a (result) expression e can be observed by the evaluation function *value*. Programs p and q have the same behaviour (are behaviourally equivalent) if they induce the same observable effects. Formally this means that the equation $p = q$ belongs to the behavioural theory of PROG. Hence for studying behavioural equivalences of programs we can apply Fact 4.1(3) which tells us that two programs p and q are behaviourally equivalent iff the property $P_{p=q,\mathrm{PROG}}(c)$ is valid for all contexts over PROG of observable sort *nat* or *bool*. For the verification of $P_{p=q,\mathrm{PROG}}(c)$ context induction provides an appropriate proof technique. We will apply this technique for the proof of a criterion for the behavioural equivalence of programs which can easily be applied for showing particular equivalences.

Lemma 4.1. For all ground terms $p, q \in W_{\mathrm{PROG}}$ of sort *prog* holds:
If $\mathrm{PROG} \vdash \mathrm{trans}(p, \mathrm{st}) = \mathrm{trans}(q, \mathrm{st})$ for all ground terms $\mathrm{st} \in W_{\mathrm{PROG}}$ of sort *state* then $(p = q) \in \mathrm{BTh}(\mathrm{PROG})$. ($W_{\mathrm{PROG}}$ denotes the set of ground terms over the signature of PROG.)

Proof by context induction. Let $p, q \in W_{\mathrm{PROG}}$ be arbitrary ground terms of sort *prog* such that $\mathrm{PROG} \vdash \mathrm{trans}(p, st) = \mathrm{trans}(q, st)$ holds for all "states" st. By Fact 4.2(3) we have to show that for all contexts $c[z_s]$ (over the signature of PROG) of observable sort *nat* or *bool* the property $P_{p=q,\mathrm{PROG}}(c)$ (for short $P(c)$) is valid where:

$P(c) = \mathrm{true} \Leftrightarrow_{\mathrm{def}}$ if s = prog then $\mathrm{PROG} \vdash c[p] = c[q]$ holds.

The validity of $P(c)$ is proved by *context induction*. By Proposition 3.1 one has to show that the conditions (1) and (2) (cf. Section 3) are satisfied where $S_0 = \{bool, nat\}$.

1. Let $c \equiv z_{\mathrm{nat}}$ or $c \equiv z_{\mathrm{bool}}$ be the trivial context consisting of the variable z_{nat} or z_{bool}. Then $P(z_{\mathrm{nat}})$ and $P(z_{\mathrm{bool}})$ are trivially satisfied.
2. For the induction step one has to consider all contexts (over the signature of PROG) of the form $f(\ldots, c[z_s], \ldots)$ where f has result sort *bool* or *nat*. If the context $c[z_s]$ is of sort *bool* or *nat* then the induction step is trivial since (if $s = prog$) in this case from the induction hypothesis $\mathrm{PROG} \vdash c[p] = c[q]$ immediately follows

$$\mathrm{PROG} \vdash f(\ldots, c[p], \ldots) = f(\ldots, c[q], \ldots)$$

If $f \equiv$ eq-id then $P(f(\ldots, c[z_s], \ldots))$ is valid since no context for programs with result sort *id* exists and hence $s \neq$ prog.

It remains to consider contexts of the form

eval($c[z_s]$, *st*), value(p_1, $c[z_s]$) with a context $c[z_s]$ of sort *exp*, a ground term st of sort *state* and a ground term p_1 of sort *prog*,
value($c[z_s]$, *e*) with a context $c[z_s]$ of sort *prog* and a ground term *e* of sort *exp*,
lookup(x, $c[z_s]$), eval(*e*, $c[z_s]$) with a context $c[z_s]$ of sort *state*, a ground term x of sort *id* and a ground term *e* of sort *exp*.

In the first case (contexts of the form eval($c[z_s]$, *st*), value(p_1, $c[z_s]$)) one can easily show (e.g. again by context induction) that for all contexts $c[z_s]$ of sort *exp* holds:

(*) $s \neq prog$ or $c[z_s]$ contains a subcontext of the form natexp($c'[z_s]$) with some context $c'[z_s]$ of sort *nat*.

Hence, if $s = prog$ from the induction hypothesis PROG $\vdash c'[p] = c'[q]$ immediately follows that PROG $\vdash c[p] = c[q]$ holds and therefore

$$\text{PROG} \vdash \text{eval}(c[p], \text{st}) = \text{eval}(c[q], \text{st}) \quad \text{and}$$
$$\text{PROG} \vdash \text{value}(p_1, c[p]) = \text{value}(p_1, c[q])$$

In the second case (contexts of the form value($c[z_s]$, *e*)), since PROG $\vdash$ value($c[z_s]$, *e*) = eval(*e*, trans($c[z_s]$, init)) holds, it is enough to show that for all contexts $c[z_s]$ of sort *prog* the following (more general) property $Q(c)$ is valid:

$$Q(c) = \text{true} \Leftrightarrow_{\text{def}} \text{if } s = prog \text{ then PROG} \vdash \text{trans}(c[p], \text{st}) = \text{trans}(c[q], \text{st})$$
holds for all ground terms st of sort *state*.

In the third case (contexts of the form lookup(x, $c[z_s]$), eval(*e*, $c[z_s]$)) one can easily show (e.g. again by context induction) that for all contexts $c[z_s]$ of sort *state* holds:

(**) $s \neq prog$ or $c[z_s]$ contains a subcontext of sort $s \in \{id, nat, bool\}$ or $c[z_s]$ contains a subcontext of the form trans($c'[z_s]$, st) with some context $c'[z_s]$ of sort *prog* and some ground term st of sort *state*.

