Property Preserving Abstractions under Parallel Composition[*]

Susanne Graf and Claire Loiseaux

IMAG, BP 53X, F-38041 Grenoble
e-mail : {graf,loiseaux}@imag.fr

Abstract. We study property preserving transformations for reactive systems. A key idea is the use of ϱ-simulations which are simulations parametrized by a relation ϱ, relating the domains of two systems. We particularly address the problem of property preserving abstractions of composed programs. For a very general notion of parallel composition, we give the conditions under which simulation is a precongruence for parallel composition and we study which kind of global properties are preserved by these abstractions.

1 Introduction

The investigation of property preserving abstractions of reactive systems has been the object of intensive research during the last years. However, the existing theoretical results are very fragmented. They strongly depend on the choice of the specification formalism and the underlying semantics.

Some results are given in the framework of linear time semantics as e.g., in [AL88,LT88b,Kur89] where the underlying semantics of as well programs as properties are languages traces. The notions of abstractions proposed are based on the use of structure homomorphisms.

In the framework of process algebras, the problem of combination of abstraction and composition is the problem of defining property preserving equivalence relations or preorders which are congruences, respectively precongruences for parallel composition and abstraction. This problem has been studied, for equivalences e.g., in [HM85,BK85,GS86,GW89,GS90b] and for preorders in [LT88a,Wal88,CS90,SG90,GL91].

The results presented here are based on those given in [BBLS92], where a general framework for property preserving abstractions is given. Program models are transition relations and abstractions are given by ϱ-simulations, which are parameterized by a relation ϱ between the domains of both systems. Thus, we do not restrict ourselves to abstractions defined by functions from the concrete to the abstract domain as cf. in [Kur89,CGL92].

In [BBLS92] the problem of compositional abstractions is not taken up at all. Here, we extend the results on property preservation to composed abstract programs, obtained by alternating steps of abstraction and composition. For a

[*] This work was partially supported by ESPRIT Basic Research Actions "SPEC" and "REACT"

general notion of parallel composition (expressed on program models), we give conditions under which composition of abstract programs preserves properties of fragments of branching-time μ-calculus.

Our program models are transition relations on some domain D and are represented symbolically. The validity of the given results does not depend on the symbolic representation, but their usability (computation of abstract programs) does. Program models may be composed by means of three composition operators, namely a synchronous, an asynchronous and a mixed one. With these three operators we can express most of the existing composition operators, for instance those of CSP [Hoa85], Lotos [ISO89], Unity [CM88], of S/R-models [KK86] and of I/O automata [LT88b].

The results presented in the paper are the following: for programs R_i and R'_i, abstraction relations ϱ_i from the domains of R_i to the domains of R'_i, we give conditions under which:

1. R_i ϱ_i-simulates R'_i implies $R_1 \odot R_2$ $(\varrho_1 \sqcap \varrho_2)$-simulates $R'_1 \odot R'_2$ where $\odot$ is one of the three parallel operators.

   This result allows us, using the results of [BBLS92], to deduce that for any property f of the fragment $\Box L_\mu$ of the μ-calculus (defined in Section 5) such that all atomic predicates of f are preserved by ϱ (see Definition 14), then $R'_1 \odot R_2$ satisfies f implies $R_1 \odot R_2$ satisfies f.

2. If $(R_i)_{\varrho_i}$ are reasonable ϱ_i-abstractions of R_i then, $(R_1)_{\varrho_1} \odot (R_2)_{\varrho_2}$ is a reasonable $(\varrho_1 \sqcap \varrho_2)$-abstraction of $R_1 \odot R_2$,

where R_ϱ stands for the abstract program computed from R by means of the abstraction relation ϱ.

The conditions depend on the considered parallel operator $\odot$; but for all parallel operators studied here, it is not necessary that the processes are defined on independent domains. However, in order to have (2) for the asynchronous and also for the mixed parallel operator, the abstraction $\varrho_1 \sqcap \varrho_2$ must be decomposable so as the relation on the common domain is independent of the relations on the domains proper to each of the processes.

The paper is organized as follows. In the following section, we define the parallel operators. In Section 3, we present the results concerning composition and abstraction, which are illustrated by a small example in Section 4. In Section 5, we study which kind of properties are preserved by the abstractions defined in Section 3 and we illustrate these results in Section 6.

2 Parallel Composition and Abstraction Operators

First, we introduce some definitions and notations concerning intersections and unions of sets on different underlying domains.

Domains are as usual sets of valuations of program variables. We suppose a universal set of global program variables $\mathcal{V}$. Any domain is the set of valuations of some subset of program variables $V \subseteq \mathcal{V}$, denoted D_V. Thus, e.g., for $V = \{x, y\}$, $D_V = D_x \times D_y$.

Definition 1. concerning independency of domains and projection functions.

- For any V, W the domains D_V and D_W are called *independent* iff $V \cap W = \emptyset$, that means they are defined on separate variable sets.
- For any V, W, we denote by $\mathcal{R}(D_V, D_W)$, the set of binary relations from D_V to D_W. If $D_V = D_W$, we write $\mathcal{R}(D_V)$.
- For V, W such that $V \subseteq W$, we denote by π_V the projection function in $\mathcal{R}(D_W, D_V)$.

Now we can define intersections and unions of sets on different domains D_V and D_W as operators on $D_{V \cup W}$.

