Practical Symbolic Model Checking of the full μ-calculus using Compositional Abstractions

Peter Kelb
OFFIS*

Dennis Dams
Utrecht University†

Rob Gerth‡
Eindhoven University of Technology§

Abstract

We apply abstract interpretation techniques to reduce the time and space requirements for model checking the full μ-calculus over parallel processes. The abstractions can be computed compositionally. The techniques have been implemented in a StateCharts model checker. Experiments show a 17-fold reduction on the average in the size of the BDDs on non-trivial specifications.

1 Introduction

Abstraction techniques [CGL94, LGS+95, DGG94, CR94] offer one answer to the so-called state explosion problem that is inherent to verification by model checking. Such techniques allow aspects of a program that are irrelevant to the property being checked to be ignored, thus reducing the model representing the program's (relevant) behaviors.

In symbolic model checking [BCM+92], both the state space of a program and its transition relation are encoded as boolean functions which are compactly represented by Binary Decision Diagrams (BDD's) [Bry92]. In this setting, abstractions should aim at the reduction of BDD's. It turns out that the blow-up of a BDD representing the behavior of a concurrent program usually results from the interdependency between individual processes rather than from the large total number of global states.

In this paper, we present an abstraction method which abstracts from such interdependencies in designated parts of processes as indicated by a user. We provide both universal and existential abstractions, suitable for verifying universal and existential properties expressed in the full μ-calculus [Koz83]. Combination of both types allows for verification of arbitrary μ-calculus properties.

The method is fully compositional: the abstraction of a concurrent program is computed from user-specified abstractions of its individual components. Hence, only the BDD's for the transition relations of the (abstracted) components have to be pre-computed.

*Westerstraße 10–12, 26121 Oldenburg, Germany. Peter.Kelb@arbi.informatik.uni-oldenburg.de
†Dept. of Philosophy, Heidelberglaan 8, 3584 CS Utrecht, The Netherlands. Dennis.Dams@phil.ruu.nl.
‡Currently working in ESPRIT project P6021: "Building Correct Reactive Systems (REACT)".
§Dept. of Mathematics and Computing Science, P.O. Box 513, 5600 MB Eindhoven, The Netherlands. robg@win.tue.nl

The duality that exists in theory between the two types of abstraction is in practice often disturbed. The reason is that existential abstractions tend to restrict the behavior of a program to the extent that relatively few existential properties may be verified.

We investigate a solution to this problem by providing conditions under which the universal abstraction may be used for verifying existential properties as well. This results in an optimized definition of the existential abstraction, obtained by combining the original existential abstraction together with the restricted universal abstraction.

This method has been integrated in a symbolic model checker for the language of StateCharts [Har87]. Experimental results display a 17-fold reduction in space on the average for the verification of many non-trivial properties, including universal and existential properties, as well as properties that have both universal and existential aspects. As the time complexity of BDD operations is related to the size of the BDDs they operate on, the time of a symbolic model check substantially decreases as well. Our optimized existential abstractions are essential for obtaining these results. Indeed, without them no useful existential abstractions could be defined.

2 Abstract Interpretation

Many of the results and constructions below are most easily expressed using the language of *Abstract Interpretation* [CC77]; a general framework to define static analyses of programs. The basic tenet is that the operations of a programming language which operate on concrete values can be mimicked by corresponding abstract operations defined over abstract values that describe sets of concrete values.

The starting point is choosing a set of *abstract states*, ${}^{\alpha}\mathcal{V}$. An abstract state *describes* sets of concrete states. The intention is that the interpretation of μ-calculus formulae is adapted over abstractions of transition systems; such an abstract interpretation of a formula φ then yields an abstract state ${}^{\alpha}v$. This result should be *safe* in the sense that in every set of concrete states that is described by ${}^{\alpha}v$, φ holds — i.e., every such described set should be a subset of the concrete (standard) interpretation of φ. Hence, if ${}^{\alpha}v$ describes some set $V \subseteq \mathcal{V}$ ($\mathcal{V}$ is the set of all concrete states), then it is also a safe description of any superset $V' \supseteq V$. The *concretization function* $\gamma\colon {}^{\alpha}\mathcal{V} \rightarrow 2^{\mathcal{V}}$ maps every abstract state to the smallest set of concrete states that it describes. Conversely, also every set of concrete states has a 'best', or most precise description. This is formalized via an *abstraction function* $\alpha\colon 2^{\mathcal{V}} \rightarrow {}^{\alpha}\mathcal{V}$. For each $V \subseteq \mathcal{V}$, $\alpha(V)$ is the most precise description in the sense that $V \supseteq \gamma(\alpha(V))$ and $V \supseteq \gamma({}^{\alpha}v)$ imply $\gamma(\alpha(V)) \supseteq \gamma({}^{\alpha}v)$ for any ${}^{\alpha}v \in {}^{\alpha}\mathcal{V}$. Thus, $\alpha(V)$ is the least description of V w.r.t. the *approximation ordering* $\sqsubseteq$ on ${}^{\alpha}\mathcal{V}$ defined by "${}^{\alpha}v \sqsubseteq {}^{\alpha}v'$ iff $\gamma({}^{\alpha}v) \supseteq \gamma({}^{\alpha}v')$". A given γ uniquely determines an appropriate α (if it exists) by setting $\alpha(V)$ to be the least (w.r.t. $\sqsubseteq$) ${}^{\alpha}v$ such that $\gamma({}^{\alpha}v) \subseteq V$. We mention that, similarly, α determines a unique appropriate γ as well.