Hence the third case is also an immediate consequence of $Q(c)$.

We now show the validity of $Q(c)$ by a new (nested) *context induction* over all contexts $c[z_s]$ of sort *prog*:

1. Let $c \equiv z_{\text{prog}}$ be the trivial context consisting of the variable z_{prog}. $Q(z_{\text{prog}})$ is valid since it is assumed that PROG $\vdash$ trans(p, st) = trans(q, st) holds for all ground terms st of sort *state*.
2. For the induction step one has to consider all contexts (over the signature of PROG) of the form $f(\ldots, c[z_s], \ldots)$ where f has result sort *prog*. For example we consider contexts of the form
   $c[z_s]$; p_1 with a context $c[z_s]$ of sort *prog* and a ground term p_1 of sort *prog*,
   $x := c[z_s]$ with a context $c[z_s]$ of sort *exp* and a ground term x of sort *id*,
   for(n, $c[z_s]$) with a context $c[z_s]$ of sort *prog* and a ground term n of sort *nat*.
   (The induction steps for the remaining cases of contexts of sort *prog* are simple variants of the induction steps for the first and for the second case.)

In the first case (contents of the form $c[z_s]; p_1$), if $s = prog$ from the induction hypothesis

$$\text{PROG} \vdash \text{trans}(c[p], \text{st}) = \text{trans}(c[q], \text{st}) \text{ follows that}$$
$$\text{PROG} \vdash \text{trans}(c[p]; p_1, \text{st}) = \text{trans}(p_1, \text{trans}(c[p], \text{st})) = \text{trans}(p_1, \text{trans}(c[q], \text{st})) = \text{trans}(c[q]; p_1, \text{st})$$

In the second case (contents of the form $x := c[z_s]$) we use (*) and conclude as above that if $s = prog$ then

$$\text{PROG} \vdash c[p] = c[q] \text{ holds and hence}$$
$$\text{PROG} \vdash \text{trans}(x := c[p], \text{st}) = \text{trans}(x := c[q], \text{st})$$

The third case (contexts of the form $\text{for}(n, c[z_s])$) is shown by induction on the structure of n (using the fact that all ground terms of sort *nat* can be reduced by the axioms of PROG to a normal form "succ(. . . succ(0) . . .)" and using the induction hypothesis for $Q(c)$):

Case 1 ($n = 0$):

$$\text{PROG} \vdash \text{trans}(\text{for}(0, c[p]), \text{st}) =$$
$$\text{trans}(\text{if natexp}(0) \text{ then nop else } c[p]; \text{for}(0\text{–}1, c[p]) \text{ fi}, \text{st}) = \ldots$$
$$(\text{since PROG} \vdash (\text{eval}(\text{natexp}(0), \text{st}) = 0) = \text{true})$$
$$\text{trans}(\text{nop}, \text{st}) = \ldots = \text{trans}(\text{for}(0, c[q]), \text{st})$$

Case 2 ($n \rightarrow succ(n)$):

$$\text{PROG} \vdash \text{trans}(\text{for}(\text{succ}(n), c[p]), \text{st}) =$$
$$\text{trans}(\text{if natexp}(\text{succ}(n)) \text{ then nop else } c[p]; \text{for}(n, c[p]) \text{ fi}, \text{st}) =$$
$$\ldots$$
$$(\text{since PROG} \vdash (\text{eval}(\text{natexp}(\text{succ}(n)), \text{st}) = 0) = \text{false})$$
$$\text{trans}(c[p]; \text{for}(n, c[p]), \text{st}) =$$
$$\text{trans}(\text{for}(n, c[p]), \text{trans}(c[p], \text{st})) =$$
$$(\text{by induction hypothesis for } Q(c))$$
$$\text{trans}(\text{for}(n, c[p]), \text{trans}(c[q], \text{st})) =$$
$$(\text{by induction hypothesis for } n)$$
$$\text{trans}(\text{for}(n, c[q]), \text{trans}(c[q], \text{st})) = \ldots =$$
$$\text{trans}(\text{for}(\text{succ}(n), c[q]), \text{st})$$

This completes the proof of $Q(c)$ and hence the context induction for the proof of $P(c)$ is accomplished. □

As it can be seen in the proof often a generalisation of the actual assertion is necessary which is sufficient to finish the proof without further (iterated) context induction. This is the case in all standard examples of behavioural theorems and behavioural implementations (see next section) which have been proved by the author, as e.g. the implementation of stacks by arrays with pointers, the implementation of sets by lists, the implementation of states by sequences of pairs, etc.

Example 4.1. The associativity law for the sequential composition is a behavioural theorem of PROG, i.e. $p_1; (p_2; p_3) = (p_1; p_2); p_3 \in \text{BTh}(\text{PROG})$

Proof. It is straightforward to show (using the axioms of PROG) that for all "states" st, $\text{PROG} \vdash \text{trans}(p_1; (p_2; p_3), \text{st}) = \text{trans}((p_1; p_2); p_3, \text{st})$ holds. Now use Lemma 4.1. □

5. Behavioural Implementations

Formal implementation notions for specifications are a necessary prerequisite for proving the correctness of programs. To be useful in practice, formal implementation concepts should be supplied by proof methods which support the verification of correct program development steps. In this section we show that context induction provides a powerful proof technique for the verification of implementations of behavioural specifications.