Definition 2. Let $D_V, D_{V_i}, D_W, D_{W_i}, i = 1, 2$ be domains, $X \subseteq D_V$, $Y \subseteq D_W$. Then we define,

- $X \cap Y = \{z \in D_{V \cup W} \mid \pi_V(z) \in X \wedge \pi_W(z) \in Y\}$
- $X \cup Y = \{z \in D_{V \cup W} \mid \pi_V(z) \in X \vee \pi_W(z) \in Y\}$

Consider binary relations $R_i \in \mathcal{R}(D_{V_i}, D_{W_i})$. Then, we define relations $R_1 \cap R_2 \in \mathcal{R}(D_{V_1 \cup V_2}, D_{W_1 \cup W_2})$ and $R_1 \times R_2 \in \mathcal{R}(D_{V_1} \times D_{V_2}, D_{W_1} \times D_{W_2})$, by

- $R_1 \cap R_2 = \{(z, z') \mid z \in D_{V_1 \cup V_2} \wedge z' \in D_{W_1 \cup W_2} \wedge (\pi_{V_1}(z), \pi_{W_1}(z')) \in R_1 \wedge (\pi_{V_2}(z), \pi_{W_2}(z')) \in R_2\}$
- $R_1 \times R_2 = \{(z, z') \mid z \in R_1 \wedge z' \in R_2\}$

We suppose that programs are represented by binary relations (transition relations) on some domain. This is a very general form of programs. We do not consider initial states since they are not necessary to obtain the results and it makes the representation of programs much simpler. In terms of TLA [Lam91], we consider programs consisting only of the invariant part.

Definition 3. Let be domains D_V, D_W and programs given by transition relations $R_1 \in \mathcal{R}(D_V)$ and $R_2 \in \mathcal{R}(D_W)$. Then, we define the transition relations of the composed processes in $\mathcal{R}(D_{V \cup W})$ by

- asynchronous composition :
  $R_1 \| R_2 \;=\; R_1 \times Id_{D_{W-V}} \;\cup\; Id_{D_{V-W}} \times R_2$
  where for any domain D, Id_D is the identity function on D.

- synchronous composition :
  $R_1 \otimes R_2 \;=\; R_1 \;\cap\; R_2$

- mixed composition :
  Consider transition relations R_i of the form $R_1 = \bigcup_{i \in I} R_{1i}$ and $R_2 = \bigcup_{j \in J} R_{2j}$. Let be $A \subseteq I \times J$, indicating which commands have to be executed synchronously, $A_1 = \{i \mid \exists j.(i,j) \in A\}$ and $A_2 = \{j \mid \exists i.(i,j) \in A\}$.
  $$R_1 [\![A]\!] R_2 \;=\; \bigcup_{(i,j) \in A} (R_{1i} \cap R_{2j}) \;\cup\; \bigcup_{i \notin A_1} (R_{1i} \times Id_{D_{W-V}}) \;\cup\; \bigcup_{j \notin A_2} (Id_{D_{V-W}} \times R_{2j})$$

Comments :

- In the asynchronous composition, in each step one of the programs executes one of the currently enabled transitions and the other idle. This operator results in the "interleaving" of the component processes if they are defined on independent domains; if not, the execution of a transition of one of the processes may change the enabling conditions of the other one.
  This operator is exactly the union operator of Unity [CM88].

- In synchronous composition, in each step both programs execute exactly one action possible in this state, such that the changes on the common variables are consistent. This operator corresponds exactly to $\wedge$ applied on programs described by TLA formulas; it is also very similar to program composition of S/R models [KK86].
  If the domains of the component processes are independent, this operator is exactly the one introduced in [GL91]; since they use also the same preorder, their results are comparable to ours in the sense that they consider a logic, subset of ours, and the particular case of independent domains.
- Finally, in the mixed composition operator, some of the actions must be executed synchronously, whereas the others are executed asynchronously. This operator is not exactly the one in CSP [Hoa85] or LOTOS [ISO89], where all processes have distinct variable sets and communicate by exchanging values; however the here defined operator allows to simulate these operators. The results stated here are valid for any operator which can be considered as a special case of the here defined operator.

The first two operators need imperatively models with shared memory between processes in order to allow communication, whereas the third one allows also communication based only on action names without shared memory. Nevertheless, in practice, processes composed by the third operator share often at least some variables which are written by one of them and read by the other (this allows to simulate the Lotos operator $|[]|$).
The mixed composition operator is the most general one as it allows to express the other ones as follows: $\|$ is equal to $[\![\emptyset]\!]$ and $\otimes$ is equal to $[\![I \times J]\!]$. We prefer however to keep these operators because they are interesting to be considered as subcases.

Lemma 4. *Let $R_1 = \bigcup_{i \in I} R_{1i}$, and $R_2 = \bigcup_{j \in J} R_{2j}$, be transition relations and $A \subseteq I \times J$ as in Definition 3. Then,*

- $R_1 \| R_2 \;=\; \bigcup_{i \in I}(R_{1i} \times \mathrm{Id}_{D_{W-V}}) \;\cup\; \bigcup_{i \in J}(\mathrm{Id}_{D_{V-W}} \times R_{2j})$
- $R_1 \otimes R_2 \;=\; \bigcup_{(i,j) \in I \times J}(R_{1i} \wedge R_{2j})$
- $R_1 [\![A]\!] R_2 \;=\; (\|_{(i,j) \in A}\,(R_{1i} \otimes R_{2j})) \;\;\|\;\; (\|_{i \notin A_1} R_{1i}) \;\;\|\;\; (\|_{j \notin A_2} R_{2j})$
  where we use the obvious n-ary extension of $\|$.
- $R_1 \| R_2 = R_1 [\![\emptyset]\!] R_2$
- $R_1 \otimes R_2 = R_1 [\![I \times J]\!] R_2$

The definition of ϱ-simulation is the same as in [BBLS92] and defines our notion of preorder on programs. First, we need to introduce the "predicate transformers" *pre* and $\widetilde{pre}$.