These requirements are often captured by saying that (α, γ) is a *Galois insertion* from $(2^{\mathcal{V}}, \supseteq)$ to $({}^{\alpha}\mathcal{V}, \sqsubseteq)$: (i) α and γ are total and monotonic, (ii) for every $V \in 2^{\mathcal{V}}$ we have $(\gamma \circ \alpha)(V) \subseteq V$, and (iii) for every ${}^{\alpha}v \in {}^{\alpha}\mathcal{V}$ we have $(\alpha \circ \gamma)({}^{\alpha}v) = {}^{\alpha}v$.

Given such an abstract interpretation of the data, functions $f\colon \mathcal{V} \rightarrow \mathcal{V}$ can be described by safe abstract interpretations ${}^{\alpha}f\colon {}^{\alpha}\mathcal{V} \to {}^{\alpha}\mathcal{V}$ that satisfy $\alpha(f(\gamma({}^{\alpha}v)) \sqsubseteq {}^{\alpha}f({}^{\alpha}v)$.[1] In particular, there is an *optimal* abstract interpretation of f defined by $\overline{{}^{\alpha}f} = \alpha \circ f \circ \gamma$, and ${}^{\alpha}f$ is safe just in case $\overline{{}^{\alpha}f} \sqsubseteq {}^{\alpha}f$ (pointwise). So, for functions, safeness means that given a description of the parameter, ${}^{\alpha}f$ yields a description of the result value.

A static analysis can then be viewed as an abstract evaluation of a property φ in which data and operations of the program are abstractly interpreted, yielding a description of any concrete evaluation of φ.

3 μ-calculus and its interpretation

The (propositional) μ-calculus generalizes a number of possibly better-known logics such as CTL, LTL and CTL*, as shown in [Dam94]. For us, its main advantage is its computational nature. The basic modalities of the logic express properties of the successors of states and every constraint on the computations of a system must be explicitly coded in terms of extremal fixed points. Indeed, the basic symbolic model checking procedure can be almost immediately read-off from the satisfaction definition.

Let $\mathcal{A}$ be some set of atomic propositions and $\mathcal{X}$ a set of variables. Moreover, let $a \in \mathcal{A}$ and $x \in \mathcal{X}$. The syntax of the formulae of $\mu L(\mathcal{A})$ is defined by

$$\varphi ::= \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \Box\varphi \mid \Diamond\varphi \mid \mu x.\varphi \mid \nu x.\varphi \mid a \mid \neg a \mid x \ .$$

A formula $\Box\varphi$ expresses that φ is true for every (immediate) successor while $\Diamond\varphi$ expresses that there is at least one successor for which φ is true. The least and greatest fixed point operators are $\mu x.$ and $\nu x.$.

Note that formulae are in *negation normal form*: negations may only appear in front of propositions. Because of the dualities between the pairs $\vee$, $\wedge$, $\Box$, $\Diamond$ and $\mu x.$, $\nu x.$[2] there is no loss of expressiveness.

As an example, the LTL formulae $\mathsf{F}a$ (on every maximal execution path, eventually the proposition a holds) and $\mathsf{G}a$ (the proposition a holds always) would be translated as $\mu x.(a \vee \Box x)$[3] and $\nu x.(a \wedge \Box x)$. We use the $\Box$ modality because LTL formulae express properties about all paths.

$\mu L(\mathcal{A})$ is interpreted w.r.t. to a *Kripke structure* $\mathcal{M} = (\mathcal{V}, \mathrm{R}, \mathcal{I})$. $\mathcal{V}$ is a set of states, $\mathrm{R} \subseteq \mathcal{V} \times \mathcal{V}$ is a total transition relation and $\mathcal{I}\colon \mathcal{A} \to 2^{\mathcal{V}}$ is the interpretation of the atomic propositions. The semantics of the μ-calculus maps a formula φ to the set of states for which φ is true, depending on an environment, e, mapping variables to sets of states. The semantics is defined