One main motivation for dealing with behavioural specifications is given by the fact that in general concrete realisations of software systems do not satisfy all properties of a requirement specification but nevertheless are considered to be correct since they produce correct observable output. Hence from the observational point of view a specification should allow to abstract from non observable properties of data structures which in the approaches of [Rei85] and [NiO88] is expressed by constructing the behaviour class of a specification. This more abstract view induces a simple notion of implementation for behavioural specifications which formalises the intuitive idea that an implementation is correct if it preserves the observable properties of a requirement specification:

A behavioural specification SP1 is a behavioural implementation of SP if the behaviour class of SP1 (after appropriate restriction) is a subclass of the behaviour class of SP.

In order to rule out trivial implementations we assume in the following that each specification contains the specification BOOL with the observable sort *bool* and restrict the behaviour class of a specification to those algebras which satisfy true $\neq$ false. Then we obtain the following formal definition of behavioural implementations:

Definition 5.1. Let SP1 = ($\Sigma 1$, Obs1, $E1$) and SP = (Σ, Obs, E) be behavioural specifications such that $\Sigma \subseteq \Sigma 1$ and Obs $\subseteq$ Obs1. Moreover, let Beh(SP1) $\neq \emptyset$. SP1 is called *behavioural implementation* of SP if for all behavioural models $B \in$ Beh(SP1),

$\langle B|_\Sigma \rangle \in$ Beh(SP) holds.

($\langle B|_\Sigma \rangle$ denotes the term generated Σ-algebra which is obtained from B by first *forgetting* all sorts and operations of $\Sigma 1$ not belonging to Σ and then *restricting* to those elements which are generated by the operations of Σ, cf. Section 2.) □

Remark 5.1. The definition of behavioural implementation is a variant of the implementation concept of [Hen91a] adopted to the theory of behavioural specifications as discussed in the last section. In order to simplify the approach we have not used here conditional axioms and we have assumed that the behaviour class of an implementing specification is not empty. □

As already mentioned above a crucial point for the usefulness of formal implementation notions is the availability of proof methods which can be applied in practical examples. For behavioural implementations we obtain the following characterization by a condition on observable contexts which is the basis for implementation proofs by context induction (cf. also [Hen91a] for a

context criterion for "observational implementations"):

Proposition 5.1. Let $SP1 = (\Sigma 1, Obs1, E1)$ and $SP = (\Sigma, Obs, E)$ be as in Definition 5.1. SP1 is a behavioural implementation of SP if and only if for all Σ-contexts $c[z_s]$ of observable sort $s_0 \in Obs$ the following property $P_{SP1,SP}(c)$ is valid:

$$P_{SP1,SP}(c) = \text{true} \Leftrightarrow_{\text{def}} \text{for all axioms } (t = r) \in E \text{ and for all ground substitutions } \sigma: X \to W_\Sigma, \text{ if } t \text{ is of sort } s \text{ then } SP1 \vdash c[\sigma(t)] = c[\sigma(r)] \text{ holds.}$$

Proof. "$\Leftarrow$": Let $P_{SP1,SP}(c)$ be valid for all Σ-contexts c of observable sort of SP and let $B \in Beh(SP1)$ be an arbitrary behavioural model of SP1. It has to be shown that $\langle B|_\Sigma \rangle \in Beh(SP)$. By definition, $\langle B|_\Sigma \rangle \in Beh(SP)$ iff $\langle B|_\Sigma \rangle \models_{Obs} t = r$ for all axioms $(t = r) \in E$, i.e. iff for all $(t = r) \in E$ and for all Σ-contexts $c[z_s]$ (where s is the sort of t) of observable sort $s_0 \in Obs$, $\langle B|_\Sigma \rangle \models c[t] = c[r]$ holds. Since $\langle B|_\Sigma \rangle$ is term generated over Σ it is enough to consider instantiations $\sigma(t)$ and $\sigma(r)$ by ground substitutions $\sigma: X \to W_\Sigma$.

Now, let $c[z_s]$ be an arbitrary Σ-context of observable sort $s_0 \in Obs$, let $(t = r)$ be an axiom of SP (such that t, r are of sort s) and let $\sigma: X \to W_\Sigma$ be an arbitrary ground substitution. Then, by assumption, $SP1 \vdash c[\sigma(t)] = c[\sigma(r)]$. Since $Obs \subseteq Obs1$, c is also an observable context of SP1 and hence the equation $c[\sigma(t)] = c[\sigma(r)]$ is (identically) satisfied by all behavioural models of SP1. In particular, $B \models c[\sigma(t)] = c[\sigma(r)]$ and hence $\langle B|_\Sigma \rangle \models c[\sigma(t)] = c[\sigma(r)]$ holds, i.e. $\langle B|_\Sigma \rangle \in Beh(SP)$.