Definition 5. Given a relation $\varrho \in \mathcal{R}(D, D')$, we define the functions $pre[\varrho], \widetilde{pre}[\varrho] \in [2^{D'} \rightarrow 2^{D}]$ by,

- $\forall X \subseteq D' \,.\, pre[\varrho](X) = \{x \in D \mid \exists x' \in X \,.\, \varrho(x, x')\}$ defines the inverse image of X by ϱ.
- $\widetilde{pre}[\varrho]$ is the dual of $pre[\varrho]$, i.e.,
  $\forall X \subseteq D' \,.\, \widetilde{pre}[\varrho](X) = \overline{pre[\varrho](\overline{X})} = \{x \in D \mid \forall x' \in D' \,.\, \varrho(x, x') \Rightarrow x' \in X\}$

Definition 6. Let $R \in \mathcal{R}(D)$ and $R_a \in \mathcal{R}(D_a)$ be transition relations, and let ϱ be an abstraction relation in $\mathcal{R}(D, D_a)$. Then

$$R \;\varrho\text{-simulates}\; R_a \quad \text{iff} \quad pre[\varrho^{-1}] \circ pre[R] \circ \widetilde{pre}[\varrho] \;\subseteq\; pre[R_a]$$

Notice that "there exists ϱ such that ϱ-simulates" defines a preorder on programs which is the same as the one defined in [GL91], and which is also the standard simulation preorder [Mil71]. If there exists ϱ such that R ϱ-simulates R_a, we say also that R simulates R_a or R_a is an abstraction of R.

Definition 7. Let be given a program by a transition relation $R = \bigcup_i R_i \in \mathcal{R}(D)$. For any abstraction relation $\varrho \in \mathcal{R}(D, D_a)$ we define an operator $_\varrho$ yielding an abstract program $R_\varrho \in \mathcal{R}(D_a)$, defined by the transition relation,

$$R_\varrho = \varrho^{-1} \circ R \circ \varrho = \bigcup_i \varrho^{-1} \circ R_i \circ \varrho$$

The following property justifies our motivation for computing abstract programs R_ϱ from R and ϱ: for a given abstraction relation ϱ we want to compute an abstract program R_ϱ with a reasonable cost and reasonably close to R, such that a maximum of properties that are satisfied on R are also satisfied on R_ϱ. In general the least abstract program R_a such that R ϱ-simulates R_a does not exist (since $pre[\varrho^{-1}] \circ pre[R] \circ \widetilde{pre}[\varrho]$ does not necessarily distribute over $\cup$ and it is therefore not of the form $pre[R_a]$ for some relation R_a).
However, R_ϱ is **reasonable** in the sense that for any transition relation R_a, such that
R ϱ-simulates R_a and $R_a \subseteq R_\varrho$ and for any property f of the μ-calculus such that R satisfies f, we have R_a satisfies f iff R_ϱ satisfies f. This is expressed by the following proposition:

Proposition 8.

$$\frac{\exists R_a.\ R \le_\varrho R_a,\ R_a \subseteq R_\varrho}{\forall f \in \Box L_\mu,\ R_a \models_I f \iff R_\varrho \models_{I_A} f}$$

where I and $I_A = post[\varrho] \circ I$ are the interpretation functions on the concrete and abstract domains respectively.

Proof. given in the full paper.

Proposition 9. *Let R be a transition relation on D, $\varrho \in \mathcal{R}(D, D_a)$ an abstraction relation total on D (i.e. $\widetilde{pre}[\varrho] \subseteq pre[\varrho]$). Then*

- *R ϱ-simulates the abstract transition relation R_ϱ which defines a reasonable abstraction of R with respect to ϱ.*
- *If even $\widetilde{pre}[\varrho] = pre[\varrho]$, i.e., ϱ is a total function from D into D_a, R_ϱ defines the least abstraction of R with respect to ϱ.*

3 Abstraction of Composed Programs

When dealing with complex programs, it is interesting to construct abstractions as far as possible before composition. This allows to compute abstractions on smaller transition relations (and domains), and to compute the composition on the so obtained smaller abstract programs. Here we show in which cases one obtains an abstraction of the original composed program by proceeding this way, and furthermore, in which cases this can be done without losing too much with respect to the abstraction obtained proceeding the other way round.

We give conditions under which simulation is monotonic with respect to the different composition operators $\odot$, i.e.,

$$\frac{(R_1 \text{ simulates } R_1') \text{ and } (R_2 \text{ simulates } R_2')}{R_1 \odot R_2 \text{ simulates } R_1' \odot R_2'}$$

holds. We show also which kind of atomic predicates of the composed program are preserved.

Let $R_1 \in \mathcal{R}(D_V)$ and $R_2 \in \mathcal{R}(D_W)$ be transition relations and $\varrho_1 \in \mathcal{R}(D_V, D_{V_a})$, $\varrho_2 \in \mathcal{R}(D_W, D_{W_a})$ abstraction relations. For any composition operator $\odot$ of Definition 3 we have to find an abstraction relation $\varrho \in \mathcal{R}(D_{V \cup W}, D_{V_a \cup W_a})$ allowing to compute $(R_1)_{\varrho_1} \odot (R_2)_{\varrho_2}$ instead of $(R_1 \odot R_2)_\varrho$, i.e. such that $(R_1)_{\varrho_1} \odot (R_2)_{\varrho_2}$ is reasonably close to $(R_1 \odot R_2)_\varrho$.

We show that for all operators of Definition 3, R_i ϱ_i-simulates R'_i implies $R_1 \odot R_2$ $(\varrho_1 \cap \varrho_2)$-simulates $R'_1 \odot R'_2$ under some conditions on the abstraction relations ϱ_i.

Proposition 10. *Let be given transition relations $R_i \in \mathcal{R}(D_{V_i})$, $R'_i \in \mathcal{R}(D_{V_{ia}})$ and abstraction relations $\varrho_i \in \mathcal{R}(D_{V_i}, D_{V_{ia}})$ total on D_{V_i}, such that $\varrho_1 \cap \varrho_2$ is total on $D_{V_1 \cup V_2}$. Then,*

$$R_i \ \varrho\text{-simulates } R'_i,\ i = 1,2 \quad \text{implies} \quad R_1 \otimes R_2 \ (\varrho_1 \cap \varrho_2)\text{-simulates } R'_1 \otimes R'_2$$

Proof. given in the full paper.