[1] $f(C) = \{f(c) \mid c \in C\}$.
[2] $\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$, $\neg\Box\varphi \equiv \Diamond\neg\varphi$ and $\neg\mu x.\varphi \equiv \nu x.\neg\varphi[\neg x/x]$.
[3] Assuming that there are no deadlocked states

as a function, $[\![\cdot]\!]$, of type:

$$
\begin{array}{lcl}
\multicolumn{3}{l}{\varphi \to (\mathcal{X} \to 2^{\mathcal{V}}) \to 2^{\mathcal{V}} \text{ such that}} \\
[\![a]\!]e & = & \mathcal{I}(a) \\
[\![\neg a]\!]e & = & \mathcal{V} \setminus \mathcal{I}(a) \\
[\![x]\!]e & = & e(x) \\
[\![\varphi_1 \vee \varphi_2]\!]e & = & [\![\varphi_1]\!]e \cup [\![\varphi_2]\!]e \\
[\![\varphi_1 \wedge \varphi_2]\!]e & = & [\![\varphi_1]\!]e \cap [\![\varphi_2]\!]e \\
[\![\Diamond\varphi]\!]e & = & \tau_{\Diamond}([\![\varphi]\!]e) \\
[\![\Box\varphi]\!]e & = & \tau_{\Box}([\![\varphi]\!]e) \\
[\![\mu x.\varphi]\!]e & = & \bigcap\{V \subseteq \mathcal{V} \mid [\![\varphi]\!]e[V/x] \subseteq V\} \\
[\![\nu x.\varphi]\!]e & = & \bigcup\{V \subseteq \mathcal{V} \mid V \subseteq [\![\varphi]\!]e[V/x]\}
\end{array}
$$

with

$$
\begin{array}{rcl}
\tau_{\Diamond}: 2^{\mathcal{V}} & \to & 2^{\mathcal{V}} \\
U & \mapsto & \mathsf{pre}_R(U) = \{v \in \mathcal{V} \mid \exists u \in U : \; R(v,u)\} \\
\tau_{\Box}: 2^{\mathcal{V}} & \to & 2^{\mathcal{V}} \\
U & \mapsto & \widetilde{\mathsf{pre}}_R(U) = \mathcal{V} \setminus \mathsf{pre}(\mathcal{V} \setminus U) = \{v \in \mathcal{V} \mid \forall u \in \mathcal{V} : \; R(v,u) \Rightarrow u \in U\}
\end{array}
$$

As every φ is monotonic the greatest $(\nu x.\varphi)$ and least $(\mu x.\varphi)$ fixed points exists ([Tar55]).

Write $[\![\cdot]\!]_{\mathcal{M}}$ to make the intended Kripke structure explicit. For closed[4] φ, we have $\mathcal{M}, v \models \varphi$ just in case $v \in [\![\varphi]\!]\bot$ where $\bot$ is the environment that maps every $x \in \mathcal{X}$ to $\emptyset$.

4 Binary decision diagrams

Reduced Ordered Binary Decision Diagrams (BDD's) [Bry86, Bry92] are a way to economically represent boolean functions in a canonical way. Although for most boolean functions the size of their BDD representation is exponentially large, in many practical cases the BDDs are sufficiently small. This, together with the fact that boolean operations, equivalence and tautology checking can be done very efficiently on BDDs, is the reason why BDDs are so popular. BDDs only supply a canonical representation relative to an arbitrary but fixed ordering on the boolean (input) variables and this ordering greatly influences the size of the BDDs.

The use of BDDs in model checking is based on coding transition relations as boolean functions. Given a transition relation $\mathrm{R} \subseteq \mathcal{V} \times \mathcal{V}$, take vectors $\vec{x}$, $\vec{x}'$ of boolean variables long enough to code for all states in $\mathcal{V}$ (e.g., take $|\vec{x}|, |\vec{x}'| \geq \log(|\mathcal{V}|)$). Then, define a boolean function $\ulcorner \mathrm{R} \urcorner(\vec{x}, \vec{x}')$[5] by

$$\ulcorner \mathrm{R} \urcorner(\vec{x}, \vec{x}') = 1 \iff \exists v, v' \in \mathcal{V} \;\; \mathrm{R}(v, v') \wedge \beta(v) = \vec{x} \wedge \beta(v') = \vec{x}' \;,$$

where β is the coding function that maps states to bit strings.

[4] A formula is closed if every variable occurring in it is bound by a fixed point operator.

[5] We usually do not make a distinction between the boolean function $\ulcorner R \urcorner$ and its BDD representation.

4.1 Symbolic model checking

Symbolic model checking is based on such BDD representations [CBM90]. The basic, non-logical operation that needs to be done is computing preconditions, $\mathsf{pre}_{\mathrm{R}}(V)$, which translates into computing *relational products*: $\exists \vec{x}'(\ulcorner \mathrm{R} \urcorner(\vec{x}, \vec{x}') \wedge \ulcorner V \urcorner(\vec{x}'))$. Obviously, to represent a set as a BDD we use its characteristic predicate. This is used in representing the interpretation function $\mathcal{I}$ as well.