"$\Rightarrow$": *Proof by contradiction.* Assume that there exists a Σ-context $c[z_s]$ of observable sort of SP and an axiom $(t = r) \in E$ (with t, r of sort s) such that $SP1 \not\models c[\sigma(t)] = c[\sigma(r)]$ for some ground substitution $\sigma: X \to W_\Sigma$. Then the equation $c[\sigma(t)] = c[\sigma(r)]$ is not satisfied by the initial model I of SP1 (in the usual sense) and hence, since c is of observable sort, it is also not behaviourally satisfied by I. Therefore $\langle I|_\Sigma \rangle$ is not a behavioural model of SP. On the other hand I satisfies true $\neq$ false (since it is assumed that $Beh(SP1) \neq \emptyset$ and **bool** is an observable sort) and therefore $I \in Beh(SP1)$ holds. Hence SP1 is not a behavioural implementation of SP. □

Proposition 5.1 characterises behavioural implementation relations by a property on the set of observable contexts. Hence for proving behavioural implementations in concrete cases the proof technique of context induction can be applied. This will be demonstrated by an example:

Example 5.1. We give a behavioural implementation of the specification STATE (see Section 4) by a specification HISTORY which implements states by sequences of pairs consisting of an identifier and its associated value (for simplicity we have omitted here all sequence operations which are not necessary for the example). In contrast to the abstract specification of states each sequence stores not only the current value of an indentifier x but also all previous values of x. Such implementations of states are particularly useful if one wants to retrieve old states of a system or, more concretely, of one wants to test and to analyze the state transitions performed by the execution of an imperative program.

The specification HISTORY comprises a usual specification NATSEQ of finite sequences of natural numbers. The function *history* computes for a given

identifier x and some state s the history of all environments of x, i.e. the sequence of all previous values of x.

spec HISTORY = **enrich** BOOL, NAT, ID, NATSEQ **by**
sorts: state
obs-sorts: nat, bool
functs: init: $\rightarrow$ state
$\langle . , . \rangle$: id $\times$ nat $\rightarrow$ state
.o.: state $\times$ state $\rightarrow$ state
update: id $\times$ nat $\times$ state $\rightarrow$ state
lookup: id $\times$ state $\rightarrow$ nat
history: id $\times$ state $\rightarrow$ natseq
ifstate $\cdot$ then $\cdot$ else $\cdot$ fi: bool $\times$ state $\times$ state $\rightarrow$ state
axioms:
$s \; o \; \text{init} = \text{init} \; o \; s = s$
$(s \; o \; t) \; o \; u = s \; o \; (t \; o \; u)$
$\text{update}(x, n, s) = \langle x, n \rangle \; o \; s$
$\text{lookup}(x, \text{init}) = 0$
$\text{lookup}(x, \langle y, n \rangle \; o \; s) = \text{ifnat eq-id}(x, y)$
then n else $\text{lookup}(x, s)$ fi
$\text{history}(x, \text{init}) = \langle 0 \rangle$
$\text{history}(x, \langle y, n \rangle \; o \; s) = \text{ifnatseq eq-id}(x, y)$
then $\langle n \rangle \; o_{\text{natseq}} \; \text{history}(x, s)$ else $\text{history}(x, s)$ fi
ifstate true then s_1 else s_2 fi $= s_1$
ifstate false then s_1 else s_1 fi $= s_2$

Fact 5.1. History is a behavioural implementation of STATE.

Informally, this fact is clear since using the operations of the signature of STATE the behaviour of states can only be observed via the *lookup*-operation which gives the same current values independently whether a state stores "old" values or not. Formally, the behavioural implementation relation can be proved by context induction.

Proof of the fact. By proposition 5.1 one has to show that for all contexts $c[z_s]$ over the signature of STATE of observable sort *nat* or *bool* the property $P_{\text{HISTORY,STATE}}(c)$ (for short $P(c)$) is valid where:

$P(c) = \text{true} \Leftrightarrow_{\text{def}}$ for all axioms $t = r$ of STATE
and for all ground substitutions
$\sigma: X \rightarrow W_{\text{STATE}}$, if t is of sort s then
$\text{HISTORY} \vdash c[\sigma(t)] = c[\sigma(r)]$ holds.

The validity of $P(c)$ is proved by context induction:

1. Let $c \equiv z_{\text{nat}}$ or $c \equiv z_{\text{bool}}$ be the trivial context consisting of the variable z_{nat} or z_{bool}. $P(z_{\text{nat}})$ and $P(z_{\text{bool}})$ are valid since it is easy to see that HISTORY satisfies all axioms $t = r$ of STATE with t, r of sort *nat* or *bool*.
2. For the induction step one has to consider all contexts (over the signature of STATE) of the form $f(\ldots, c[z_s], \ldots)$ where f has result sort *bool* or *nat*. If the context $c[z_s]$ is of sort *bool* or *nat* then the induction step is trivial since in this case from the induction hypothesis $\text{HISTORY} \vdash c[\sigma(t)] = c[\sigma(r)]$ immediately follows

$$\text{HISTORY} \vdash f(\ldots, c[\sigma(t)], \ldots) = f(\ldots, c[\sigma(r)], \ldots)$$

for all axioms $t = r$ of STATE (where t is of sort s) and for all ground substitutions $\sigma: X \to W_{\text{STATE}}$. Hence it is enough to consider contexts of the form lookup$(x, c[z_s])$ with a context $c[z_s]$ of sort *state* and a ground term x of sort *id*.

In this case one has to show that for all contexts $c[z_s]$ of sort *state* the following property $Q(c)$ is valid:

$Q(c) = \text{true} \Leftrightarrow_{\text{def}}$ for all axioms $t = r$ of STATE and for all ground substitutions $\sigma: X \to W_{\text{STATE}}$, if t is of sort s then HISTORY $\vdash$ lookup$(x, c[\sigma(t)]) =$ lookup$(x, c[\sigma(r)])$ holds.