We are interested in the particular case that $R_1 \otimes R_2$ $(\varrho_1 \cap \varrho_2)$-simulates $R_{1_{\varrho_1}} \otimes R_{2_{\varrho_2}}$ and furthermore $R_{1_{\varrho_1}} \otimes R_{2_{\varrho_2}}$ is an abstraction reasonably close to $(R_1 \otimes R_2)_{\varrho_1 \cap \varrho_2}$.
It turns out that if the values in $D_{V_{1a} \cap V_{2a}}$ in ϱ_1 and in ϱ_2 depend only on $D_{V_1 \cap V_2}$ and if the projection of $\varrho_1 \cap \varrho_2$ on $\mathcal{R}(D_{V_1 \cap V_2}, D_{V_{1a} \cap V_{2a}})$ is a *function*, we have,

$$R_{1_{\varrho_1}} \otimes R_{2_{\varrho_2}} = (R_1 \otimes R_2)_{\varrho_1 \cap \varrho_2}$$

and it is certainly less expensive to compute $R_{1_{\varrho_1}} \otimes R_{2_{\varrho_2}}$ than $(R_1 \otimes R_2)_{\varrho_1 \cap \varrho_2}$.

Proposition 11. *Let be given programs by transition relations $R_i \in \mathcal{R}(D_{V_i})$, and $R'_i \in \mathcal{R}(D_{V_{ia}})$. Let be $W_1 = V_1 - V_2, W = V_1 \cap V_2, W_2 = V_2 - V_1$ and $W_{1a} = V_{1a} - V_{2a}, W_a = V_{1a} \cap V_{2a}, W_{2a} = V_{2a} - V_{1a}$. Let $\varrho_i \in \mathcal{R}(D_{V_i}, D_{V_{ia}})$ be abstraction relations, such that $\varrho_1 \cap \varrho_2$ total on $D_{V_1 \cup V_2}$, and such that ϱ_i can be put into the form $\varrho_i = \varrho_{i1} \cap \varrho_{i2}$, where $\varrho_{i1} \in \mathcal{R}(D_{W_i}, D_{W_{ia}})$ and $\varrho_{i2} \in \mathcal{R}(D_{V_i}, D_{W_a})$. Then,*

$$R_i \ \varrho\text{-simulates } R'_i,\ i = 1,2 \quad \text{implies} \quad R_1 \| R_2 \ (\varrho_1 \cap \varrho_2)\text{-simulates } R'_1 \| R'_2$$

Proof. given in the full paper.

Remains to see in which cases $R_{1_{\varrho_1}} \| R_{2_{\varrho_2}}$ is a reasonable abstraction of $R_1 \| R_2$. It is easy to see that in general $R_{1_{\varrho_1}} \| R_{2_{\varrho_2}}$ and $(R_1 \| R_2)_{\varrho_1 \cap \varrho_2}$ are not comparable, but both reasonable abstractions.
In the case that $\varrho_1 \cap \varrho_2$ can be put into the form $\varrho_{11} \times \varrho \times \varrho_{21}$ such that $\varrho \in \mathcal{R}(D_W, D_{W_a})$ and $\varrho_{i1} \in \mathcal{R}(D_{W_i}, D_{W_{ai}})$ are functions, $R_{1_{\varrho_1}} \| R_{2_{\varrho_2}}$ and $(R_1 \| R_2)_{\varrho_1 \cap \varrho_2}$ coincide. If ϱ_{i1} are not functions, we have even $R_{1_{\varrho_1}} \| R_{2_{\varrho_2}} \subseteq (R_1 \| R_2)_{\varrho_1 \cap \varrho_2}$.

Proposition 12. *Let be given programs by transition relations* $R_i = \bigcup_{I_i} R_{ij} \in \mathcal{R}(D_{V_i})$ *and*
$R'_i = \bigcup_{I_i} R'_{ij} \in \mathcal{R}(D_{V_{ia}})$. *Let be* $W_1 = V_1 - V_2, W = V_1 \cap V_2, W_2 = V_2 - V_1$ *and* $W_{1a} = V_{1a} - V_{2a}, W_a = V_{1a} \cap V_{2a}, W_{2a} = V_{2a} - V_{1a}$. *Let* $\varrho_i \in \mathcal{R}(D_{V_i}, D_{V_{ia}})$ *be abstraction relations, such that* $\varrho_1 \cap \varrho_2$ *total on* $D_{V_1 \cup V_2}$, *and such that* ϱ_i *can be put into the form* $\varrho_i = \varrho_{i1} \times \varrho_{i2}$, *where* $\varrho_{i1} \in \mathcal{R}(D_{W_i}, D_{W_{ia}})$ *and* $\varrho_{i2} \in \mathcal{R}(D_W, D_{W_a})$. *Let be furthermore* $A \subseteq I_1 \times I_2$ *a synchronization set. Then,*

$$R_1 [\![A]\!] R_2 \ (\varrho_1 \cap \varrho_2)\text{-simulates } R_{1\varrho_1} [\![A]\!] R_{2\varrho_2}.$$

Proof. The fact that $R_1 [\![]\!] R_2$ can be expressed by using only $\otimes$ and $\|$ as given in Lemma 4 and that the condition of both of the preceding Propositions are satisfied is enough to prove the Proposition.