To symbolicly model check a μ-calculus formula φ over a Kripke structure $\mathcal{M}$, we compute the characteristic predicate of $[\![\varphi]\!]\bot$. Hence

$$v \in [\![\varphi]\!]_{\mathcal{M}}\bot \quad \text{iff} \quad \beta(v) \in \ulcorner [\![\varphi]\!]_{\mathcal{M}} \urcorner \bot \ .$$

We follow the definition of $[\![\,\cdot\,]\!]$: Boolean connectives translate to the corresponding boolean operations on the BDDs of the (immediate) subformulae. For formulae of the form $\Diamond\varphi$ we compute $\mathsf{pre}_{\mathrm{R}}(\ulcorner [\![\varphi]\!] \urcorner)$ using a relational product; likewise for $\Box\varphi$ formulae. Fixed points are computed iteratively as usual, by successively computing their approximations until they stabilize. E.g., a least fixed point is computed as

$$\begin{aligned} \ulcorner [\![\mu x.\varphi]\!] \urcorner e \quad &= \quad \ulcorner [\![\mu x.\varphi]\!] \urcorner^{\infty} e \\ \text{where} \quad & \ulcorner [\![\mu x.\varphi]\!] \urcorner^{0} e = \mathbf{ff} \text{ and} \\ & \ulcorner [\![\mu x.\varphi]\!] \urcorner^{i+1} e = \ulcorner [\![\mu x.\varphi]\!] \urcorner^{i} e \vee \ulcorner [\![\varphi]\!] \urcorner e\,[x \mapsto \ulcorner [\![\mu x.\varphi]\!] \urcorner^{i} e] \end{aligned}$$

If the Kripke structure $\mathcal{M}$ is finite, this procedure will terminate after at most $|\mathcal{M}|$ iterations. In practice, techniques such as *iterative squaring* are used to (exponentially) speed-up the fixed-point computations (see [BCM$^+$92]).

5 Programs

Let $\mathcal{S}$ be the set of *local states*. A program P either has the form $P' \parallel P''$, for programs P' and P'' or is a labeled transition system (lts) $(\mathbf{S}, \ell, \mathbf{I})$ with $\mathbf{S} \subseteq \mathcal{S}$. The latter is also called a *process*. Here, $\ell \colon \mathbf{S} \times \mathbf{S} \to \mathcal{L}(\mathcal{S})$, where $\mathcal{L}(\mathcal{S})$ denotes the propositional logic over the set of 'propositions' $\mathcal{S}$. The intuition is that a 'transition' (s, s') with label $\ell(s, s')$ in P is enabled if the global starting state (in which P is 'at' s) satisfies the constraint expressed by the label. $\mathbf{I} \subseteq \mathbf{S}$ is the set of initial states. Figure 1 gives an example. The lts P on the right-hand-side is equivalent with $[\![P^0 \parallel P^1]\!]$ (as will be defined below). Observe that parallel processes execute synchronously.

Although such programs may look unfamiliar, they correspond to simplified Statechart models [Har87] in which no events are broadcast. Indeed, the test $\neg(t_1 \vee t_2)$ in Figure 1 corresponds to the Statechart condition `[not(IN(t_1) or IN(t_2))]`. The verifier that we have implemented deals with full Statecharts.

For convenience we want to view (global) states of parallel processes as *sets* of local states[6]. For this reason, we assume that

[6] Hence, we identify $\{\, \{s\} \mid s \in \mathbf{S} \,\}$ with the local states of an lts $(\mathbf{S}, \ell, \mathbf{I})$.

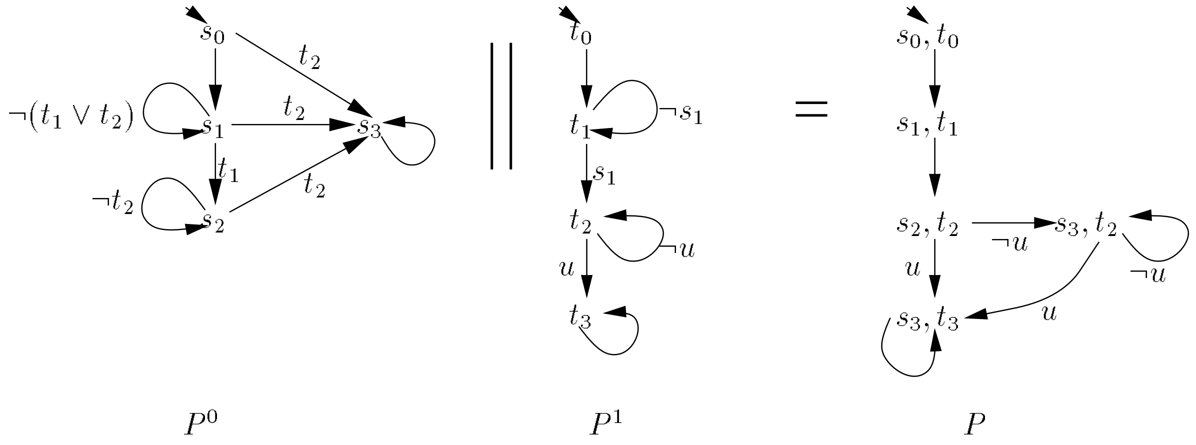

Figure 1: Example program

- If $P = P' \parallel P''$ then $S' \cap S'' = \emptyset$ for every $S' \in \mathcal{S}(P')$ and $S'' \in \mathcal{S}(P'')$; $\mathcal{S}(Q)$ stands for the set of states of Q.