The proof of $Q(c)$ is done by a new (nested) *context induction* over all contexts $c[z_s]$ of sort *state*.

1. Let $c \equiv z_{\text{state}}$ be the trivial context consisting of the variable z_{state}. Then we have to consider the (three) axioms $t = r$ of STATE where t, r are of sort *state*. Since the two axioms for the auxiliary function "*ifstate* . . ." belong to the axioms of HISTORY as well it remains to show that (for ground terms x', y of sort *id* and n, m of sort *nat*):

   HISTORY $\vdash$ lookup$(x,$ update$(x', n,$ update$(y, m, s))) =$
   lookup$(x,$ ifstate eq-id(x', y) then update(x', n, s)
   else update$(y, m,$ update$(x', n, s))$ fi)

   The proof is straightforward by distinguishing all possible cases for the values of eq-id(x, x'), eq-id(x', y) and eq-id(x, y) (it is assumed that ID is sufficiently complete over BOOL such that eq-id(t, t') reduces to true or false for all ground terms t, t' of sort *id*).
2. For the induction step one has to consider all contexts (over the signature of STATE) of the form $f(\ldots, c[z_s], \ldots)$ where f has result sort *state*. If c is of sort *nat* or *bool* we are ready by the overall induction hypothesis for $P(c)$ since in this case HISTORY $\vdash c[\sigma(t)] = c[\sigma(r)]$ implies

   HISTORY $\vdash f(\ldots, c[\sigma(t)], \ldots) = f(\ldots, c[\sigma(r)], \ldots)$

   Hence it is enough to consider contexts of the form

   update$(x', n, c[z_s])$ with a context $c[z_s]$ of sort *state* and ground terms x', n of sort *id*, *nat* resp.,
   ifstate b then $c[z_s]$ else st fi with a context $c[z_s]$ of sort *state* and ground terms b, st of sort *bool*, *state* resp.

   In the first case we distinguish whether eq-id(x, x') reduces to true or to false. If e.g. eq-id(x, x') reduces to false then for each axiom $t = r$ of STATE (with appropriate sort) and for each ground substitution $\sigma: X \to W_{\text{STATE}}$:

   HISTORY $\vdash$ lookup$(x,$ update$(x', n, c[\sigma(t)])) =$
   lookup$(x, \langle x', n \rangle \; o \; c[\sigma(t)]) =$
   ifnat eq-id(x, x') then n else lookup$(x, c[\sigma(t)])$ fi =

ifnat false then n else lookup(x, $c[\sigma(t)]$) fi =
lookup(x, $c[\sigma(t)]$) =
(by induction hypothesis for $Q(c)$) =
lookup(x, $c[\sigma(r)]$) = $\cdots$ =
lookup(x, update(x', n, $c[\sigma(r)]$))

In the second case (contexts of the form "ifstate b then $c[z_s]$ else st fi") the induction step follows analogously from the induction hypothesis and from the fact that HISTORY is sufficiently complete over BOOL.

This completes the proof of $Q(c)$ and hence the context induction for the proof of $P(c)$ is accomplished. In summary we have shown that HISTORY is a behavioural implementation of the specification STATE. □

6. FRI Implementations

In the last section we considered behavioural implementation relations and their verification by context induction. Important alternative approaches to formal implementations are based on the "forget-restrict-identify" concept which requires to connect the model(s) of a "concrete" specification with the model(s) of an "abstract" specification e.g. by means of an abstraction function or a congruence relation (cf. e.g. [EKM82] for the initial semantics approach, [SaW82], [BMP86] for the loose semantics approach). Following the loose approach of [BMP86] in this section we give a context criterion for "forget-restrict-identify" (FRI) implementations which implies that context induction is also an appropriate proof method for FRI implementations. Moreover, as a consequence of the context criterion, we obtain that under certain conditions FRI implementations and behavioural implementations are equivalent. This leads to the suggestion to use context induction as a uniform proof technique for the verification of implementation relations.

Before we give the definition of FRI implementation the underlying notions of hierarchical specifications used in [BMP86] are briefly summarised. (In contrast to [BMP86] the definitions are restricted here to the case of equational axioms and total Σ-algebras.)

Definition 6.1 ([BMP86]). An (equational) *hierarchical specification* $SP = (\Sigma, E, P)$ consists of a signature Σ, a set E of equations (called axioms) and a primitive type $P = (\Sigma_P, E_P)$ where $\Sigma_P \subseteq \Sigma$, $E_P \subseteq E$. It is assumed that any primitive type P contains the specification BOOL of the truth values.

A Σ-algebra A is called *model* of a hierarchical specification SP if A is term generated, $A \models t = r$ for all axioms $(t = r) \in E$, $A \models \text{true} \neq \text{false}$ and $A|_{\Sigma_P}$ is a model of P (in particular $A|_{\Sigma_P}$ is term generated over Σ_P.) The *model class* of SP is denoted by Mod(SP). A specification is called *monomorphic* if it admits (up to isomorphism) only one model.

A hierarchical specification is called *sufficiently complete* if for all terms $t \in W_\Sigma$ of primitive sort there exists a primitive term $p \in W_{\Sigma_P}$ such that $SP \vdash t = p$. □

Based on hierarchical specifications the notion of FRI implementation is defined. (Note that in [BMP86], FRI implementations are called "algebraic implementations". In order to distinguish more clearly between the behavioural and the forget-restrict-identify approach we have changed this terminology.)