Proposition 13. *Let* $R_1 \in \mathcal{R}(D_{V_1}), R_2 \in \mathcal{R}(D_{V_2})$ *be transition relations.*

1. $R_1 \otimes R_2$ *ϱ-simulates* R_1 *for some* ϱ *such that*

$$\forall X \subseteq D_{V_1}.\ pre[\varrho](pre[\varrho^{-1}](X)) = X$$

2. R_1 *ϱ-simulates* $R_1 \| R_2$ *for some* ϱ *such that*

$$\forall X \subseteq D_{V_1}.\ pre[\varrho](pre[\varrho^{-1}](X)) = X$$

Proof. Let be $W = V_1 \cup V_2$. The required abstraction relations are

- $\varrho_1 = \{(d, d') \mid d \in D_W \wedge d' \in D_{V_1} \wedge \pi_{V_1}(d) = d'\} \in \mathcal{R}(D_W, D_{V_1})$ in case (1)
- $\varrho_2 = \{(d', d) \mid d' \in D_{V_1} \wedge d \in D_W \wedge \pi_{V_1}(d) = d'\} \in \mathcal{R}(D_{V_1}, D_W)$ in case (2)

Notice that ϱ_1 is a function but ϱ_2 is not.

By using the results given in [BBLS92], Proposition 13 allows to deduce that formulas of $\Box L_\mu$ (cf. Section 5) are preserved from an asynchronous product to its components, and from each component process to the synchronous product.

Now, we obtain from the preceding Propositions and the fact that $R \odot R = R$ for any transition relation and any parallel operator, the results of [GL91] as a particular case for the operator $\otimes$.

4 Example

In this section we illustrate the Propositions 11 and 12 with an example of a mobile moving on a grid.

The motion of a mobile on a grid is controlled by a controller so as to visit cyclically the points $CDACDA$.... Initially the mobile is within the rectangle defined by the points (A, B, C, D) (see figure 1). Its motion results of two independent motors.

The motor M_X makes the mobile move horizontally and M_Y vertically, the controller *Ctrl* gives orders to both motors. We describe processes with a set of guarded commands of the following form:

$(label)$ $guard \rightarrow command$

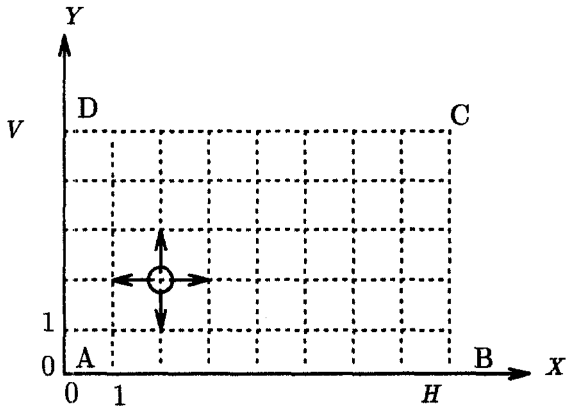

Fig. 1. Mobile

where the *label* identifies the guarded command and can be used for synchronization, the *guard* is a boolean condition which authorizes or not the execution of the *command.*

The motor M_X is defined on the variables:
- dir_X: a three-valued variable denoting the movement direction (Left, Right or Stop);
- X: a real number which denotes the position of the mobile on the horizontal axe.
- δ_X is a random input and is a positive real number.

Its transition relation is given by:

$$\begin{array}{lll} M_X\text{: (right)} & (dir_X = R) \wedge (X + \delta_X \leq H) & \rightarrow X := X + \delta_X \\ \text{(left)} & (dir_X = L) \wedge (X - \delta_X \geq 0) & \rightarrow X := X - \delta_X \\ \text{(A)} & true & \rightarrow dir_X := R \\ \text{(C)} & true & \rightarrow dir_X := L \\ \text{(D)} & true & \rightarrow dir_X := S \end{array}$$

The motor M_Y is defined analogously.

$$\begin{array}{lll} M_Y\text{: (up)} & (dir_Y = U) \wedge (Y + \delta_Y \leq V) & \rightarrow Y := Y + \delta_Y \\ \text{(down)} & (dir_Y = D) \wedge (Y - \delta_Y \geq 0) & \rightarrow Y := Y - \delta_Y \\ \text{(A)} & true & \rightarrow dir_Y := U \\ \text{(C)} & true & \rightarrow dir_Y := S \\ \text{(D)} & true & \rightarrow dir_Y := D \end{array}$$

The controller is defined on four variables:
- X, Y denote the current position of the mobile,
- X_C,Y_C are the coordinates of the previous visited control point.

$$\begin{array}{ll} Ctrl\text{: (A) } (X_C = 0) \wedge (Y_C = V) \wedge (Y = 0) & \rightarrow Y_C := 0 \\ \text{(C) } (X_C = 0) \wedge (Y_C = 0) \wedge (X = H) \wedge (Y = V) & \rightarrow X_C := H \wedge Y_C := V \\ \text{(D) } (X_C = H) \wedge (Y_C = V) \wedge (X = 0) & \rightarrow X_C := 0 \end{array}$$

The whole program is defined by $(M_X \parallel M_Y) \quad |[(A,A),(C,C),(D,D)]| \quad Ctrl.$

This system has an infinite number of states as the mobile can be in any position within the rectangle defined by the points A, B, C and D.

In order to verify that the mobile visits cyclically the points A, C and D, if it is correctly initialized, the only information we need is whether each coordinate X (respectively Y) is equal to 0, is between 0 an H (respectively V) or is equal to H (respectively V).

We propose the following abstraction relations consisting in replacing the coordinates X and Y by three-valued variables $x \in \{h_0, h_1, h_2\}$ and $y \in \{v_0, v_1, v_2\}$ and replacing in the controller the coordinates of the control point X_C and Y_C by a three-valued variable $Pcp \in \{A, C, D\}$ recording the previous visited control point.