Because of this constraint, program states can be taken from $2^{\mathcal{S}}$; one local state per component. For $S \in 2^{\mathcal{S}}$ and $\varphi \in \mathcal{L}(\mathcal{S})$, $S \models \varphi$ is defined as usual and $S \models s$ iff $s \in S$. If $P^0 = (\mathbf{S}^0, \ell^0, \mathbf{I}^0)$ and $P^1 = (\mathbf{S}^1, \ell^1, \mathbf{I}^1)$ are lts's, then $P^0 \parallel P^1$ determines an lts, $[\![P^0 \parallel P^1]\!] = (\mathbf{S}, \ell, \mathbf{I})$, as follows:

- $\mathbf{S} = \{S^0 \cup S^1 \mid S^0 \in \mathbf{S}^0, S^1 \in \mathbf{S}^1\}$ and $\mathbf{I} = \{I^0 \cup I^1 \mid I^0 \in \mathbf{I}^0, I^1 \in \mathbf{I}^1\}$,
- $\ell = \ell^0 \parallel \ell^1$ defined by $(\ell^0 \parallel \ell^1)(S,\ S') = \ell^0(S^0, S^{0\prime})[\mathbf{tt}/S^1] \wedge \ell^1(S^1, S^{1\prime})[\mathbf{tt}/S^0]$, where $[\mathbf{tt}/T]$ substitutes $\mathbf{tt}$ for every state in $T \in 2^{\mathcal{S}}$, $S = S^0 \cup S^1$, $S' = S^{0\prime} \cup S^{1\prime}$, $S^0, S^{0\prime} \subseteq \mathbf{S}^0$ and $S^1, S^{1\prime} \subseteq \mathbf{S}^1$ (remember that $\forall S^0 \subseteq \mathbf{S}^0, S^1 \subseteq \mathbf{S}^1 :\ S^0 \cap S^1 = \emptyset$).

As an example, consider Figure 1 again. The label of the $\{s2, t2\} - \{s3, t3\}$ transition is obtained as $\ell^0(s_2, s_3)[\mathbf{tt}/\{t2\}] \wedge \ell^1(t_2, t_3)[\mathbf{tt}/\{s_2\}] \ \equiv\ t2[\mathbf{tt}/\{t2\}] \wedge u[\mathbf{tt}/\{s_2\}] \ \equiv\ \mathbf{tt} \wedge u \ \equiv\ u$.

An lts $P = (\mathbf{S}, \ell, \mathbf{I})$ induces a 'standard' transition relation, R^ℓ, on the global state space, as follows. Let $S, S' \in \mathbf{S}$ be local states of P and consider global states $S \cup T$ and $S' \cup T'$ ($T, T' \in 2^{\mathcal{S}}$), i.e., whose P-components are S and S' resp. Then

$$\mathrm{R}^\ell(S \cup T, S' \cup T') \ \text{ iff } \ S \cup T \models \ell(S, S')\ .$$

For technical reasons, we also impose the constraint

- For any labeled transition system $(\mathbf{S}, \ell, I)$ the transition relation must be total:

$$\models \bigvee_{S' \in \mathbf{S}} \ell(S, S') \quad \text{for every } S \in \mathbf{S}\ .$$

Its consequence is that programs can never deadlock. I.e., in every global state there is at least one (globally) enabled transition in every process of the program.

Lemma 5.1 *With notation as above*

- $\|$ *is commutative and associative; i.e.,* $[\![P^0 \parallel P^1]\!] = [\![P^1 \parallel P^0]\!]$ *and* $[\![P^0 \parallel (P^1 \parallel P^2)]\!] = [\![(P^0 \parallel P^1) \parallel P^2]\!]$.
- *If* P *is a program then* $[\![P]\!]$ *has a total transition relation.*
- *For lts's* P^0 *and* P^1, $\mathrm{R}^{\ell} = \mathrm{R}^{\ell^0} \cap \mathrm{R}^{\ell^1}$

Proof. The first point is a direct consequence of commutativity and associativity of the operators $\cup$ and $\wedge$ in the definition of $[\![\cdot]\!]$. The second point is easily proven by induction over the structure of programs.