Definition 6.2 ([BMP86]). Let $SP1 = (\Sigma 1, E1, P)$ and $SP = (\Sigma, E, P)$ be hierarchical specifications with the same primitive type P and with $\Sigma \subseteq \Sigma 1$. Moreover, let SP1 be consistent, i.e. $Mod(SP1) \neq \emptyset$.

SP1 is called *FRI implementation* of SP if for all models $B \in Mod(SP1)$ there exists a model $A \in Mod(SP)$ and a Σ-homomorphism $\phi: \langle B|_\Sigma \rangle \to A$ (see Definition 5.1 for the notation $\langle B|_\Sigma \rangle$). □

As in the case of behavioural implementations also for the applicability of the notion of FRI implementation in practical examples the availability of appropriate proof methods is important. In the following we show that FRI implementations can be verified by using a modified version of the context criterion for behavioural implementations (cf. Proposition 5.1), where instead of observable contexts, all contexts of primitive sort are considered.

Proposition 6.1. Let $SP1 = (\Sigma 1, E1, P)$ and $SP = (\Sigma, E, P)$ be as in Definition 6.2 and let S_p be the set of primitive sorts (i.e. sorts of P).
SP1 is an FRI implementation of SP if for all Σ-contexts $c[z_s]$ of primitive sort $s_0 \in S_p$ the property $P_{SP1,SP}(c)$ defined in Proposition 5.1 is valid.

Proof. Let $P_{SP1,SP}(c)$ be satisfied for all Σ-contexts c of primitive sort and let $B \in Mod(SP1)$ be an arbitrary model of SP1. It has to be shown that there exists a model $A \in Mod(SP)$ and a Σ-homomorphism $\phi: \langle B|_\Sigma \rangle \to A$.
Let $A =_{def} \langle B|_\Sigma \rangle / \sim$ be the quotient of $\langle B|_\Sigma \rangle$ w.r.t. the following congruence relation on $\langle B|_\Sigma \rangle$:

$$a \sim b \Leftrightarrow_{def} \text{for all } \Sigma\text{-contexts } c[z_s] \text{ (where } s \text{ is the sort of } a \text{ and } b\text{)}$$
$$\text{of primitive sort, } c[a/z_s]^B = c[b/z_s]^B \text{ holds.}$$

Obviously, $\sim$ is a congruence relation. Then the canonical epimorphism defines a Σ-homomorphism $\phi: \langle B|_\Sigma \rangle \to A$. It remains to show that A is a model of SP. Since B satisfies true $\neq$ false, A satisfies true $\neq$ false as well and since the restriction of B to the primitive signature Σ_P is term generated by Σ_P the same is true for A. Then, since A is term generated by Σ, it is enough to show that for all axioms $(t = r) \in E$ and for all ground substitutions $\sigma: X \to W_\Sigma$, $A \models \sigma(t) = \sigma(r)$ holds (i.e. $\sigma(t)^A \sim \sigma(r)^A$), or equivalently $c[\sigma(t)]^B = c[\sigma(r)]^B$ for all Σ-contexts $c[z_s]$ of primitive sort (if t is of sort s). But the latter condition is satisfied since B is a model of SP1 and it is assumed that for all Σ-contexts $c[z_s]$ of primitive sort $P_{SP1,SP}(c)$ is valid, i.e. $SP1 \vdash c[\sigma(t)] = c[\sigma(r)]$ holds (if t is of sort s). □

Remark 6.1. The condition of Proposition 6.1 on contexts is not sufficient if FRI implementations are based on the initial algebra semantics (cf. [EKM82]) since if B is the initial algebra of SP1 then the algebra A which is constructed in the proof of Proposition 6.1 is not necessarily the initial algebra of SP. □

Proposition 6.1 gives a criterion for FRI implementations by a property on the set of contexts of primitive sorts. Hence, as for behavioural implementations, for the verification of FRI implementations context induction is an appropriate proof technique. For example, we can show by the same context induction as in the proof of Fact 5.1 that HISTORY is an FRI implementation of STATE if the subspecification NAT of the natural numbers is designated as primitive type.

More generally, as a particular consequence of the context characterisation of behavioural implementations and of the context criterion for FRI im-

plementations, one obtains that if for two behavioural specifications SP1 and SP a common subspecification can be identified with all sorts observable, then SP1 is an FRI implementation of SP if it is a behavioural implementation of SP.

An interesting question is under which conditions also the reverse direction is true, i.e. when FRI implementations are behavioural implementations? An answer can be given by the following proposition which sharpens the context criterion of Proposition 6.1 to a characterisation of FRI implementations:

Proposition 6.2. Let SP1 = ($\Sigma 1$, $E1$, P) and SP = (Σ, E, P) be as in Definition 6.2. Moreover, let SP1 be sufficiently complete and let P be monomorphic. Then the validity of $P_{SP1,SP}(c)$ for all Σ-contexts $c[z_s]$ of primitive sort is a necessary condition for SP1 to be an FRI implementation of SP.