ϱ_X: $(dir_X, X)\varrho_X(dir_X, x)$ iff
$$(x = h_0 \wedge X = 0) \vee (x = h_1 \wedge 0 < X < H) \vee (x = h_2 \wedge X = H)$$
ϱ_Y: $(dir_Y, Y)\varrho_Y(dir_Y, y)$ iff
$$(y = v_0 \wedge Y = 0) \vee (y = v_1 \wedge 0 < Y < V) \vee (y = v_2 \wedge Y = V)$$

ϱ_{Ctrl}: $(X_C, Y_C, X, Y)\varrho_{Ctrl}(Pcp, x, y)$ iff
$$\begin{aligned}&[(x = h_0 \wedge X = 0) \vee (x = h_1 \wedge 0 < X < H) \vee (x = h_2 \wedge X = H)]\wedge\\ &[(y = v_0 \wedge Y = 0) \vee (y = v_1 \wedge 0 < Y < V) \vee (y = v_2 \wedge Y = V)]\wedge\\ &[(Pcp = A \wedge X_C = 0 \wedge Y_C = 0) \vee (Pcp = C \wedge X_C = H \wedge Y_C = V)\vee\\ &(Pcp = D \wedge X_C = 0 \wedge Y_C = V)]\end{aligned}$$

Note that the domains of M_X and M_Y are independent and so are the respective abstractions. We compute the following abstractions for the motors and the controller:

$$\begin{array}{llll}(M_X)_{\varrho_X}: & \text{(right)} & (dir_X = R) \wedge (x = h_0) & \rightarrow x := h_1\\ & \text{(right)} & (dir_X = R) \wedge (x = h_1) & \rightarrow x := h_1 \text{ or } x := h_2\\ & \text{(left)} & (dir_X = L) \wedge (x = h_2) & \rightarrow x := h_1\\ & \text{(left)} & (dir_X = L) \wedge (x = h_1) & \rightarrow x := h_0 \text{ or } x := h_1\\ & \text{(A)} & true & \rightarrow dir_X := R\\ & \text{(C)} & true & \rightarrow dir_X := L\\ & \text{(D)} & true & \rightarrow dir_X := S\end{array}$$

We obtain an analogous abstract program for M_Y.

$$\begin{array}{llll}(Ctrl)_{\varrho_{Ctrl}}: & \text{(A)} & (Pcp = D) \wedge (y_0 = v_0) & \rightarrow Pcp := A\\ & \text{(C)} & (Pcp = A) \wedge (x = h_2) \wedge (y = v_2) & \rightarrow Pcp := C\\ & \text{(D)} & (Pcp = C) \wedge (x_0 = h_0) & \rightarrow Pcp := D\end{array}$$

From Propositions 11 and 12 and the fact that $\varrho_X \cap \varrho_Y \cap \varrho_{Ctrl} = \varrho_{Ctrl}$ we have that

$P_A : ((M_X)_{\varrho_X} \parallel (M_Y)_{\varrho_Y}) \quad |[(A,A),(C,C),(D,D)]| \quad Ctrl_{\varrho_{Ctrl}}$ is an ϱ_{Ctrl}-abstraction of P.

5 Preservation of Properties

It is interesting to characterize the "global" properties preserved by the abstraction relation $\varrho_1 \cap \varrho_2$ on the compositions of abstract programs defined previously.

From the results given in [BBLS92] we have the following result on preservation of properties of $\Box L_\mu$, which is the fragment of the μ-calculus of [Koz83], consisting of the formulas without occurrences of negations and using only universal quantification on paths. $\Box L_\mu$ is strictly more expressive than linear time μ-calculus, and therefore contains all regular safety properties.

For a transition relation R, the meaning of formulas are subsets of the domain D of R, where the meaning of atomic predicates in $\mathcal{P}$ is given by an interpretation function $I : \mathcal{P} \to 2^D$.

We say R satisfies f or $R \models_I f$ if the meaning of f depending on the transition relation R and interpretation function I is equal to D.

In order to verify a property f of $\Box L_\mu$ on a program R on D with interpretation functions of atomic predicates $I : \mathcal{P} \to 2^D$ respectively $I_a : \mathcal{P} \to 2^{D_a}$, we can proceed as follows: find an abstraction relation ϱ and then,

(1) Verify $R_\varrho \models_{pre[\varrho^{-1}] \circ I} f$
or
(2) Verify $R_\varrho \models_{I_a} f$.

We know from [BBLS92] that in case (1), we have $R_\varrho \models_{pre[\varrho^{-1}] \circ I} f$ implies

$$R \models_{pre[\varrho] \circ pre[\varrho^{-1}] \circ I} f.$$

Thus, in order to obtain the initially required result, $R \models_I f$ we need for any predicate symbol p occurring in f

$$I(p) \subseteq pre[\varrho] \circ pre[\varrho^{-1}] \circ I\ (p) \quad (*)$$

As the opposite inclusion is always true, (*) equivalent to

$$pre[\varrho] \circ pre[\varrho^{-1}] \circ I\ (p) = I(p).$$

Analogously, in case (2) $R_\varrho \models_{I_a} f$ implies $R \models_{pre[\varrho] \circ I_a} f$.

As before, in order to be sure, that f is the same property on both interpretations, we need to know that all predicates p occurring in f,

$$pre[\varrho^{-1}] \circ pre[\varrho] \circ I_a\ (p) = I_a(p),$$

i.e. $I_a(p)$ is in the image of ϱ on which $pre[\varrho]$ defines an isomorphism from $image(\varrho)$ onto $image(\varrho^{-1})$ [Ore44].

Therefore, we already know which type of formulas we are allowed to verify on abstract programs. Here, we are interested in characterizing the set of predicates (considered as subsets of the domain D, respectively D') of the composed concrete program that can be used in these formulas, such that f is preserved in the way explained above.

Definition 14. Let be D, D_a domains, $I : \mathcal{P} \to 2^D$ respectively $I_a : \mathcal{P} \to 2^{D_a}$ interpretation functions of atomic predicates and ϱ an abstraction relation in $\mathcal{R}(D, D_a)$. Then we say for a predicate p that it is preserved by ϱ iff

$$pre[\varrho] \circ pre[\varrho^{-1}] \circ I\ (p) = I(p) \quad \text{respectively} \quad pre[\varrho^{-1}] \circ pre[\varrho] \circ I_a\ (p) = I_a(p).$$

Notice that this notion of preservation of predicates depends only on the abstraction relation ϱ, and not on the particular program (i.e. transition relation) under study.