For the third point, let $P^0 = (\mathbf{S}^0, \ell^0, \mathbf{I}^0)$ and $P^1 = (\mathbf{S}^1, \ell^1, \mathbf{I}^1)$, $S^0, S^{0'} \in \mathbf{S}^0$, $S^1, S^{1'} \in \mathbf{S}^1$, and $T, T' \in 2^{\mathcal{S}}$ (such that for each $S \in \mathbf{S}^0 \cup \mathbf{S}^1$, $T \cap S = \emptyset$ and $T' \cap S = \emptyset$). Assume $\mathrm{R}^{\ell}(S^0 \cup S^1 \cup T, S^{0'} \cup S^{1'} \cup T')$. By the definitions of R^{ℓ} and $\ell^0 \parallel \ell^1$, this is equivalent to $S^0 \cup S^1 \cup T \models \ell^0(S^0, S^{0'})[\mathbf{tt}/S^1] \wedge \ell^1(S^1, S^{1'})[\mathbf{tt}/S^0]$. Because $S^i \models s^i$ for every $s^i \in S^i$ $(i = 0, 1)$, this implies $S^0 \cup S^1 \cup T \models \ell^0(S^0, S^{0'})$ and $S^0 \cup S^1 \cup T \models \ell^1(S^1, S^{1'})$. Furthermore, the reverse implication is obvious. By the definitions of R^{ℓ^0} and R^{ℓ^1}, $S^0 \cup S^1 \cup T \models \ell^0(S^0, S^{0'})$ and $S^0 \cup S^1 \cup T \models \ell^1(S^1, S^{1'})$ reduce to $\mathrm{R}^{\ell^0}(S^0 \cup S^1 \cup T, S^{0'} \cup S^{1'} \cup T')$ and $\mathrm{R}^{\ell^1}(S^0 \cup S^1 \cup T, S^{0'} \cup S^{1'} \cup T')$. $\square$

Hence, program semantics is well-defined.

Finally, we define when a program satisfies a $\mu L(\mathcal{S})$-formula:

Definition 5.2 *A program* P *with* $[\![P]\!] = (\mathbf{S}, \ell, \mathbf{I})$ *induces a Kripke structure* $\mathcal{M}_P = (2^{\mathcal{S}}, \mathrm{R}^{\ell}, \mathcal{I})$ *where* $\mathcal{I}(s) = \{S \in 2^{\mathcal{S}} \mid s \in S\}$ *for every* $s \in \mathcal{S}$. *For any* $\mu L(\mathcal{S})$ *formula* φ *define* $P \models \varphi$ *by* $\mathbf{I} \subseteq [\![\varphi]\!]_{\mathcal{M}_P}\bot$.

So, e.g., the program in Figure 1 would satisfy $P \models \mu x.((s_2 \wedge t_2) \vee \Diamond x)$; i.e., the s_2, t_2-state is reachable. Indeed, $[\![\mu x.((s_2 \wedge t_2) \vee \Diamond x)]\!]\bot$ includes the initial state of P.

6 Program abstractions

Although all the ingredients are in place to build a symbolic model checker, experience shows that in practice it would have limited applicability. The main problem is the amount of memory needed to store the BDDs that describe the program's transition relation and that describe the intermediate results during the computation of $[\![\varphi]\!]_{\mathcal{M}}$. Also, since the time complexity of BDD operations are low order polynomial in the size of the BDDs they operate on, model checking time is affected as well.

The abstract interpretation framework suggests two ways to reduce the time and space requirements. One can replace the set $(2^{\mathcal{S}})$ of concrete states by a smaller set 2^D of abstract states so that fewer BDD variables are needed to represent subsets of D. Also, although given an abstract domain the concrete functions are often replaced by their optimal abstract interpretations, one can use safe approximations that are easier to compute instead.

Next, observe that a large set does not imply that the BDD representation of its characteristic predicate has a large number of BDD nodes as well. The set of all states is a case in point: it is represented by the 1 node BDD $\ulcorner\mathbf{tt}\urcorner$. Thus our aim should not be to reduce the size of the state sets but to reduce the number of nodes in the BDDs representing these sets.

Now, given a program P (or rather the Kripke structure induced by it), the representations of the semantics of the propositions are very small, although the represented sets can be quite large. Indeed, each proposition describes just a single local state of a process in P. The computation of conjunction and disjunction are of polynomial time and space and the computation of the fixpoints is just an iteration. The point were the BDD representations of the intermediate sets become large is during the computation of the relational product with the transition relation—computing $\tau_\diamond$ respectively $\tau_\Box$. From these observations we conclude that the set of abstract states D need not be different (smaller) than $\mathcal{S}$ in order to reduce time and space during BDD-based model checking, but that the abstract operators ${}^\alpha\tau_\diamond$ and ${}^\alpha\tau_\Box$ should lead to sets with smaller BDD-representations than the original ones.

Thus our abstract interpretation, ${}^\alpha[\![\,\cdot\,]\!]$, of $\mu L(\mathcal{S})$ takes ${}^\alpha\mathcal{V} = 2^{\mathcal{S}}$ as abstract domain and replaces only the $\tau_\Box$ and $\tau_\diamond$ functions by safe abstract interpretations ${}^\alpha\tau_\Box$ and ${}^\alpha\tau_\diamond$ in the definition on page 4. Hence, the other operations are 'replaced' by their optimal interpretations.

Lemma 6.1 *If ${}^\alpha\tau_\Box$ and ${}^\alpha\tau_\diamond$ are safe interpretations, i.e., if ${}^\alpha\tau_\Box(S) \subseteq \tau_\Box(S)$ and ${}^\alpha\tau_\diamond(S) \subseteq \tau_\diamond(S)$ for $S \in 2^{\mathcal{S}}$ then*

$$ {}^\alpha[\![\varphi]\!]e \subseteq [\![\varphi]\!]e \quad \textit{for any } \varphi \textit{ and } e \ . $$

Proof. Straightforward. The essential point is that all operations in the definition of $[\![\,\cdot\,]\!]$ on page 4 are monotonic (except for the negation—but it is only applied to propositions). □

Hence, using such abstractions, we may erroneously conclude that $P \not\models \varphi$, but if $\mathbf{I} \subseteq {}^\alpha[\![\varphi]\!]\bot$ then $P \models \varphi$ holds.