Proof. The proof is done by contradiction: Assume there exists a context $c[z_s]$ of primitive sort such that $P_{SP1,SP}(c)$ is not valid. Then there exists an axiom $(t = r) \in E$ (of sort s) and a ground substitution $\sigma: X \to W_\Sigma$ such that $SP1 \not\vdash c[\sigma(t)] = c[\sigma(r)]$. Since SP1 is sufficiently complete and consistent an initial model $I \in \mathrm{Mod}(SP1)$ exists and, by assumption, $I \not\models c[\sigma(t)] = c[\sigma(r)]$. If SP1 is an FRI implementation of SP a model $A \in \mathrm{Mod}(SP)$ exists and a Σ-homomorphism $\phi: \langle I|_\Sigma \rangle \to A$. Since P is monomorphic and c is of primitive sort, $\phi(c[\sigma(t)]^I) \neq \phi(c[\sigma(r)]^I)$, i.e. $c[\sigma(t)]^A \neq c[\sigma(r)]^A$. Consequently, A does not satisfy the axiom $t = r$ of E and hence A is not a model of SP (contradiction!) □

Proposition 6.2 and Proposition 5.1 use the same context property $P_{SP1,SP}(c)$ for the characterisation of FRI implementations and behavioural implementations. Hence, under the assumption of Proposition 6.2 both implementation concepts are equivalent:

Corollary 6.1. Let SP1 and SP be hierarchical specifications with monomorphic primitive type P and let $SP1$ be sufficiently complete. Then SP1 is an FRI implementation of SP if and only if SP1 is a behavioural implementation of SP where exactly the primitive sorts are specified as observable.

Proof. By Proposition 6.2 and Proposition 5.1. □

The above result is not surprising since observability concepts are historically founded in concepts considering data type extensions where already abstractions from non-primitive values w.r.t. the operations with primitive result were studied (cf. e.g. [GGM76], [Kam83], [BPW84]). Behavioural approaches perform the consequent next step by giving a more general definition of the semantics of specifications which includes not only the models of a specification but also all behavioural equivalent data structures. This leads to an intuitively clear and simple notion of implementation which was the basis for the development of the context criterion.

7. A General Scheme for Proofs by Context Induction

The context induction proofs in the examples show that many steps of the induction can be carried out schematically. In particular, when proving a property $P(c)$ (for all contexts c of sort $s \in S_0$) the induction assertions $P(f(\ldots, c, \ldots))$ can be generated automatically for all possible cases of f

(where f is a function symbol with result sort $s \in S_0$, c is a context of sort, say s' and the dots ... stand for arbitrary (ground) terms for the remaining arguments of f). For the proof of each assertion $P(f(\ldots, c, \ldots))$ we suggest to proceed as follows:

1. If the sort s' of the context c belongs to S_0 then prove $P(f(\ldots, c, \ldots))$ by use of the induction hypothesis (which in all our examples works quite simple or is even trivial).
2. If the sort s' of c does not belong to S_0 then find a property $Q(c)$ which implies $P(f(\ldots, c, \ldots))$ and prove $Q(c)$ by a new (nested) context induction over all contexts of sort s'. (In the simplest case $Q(c) = P(f(\ldots, c, \ldots))$ as e.g. in the proof of Fact 5.1). Obviously, the choice of an appropriate property $Q(c)$ (which is general enough to finish the proof without a further iteration of context induction) requires real brainwork and can in general not be automated. In some cases the proof of the induction assertion $P(f(\ldots, c, \ldots))$ can be simplified by analysing the form of the possible subcontexts of c (cf. e.g. the conditions (*) and (**) in the proof of Lemma 4.1).

A recursive (semi-)algorithm which implements the above proof strategy for the particular case of implementation proofs is given in [Hen91b]. Of course the final aim would be the construction of a system which allows (interactive) context induction proofs by machine.

8. Concluding Remarks

The present study shows that context induction provides an appropriate proof method for the verification of behavioural abstractions, as e.g. the behavioural validity of equations and the correctness of behavioural implementations. In particular, it has been shown that context induction is well suited for the verification of FRI implementations as well. Hence we suggest to use context induction as a uniform proof principle in the process of formal program development.

As a further possible application context induction can be used for proving identities which are valid in the terminal (final) algebra of a specification (cf. [Wan79]). In contrast to initial algebras, terminal algebras identify as much elements as possible without violating the properties of an underlying primitive data type. Hence terminal validity corresponds to behavioural validity if observable sorts and primitive sorts are identified. A formal characterisation of terminal models by contexts of primitive sort is given in [BPW84] which shows that context induction can be applied for proving terminal validities.

This approach is restricted to the case of term generated algebras. An interesting question is whether context induction is also an appropriate proof technique if non term generated algebras are considered. Then, following the approach of [NiO88], one has to reason about observable contexts which possibly contain variables of observable sort. It is obvious that the principle of context induction is also valid for contexts with variables. Since for non term generated algebras the problems 1–3 of Section 4 can be characterised by conditions for contexts with variables context induction can be applied also in these cases. For example, due to the Proposition 2.1.12 in [NiO88], an

equation $t = r$ belongs to the behavioural theorems of a specification SP (admitting non term generated models) if and only if for all observable contexts c with variables $x \in X_{obs}$ and for all substitutions $\sigma: X \to W_\Sigma(X_{obs})$ (where X_{obs} is a family of variables of observable sort), $SP \vdash c[\sigma(t)] = c[\sigma(r)]$ holds. Hence for showing that an equation $t = r$ is a behavioural theorem context induction is appropriate. But note, since in this more general case the condition for behavioural theorems is stronger than in the term generated case we obtain in general less behavioural theorems as before. For example, the associativity law for the sequential composition of programs (cf. Example 4.1) is not a behavioural theorem of PROG if non term generated (behavioural) models are considered (e.g. the "observable" equation value(for(x, p_1; (p_2; p_3)), e) = value(for(x, (p_1; p_2); p_3), e) with a variable x of observable sort *nat* is not deducible from the axioms of PROG). It should be remarked that the equation p_1; (p_2; p_3) = (p_1; p_2); p_3 satisfies the assumption of Lemma 4.1 (also if non ground terms are considered) which shows that Lemma 4.1 is not more true in the non term generated case.