In the following Proposition, we characterize a set of predicates on domains of programs of the form $R_1 \odot R_2$ that is preserved by relations of the form $\varrho_1 \sqcap \varrho_2$ as in the Propositions 10 to 12.

Proposition 15. *Let $\varrho_i \in \mathcal{R}(D_{V_i}, D_{V_{ia}})$, $i = 1, 2$ be abstraction relations total on D_{V_i} and such that $\varrho_1 \sqcap \varrho_2$ is total on $D_{V_1 \cup V_2}$. Let p be a subset of $D_{V_1 \cup V_2}$ (interpretation of some atomic predicate) that can be put into the form $\bigcup_{i \in J} p_i^1 \cap p_i^2$ where $p_i^1 \subseteq D_{V_1}$ and $p_i^2 \subseteq D_{V_2}$ and J finite; let p_a be a subset of $D_{V_{1a} \cup V_{2a}}$ that can be put into the form $\bigcup_{i \in J'} p_{ai}^1 \cap p_{ai}^2$ where $p_{ai}^1 \subseteq D_{V_{1a}}$ and $p_{ai}^2 \subseteq D_{V_{2a}}$ and J' finite. Then,*

- *If all the p_i^j are preserved by ϱ_j (for $i \in J$ and $j = 1, 2$), p is preserved by $\varrho_1 \sqcap \varrho_2$.*
- *If all the p_{ai}^j are preserved by ϱ_j (for $i \in J'$ and $j = 1, 2$), p_a is preserved by $\varrho_1 \sqcap \varrho_2$.*

Proof. $pre[\varrho](pre[\varrho^{-1}](\bigcup_i p_i)) = \bigcup_i pre[\varrho](pre[\varrho^{-1}](p_i))$ and $\forall i \in J$
$pre[\varrho_j](pre[\varrho_j^{-1}](p_i^j)) = p_i^j, j = 1, 2$ implies
$pre[\varrho_1 \sqcap \varrho_2](pre[\varrho_1^{-1} \sqcap \varrho_2](p_i^1 \cap p_i^2)) =$
$pre[\varrho_1](pre[\varrho_1^{-1}](p_i^1)) \cap pre[\varrho_2](pre[\varrho_2^{-1}](p_i^2)) = p_i^1 \cap p_i^2$.

Comment: Notice that not only sets of this form may be preserved by $\varrho_1 \sqcap \varrho_2$.

However, in the case that $\varrho_1 \sqcap \varrho_2$ is a product of independent relations, i.e., $\varrho_1 \sqcap \varrho_2 = \varrho_{11} \times \varrho \times \varrho_{22}$, as it has been required in Propositions 11 and 12, $pre[\varrho_1 \sqcap \varrho_2] \circ pre[\varrho_1^{-1} \sqcap \varrho_2]$ is of the form $(pre[\varrho_{11}] \circ pre[\varrho_{11}^{-1}]) \times (pre[\varrho] \circ pre[\varrho^{-1}]) \times (pre[\varrho_{22}] \circ pre[\varrho_{22}^{-1}])$.
Then, only sets p which can be put into the form

$$\bigcup_{i \in J} p_i^1 \cap p_i^x \cap p_i^2$$

where $p_i^1 \subseteq D_{V_1 - V_2}$, $p_i^x \subseteq D_{V_1 \cap V_2}$ and $p_i^2 \subseteq D_{V_2 - V_1}$ are preserved by $\varrho_1 \sqcap \varrho_2$ **iff** all the p_i^1 are preserved by ϱ_{11}, all the p_i^x are preserved by ϱ and all the p_i^2 are preserved by ϱ_{22}. That means instead of dealing with relations in $\mathcal{R}(D_{V_1 \cup V_2}, D_{V_{1a} \cup V_{2a}})$ we deal only with relations on subdomains.

6 Example Continued

From the results given in [BBLS92] we have that for any formula f in $\Box L\mu$ and any interpretation function I of atomic predicates on the abstract domain,

$P_A \models_I f$ implies $P \models_{pre[\varrho] \circ I} f$

The following CTL formula expresses the fact that the mobile, if it is correctly initialized and does effectively change control points, visits the control points A, C and D cyclically. This formula can be translated into a $\Box L\mu$ formula.

$f = (Pcp = A)$ implies $\neg(Pcp = D)$ until $(Pcp = C) \wedge$
$(Pcp = C)$ implies $\neg(Pcp = A)$ until $(Pcp = D) \wedge$
$(Pcp = D)$ implies $\neg(Pcp = C)$ until $(Pcp = A)$

In order to be sure that the formula is preserved, we have to verify that predicates that appear in the formula are preserved. The predicates involved in the formula appear only in ϱ_{Ctrl}, we verify:

$$pre[\varrho_{Ctrl}^{-1}](pre[\varrho_{Ctrl}](I(Pcp = A))) = I(Pcp = A)$$

This equality is obvious, and so are the equalities for the other predicates.

7 Discussion

We have studied property preserving abstractions of composed programs for a general notion of parallel composition. The results are close to those given in [Kur89] in the linear framework and are extensions of those given in [GL91].

A key idea is the parametrization of simulations by a relation ϱ which allows the computation of an abstract program (an idea which has been extensively used in the domain of abstract interpretation, cf. e.g. in [CC77]) and is good means to express composition of simulations.

The presented results are exploited in a tool which is currently being implemented. Its inputs are expressions using parallel and abstraction operators on boolean guarded command programs. The evaluation of such an expression results in guarded command program. Moreover, our tool verifies symbolically any μ-calculus formula on programs and allows to know whether basic predicates are preserved, in sense of Definition 14, by the applied abstractions.