In the concrete case $\tau_\Box$ and $\tau_\diamond$ are computed from the transition relation R as relational products. This is possible for the abstract versions as well, except that two different transition relations are needed:

Lemma 6.2 $\mathsf{pre}_{\mathrm{R}'} \subseteq \mathsf{pre}_{\mathrm{R}}$ *if* $\mathrm{R}' \subseteq \mathrm{R}$ *and* $\widetilde{\mathsf{pre}}_{\mathrm{R}''} \subseteq \widetilde{\mathsf{pre}}_{\mathrm{R}}$ *if* $\mathrm{R} \subseteq \mathrm{R}''$

Write $\ell' \preceq \ell$ in case $\mathrm{R}^{\ell'} \subseteq \mathrm{R}^{\ell}$.

We can, in fact, compute abstractions compositionally:

Lemma 6.3 *If $\ell_i' \preceq \ell_i$ $(i = 0, 1)$ then $\ell_0' \parallel \ell_1' \preceq \ell_0 \parallel \ell_1$.*

Proof. Follows from the third point of Lemma 5.1. □

For, e.g., $P^0 \| P^1$, $\ulcorner \mathsf{pre}_{\mathrm{R}^{\ell^0 \| \ell^1}}(C) \urcorner$ would be computed as the relational product $\exists \vec{x}'(\ulcorner R^{\ell^0} \urcorner(\vec{x}, \vec{x}') \wedge \ulcorner R^{\ell^1} \urcorner(\vec{x}, \vec{x}') \wedge \ulcorner C \urcorner(\vec{x}'))$ and the global transition relation $R^{\ell^0 \| \ell^1}$ is not explicitly represented by a BDD.

6.1 Abstraction from interactions

A relational product grows large if there is much interaction between the various processes[7]. Accordingly, the idea behind our abstractions is to reduce such interactions.

So, a universal abstraction replaces the (satisfiable) constraints on some transitions by **tt**, while an existential abstraction removes some edges with non-trivial constraints. As a consequence, the abstracted part is no longer influenced by the other processes. Also, its influence on the non-abstracted processes is 'simplified'.

Definition 6.4 *Let P be an lts $(\mathbf{S}, \ell, \mathbf{I})$. An abstraction of P is determined by selecting a subset $S_A \subseteq \mathbf{S}$. Define ${}^{\alpha}\ell_{\Box}(S, S') \stackrel{D}{=} \mathbf{tt}$ if $S, S' \in S_A$ and $\not\models \neg\ell(S, S')$; define ${}^{\alpha}\ell_{\Diamond}(S, S') \stackrel{D}{=} \mathbf{ff}$ if $S, S' \in S_A$ and $\not\models \ell(S, S')$. Elsewhere, ${}^{\alpha}\ell_{\Box}$ and ${}^{\alpha}\ell_{\Diamond}$ coincide with ℓ. Then the universal abstraction is ${}^{\alpha}P_{\Box} \stackrel{D}{=} (\mathbf{S}, {}^{\alpha}\ell_{\Box}, \mathbf{I})$; the existential abstraction is ${}^{\alpha}P_{\Diamond} \stackrel{D}{=} (\mathbf{S}, {}^{\alpha}\ell_{\Diamond}, \mathbf{I})$*

The condition $\not\models \neg\ell(S, S')$ in the definition of ${}^{\alpha}\ell_{\Box}$ is included to ensure that no new transition is added between two states S and S' if there was no transition at all (i.e, $\models \neg\ell(S, S')$ would hold for these states) before. Similarly, the condition $\not\models \ell(S, S')$ in the definition of ${}^{\alpha}\ell_{\Diamond}$ prevents from disappearing those transitions which have tautological constraints.

We shall often write ${}^{\alpha}\mathrm{R}^{\ell}_{\Box}$ instead of $\mathrm{R}^{{}^{\alpha}\ell_{\Box}}$, etc.

Lemma 6.5 *With notation as above, ${}^{\alpha}\ell_{\Diamond} \preceq \ell$ and $\ell \preceq {}^{\alpha}\ell_{\Box}$.*

How good are these abstractions; i.e., can we often find abstractions that both offer substantial reduction and allow many properties to be verified? This is difficult to assess in general but, heuristically, the universal abstraction (${}^{\alpha}\ell_{\Box}$) is often more usable than the existential one. This is because in the lattice of relations, program relations tend to be closer to the empty relation than they are to the full one. Indeed, a program describes the allowed behavior rather than those behaviors that are disallowed. E.g., a function for computing, say, the square root of its input, is usually not implemented by a program that admits arbitrary behavior only constrained by the requirement that it computes the square root. Quite the opposite: a square root program should only preform those actions that are necessary for its correct functioning. This means that there tends to be ample room to add transitions to a relation—as a universal abstraction does—without hitting the full relation, but less so to remove them. The effect of this is that it tends to be easier to find a universal abstraction that preserves the truth of a universal property (i.e., a property that does not use $\Diamond$-modalities) than it is to find a usable existential abstraction that preserves an existential property. Indeed, from the 13 properties that use $\Diamond$-modalities on page 15 none could be verified using existential abstractions alone.