Concerning behavioural implementations the definition of the implementation relation is not influenced if the behaviour class of a specification is extended to non term generated algebras as long as we are interested in executable implementations (where all data objects are finitely representable). Then in any case we restrict the behavioural models B of an implementing specification SP1 to those elements which are generated by the operations of the more abstract specification SP and therefore the context characterisation of behavioural implementations (of Proposition 5.1) and all examples remain valid.

Acknowledgements

This work has been partially sponsored by the ESPRIT Project 1550 DRAGON and by the Basic Research Working Group 3264 COMPASS. It is a revised and extended version of [Hen90].

I gratefully acknowledge many helpful comments of Martin Wirsing and Peter Padawitz.

References

[BoM88] Boyer, R. S. and Moore, J. S.: *A Computational Logic Handbook*. Academic Press, 1988.

[BMP86] Broy, M., Möller, B., Pepper, P. and Wirsing, M.: Algebraic Implementations Preserve Program Correctness. *Science of Computer Programming*, **7**(1), 35–54 (1986).

[BPW84] Broy, M., Pair, C. and Wirsing, M.: A Systematic Study of Models of Abstract Data Types. *Theoretical Computer Science*, **33**, 139–174 (1984).

[Bur69] Burstall, R. M.: Proving Properties of Programs by Structural Induction. *Computer Journal*, **12**, 41–48 (1969).

[EhM85] Ehrig, H. and Mahr, B.: *Fundamentals of Algebraic Specification 1*. EATCS Monographs on Theoretical Computer Science, Vo. 6, Springer-Verlag, 1985.

[EKM82] Ehrig, H., Kreowski, H. J., Mahr, B. and Padawitz, P.: Algebraic Implementation of Abstract Data Types. *Theoretical Computer Science*, **20**, 209–263 (1982).

[GaG88] Garland, S. J. and Guttag, J. V.: Inductive Methods for Reasoning about Abstract Data Types. *Proc. POPL'88*, pp. 219–228, 1988.

[GGM76] Giarratana, V., Gimona, F. and Montanari, U.: Observability Concepts in Abstract Data Type Specification. In: *Proc. MFCS '76, 5th Int. Symp. on Mathematical Foundations of Computer Science,* A. Mazurkiewicz (ed.), Lecture Notes in Computer Science **45,** Springer-Verlag, pp. 576–587, 1976.

[GoM82] Goguen, J. A. and Meseguer, J.: Completeness of Many-Sorted Equational Logic. *ACM SIGPLAN Notices,* **16**(7), 24–32 (1981); **17**(1), 9–17 (1982).

[Hen90] Hennicker, R.: Context Induction: a Proof Principle for Behavioural Abstractions. In: *Proc. DISCO '90, Int. Symp. on Design and Implementation of Symbolic Computation Systems*, A. Miola (ed.), Capri, April 1990, Lecture Notes in Computer Science **429,** Springer -Verlag, pp. 101–110, 1990.

[Hen91a] Hennicker, R.: Observational Implementation of Algebraic Specifications. *Acta Informatica,* **28**(3), 187–230 (1991).

[Hen91b] Hennicker, R.: A Semi-Algorithm for Algebraic Implementation Proofs. Technische Berichte der Fakultät für Mathematik und Informatik, Universität Pasau, MIP-9108, 1991.

[Kam83] Kamin, S.: Final Data Types and Their Specification. *ACM TOPLASS,* **5**(1), 97–121 (1983).

[NiO88] Nivela, M^a P. and Orejas, F.: Initial Behaviour Semantics for Algebraic Specifications. In: *Proc. 5th Workshop on Algebraic Specifications of Abstract Data Types,* D. T. Sannella and A. Tarlecki (eds), Lecture Notes in Computer Science **332,** Springer-Verlag, pp. 184–207, 1988.

[PaW84] Padawitz, P. and Wirsing, M.: Completeness of Many-Sorted Equational Logic Revisited. *Bulletin EATCS,* **24,** 88–94 (1984).

[PBB82] Pepper, P., Broy, M., Bauer, F. L., Partsch, H., Dosch, W. and Wirsing, M.: Abstrakte Datentypen: Die algebraische Spezifikation von Rechenstrukturen. *Informatik–Spektrum,* **5,** 107–119 (1982).

[Rei85] Reichel, H.: Initial Restrictions of Behaviour. *IFIP Working Conference, The role of Abstract Models in Information Processing,* 1985.

[SaW82] Sannella, D. T. and Wirsing, M.: Implementation of Parameterized Specifications. In: *Proc. ICALP '82, 9th Coll. on Automata, Languages and Programming*, M. Nielsen and E. M. Schmidt (eds), Lecture Notes in Computer Science **140,** Springer-Verlag, pp. 473–488, 1982.

[Wan79] Wand, M.: Final Algebra Semantics and Data Type Extensions. *Journal of Computer and System Sciences,* **19,** 27–44 (1979).

Received April 1990
Accepted in revised form December 1990 by E. Astesiano