Programs are represented by *sets* of relations instead of just a relation. Internally, each guarded command is implemented by a BDD ("Binary Decision Diagrams" [Bry86]) which is an efficient representation of boolean expressions. We never compute the BDD corresponding to the global transition relation as

- for the operator $[\![]\!]$, we need the transition relations of each guarded command.
- the space needed for representation in memory of a set of relations is likely to be much smaller than that needed to represent the global transition relation [HDDY92].

The tool will be connected to the Caesar tool [GS90a], which translates Lotos programs into Petri nets. For an important subclass of Lotos programs, these Petri nets can easily be translated into parallel compositions of boolean guarded command programs, which will allow to test the tool for important examples.

All the results obtained here are also valid if one represents programs by sets of functions and this should allow to obtain still smaller representations of programs as shown in [Fil91]. However, in case of functional representation, the abstract program cannot in all cases be computed as easily as R_ϱ for a program R and a relation ϱ. Experimentation is still necessary to compare the efficiencies of the two approaches.

Acknowledgements

We thank Saddek Bensalem and Joseph Sifakis for many helpful discussions and judicious remarks.

References

[AL88] M. Abadi and L. Lamport. *The existence of refinement mappings.* Technical Report SRC-29, DEC Research Center, 1988.

[BBLS92] A. Bouajjani, S. Bensalem, C. Loiseaux, and J. Sifakis. Property preserving simulations. In *Workshop on Computer-Aided Verification (CAV), Montréal,* To appear in LNCS, june 1992.

[BK85] J. A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *TCS,* 37 (1), 1985.

[Bry86] R. E. Bryant. Graph based algorithms for boolean function manipulation. *IEEE Trans. on Computation,* 35 (8), 1986.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th POPL,* january 1977.

[CGL92] E.M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. In *Symposium on Principles of Programming Languages (POPL 92),* ACM, october 1992.

[CM88] K. M. Chandy and J. Misra. *Parallel Program Design.* Addison-Wesley, Massachusetts, 1988.

[CS90] R. Cleaveland and B. Steffen. When is "partial" adequate? a logic-based proof technique using partial specifications. In *LICS,* 1990.

[Fil91] T. Filkorn. Functional extension of symbolic model checking. In *Workshop on Computer-Aided Verification 91, Aalborg (Denmark),* LNCS Vol. 575, june 1991.

[GL91] O. Grumberg and E. Long. Compositionnal model checking and modular verification. In J.C.M. Baeten and J.F. Groote, editors, *Concur '91, 2nd International Conference on Concurrency Theory,* pages 250–265, Springer-Verlag, august 1991.

[GS86] S. Graf and J. Sifakis. A logic for the specification and proof of regular controllable processes of CCS. *Acta Informatica,* 23, 1986.

[GS90a] Hubert Garavel and Joseph Sifakis. Compilation and verification of Lotos specifications. In L. Logrippo, R. L. Probert, and H. Ural, editors, *Proceedings of the 10th International Symposium on Protocol Specification, Testing and Verification (Ottawa),* IFIP, North Holland, Amsterdam, june 1990.

[GS90b] S. Graf and B. Steffen. Compositional minimisation of finite state processes. In *Workshop on Computer-Aided Verification, Rutgers,* LNCS 531, june 1990.

[GW89] R.J. Van Glabbeek and W.P. Weijland. *Branching time and abstraction in bisimulation semantics (extended abstract).* CS-R 8911, Centrum voor Wiskunde en Informatica, Amsterdam, 1989.

[HDDY92] A.J. Hu, D.L. Dill, A.J. Drexler, and C.H. Yang. Higher-level specification and verification with bdds. In *4th Workshop on Computer-Aided Verification (CAV92), Montréal,* To appear in LNCS, Springer Verlag, june 1992.

[HM85] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the Association for Computing Machinery*, 32:137–161, 1985.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall International, 1985.

[ISO89] ISO. *IS ISO/OSI 8807 - LOTOS: a formal description technique based on the temporal ordering of observational behaviour.* International Standard, ISO, 1989.

[KK86] J. Katzenelson and B. Kurshan. S/R: A Language for Specifying Protocols and other Coordinating Processes. In *5th Ann. Int'l Phoenix Conf. Comput. Commun.*, pages 286–292, IEEE, 1986.

[Koz83] D. Kozen. Results on the propositional μ-calculus. In *Theoretical Computer Science*, North-Holland, 1983.

[Kur89] R.P. Kurshan. Analysis of discrete event coordination. In *REX Workshop on Stepwise Refinement of Distributed Systems, Mook*, LNCS 430, Springer Verlag, 1989.

[Lam91] L. Lamport. *The Temporal Logic of Actions.* Technical Report 79, DEC, Systems Research Center, 1991.

[LT88a] K. G. Larsen and B. Thomsen. Compositional proofs by partial specification of processes. In *LICS 88*, 1988.

[LT88b] N.A. Lynch and M.R. Tuttle. *An introduction to Input/Ouput Automata.* MIT/LCS/TM 373, MIT, Cambridge, Massachussetts, november 1988.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *Proc. Second Int. Joint Conf. on Artificial Intelligence*, pages 481–489, BCS, 1971.

[Ore44] O. Ore. Galois connexions. *Trans. Amer. Math. Soc*, 55:493–513, February 1944.

[SG90] G. Shurek and O. Grumberg. The Modular Framework of Computer-aided Verification: Motivation, Solutions and Evaluation Criteria. In *Conference on Automatic Verification (CAV), Rutgers, NJ*, LNCS 531, Springer Verlag, 1990.

[Wal88] D. J. Walker. Bisimulation and Divergence in CCS. In *3th Symposium on Logic in Computer Science (LICS 88)*, IEEE, 1988.