[7] This observation can be made more precise if we look at the particular way states are encoded.

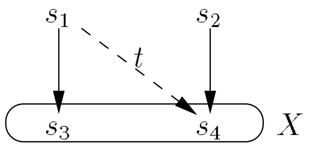

Figure 2: Universal abstraction functioning as an existential one

6.2 A better existential abstraction

Specifically, as we compute ${}^{\alpha}\mathrm{R}^{\ell}_{\Box}$ anyway, is it possible to use ${}^{\alpha}\mathrm{R}^{\ell}_{\Box}$ for existential abstractions as well? First, observe that on the **un**abstracted part of a process's transition relation ${}^{\alpha}\mathrm{R}^{\ell}_{\Diamond} = \mathrm{R} = {}^{\alpha}\mathrm{R}^{\ell}_{\Box}$; on the abstracted part we have ${}^{\alpha}\mathrm{R}^{\ell}_{\Diamond} \subseteq \mathrm{R} \subseteq {}^{\alpha}\mathrm{R}^{\ell}_{\Box}$. However, as long as $\mathsf{pre}_{{}^{\alpha}\mathrm{R}^{\ell}_{\Box}}(S) \subseteq \mathsf{pre}_{\mathrm{R}^{\ell}}(S)$ for sets S that arise as intermediate results, this will cause no problems. That this is possible is illustrated in Figure 2. The label on the dashed transition will be removed by a universal abstraction ${}^{\alpha}\mathrm{R}^{\ell}_{\Box}$ (for $S_A = \{s_i \mid i = 1 \ldots 4\}$). However, obviously $\mathsf{pre}_{{}^{\alpha}\mathrm{R}^{\ell}_{\Box}}(X) \subseteq \mathsf{pre}_{\mathrm{R}^{\ell}}(X) = \{s_1, s_2\}$. Lemma 6.7 below characterizes these situations.

First we need a notion of *independency*. Intuitively, a set of global states X is *independent* of a set of local states S of a process P, if the other processes being in a state in X does not depend on P being in a specific local state in $X \cap S$.

For example, considering Figure 3 on page 12, let $\mathbf{S}$ be the local states of the left-hand-side process, and let $S = \{s_0, s_1, s_2\}$. Then $\{\ \{s_0, t_0\}, \{s_0, t_1\}, \{s_1, t_1\}, \{s_2, t_1\}\ \}$ is not independent of S because the local state t_0 only occurs with s_0 (which is in S) but not with the other local states in S. Both $\{\ \{s_0, t_1\}, \{s_1, t_1\}, \{s_2, t_1\}\ \}$ and $\{\ \{s_0, t_0\}, \{s_1, t_0\}, \{s_2, t_0\}, \{s_0, t_1\}, \{s_1, t_1\}, \{s_2, t_1\}\ \}$ are independent however.

Definition 6.6 *Let $\mathbf{S} \subseteq \mathcal{S}$ be the set of local states of a single process and let $S \subseteq \mathbf{S}$.*

- *For a set $X \subseteq 2^{\mathcal{S}}$ of global states, X is* independent *of S if for each $x \in X$ with $x \cap \mathbf{S} \in S$ and $x \cap (\mathcal{S} \setminus \mathbf{S}) = t$, we have $\forall s \in S:\ s \cup t \in X$.*
- $\mathrm{Suc}(S) = \{s' \in \mathbf{S} \mid \exists s \in S\ \not\models \neg\ell(s, s')\}$.

Now, the situations as in Figure 2 can be characterized:

Lemma 6.7 *Let S_A determine an abstraction of the lts $(\mathbf{S}, \ell, \mathbf{I})$. For any $X \subseteq 2^{\mathcal{S}}$ which is independent of $S_A \cup \mathrm{Suc}(S_A)$ we have $\mathsf{pre}_{{}^{\alpha}\mathrm{R}^{\ell}_{\Box}}(X) \subseteq \mathsf{pre}_{\mathrm{R}^{\ell}}(X)$.*

Proof. Let $y \in \mathsf{pre}_{{}^{\alpha}\mathrm{R}^{\ell}_{\Box}}(X)$. So, we can choose $x \in X$ such that ${}^{\alpha}\mathrm{R}^{\ell}_{\Box}(y, x)$, whence $y \models {}^{\alpha}\ell_{\Box}(y \cap \mathbf{S}, x \cap \mathbf{S})$.

If $x \cap \mathbf{S} \notin S_A$ or $y \cap \mathbf{S} \notin S_A$ then, by definition of ${}^{\alpha}\ell_{\Box}$, ${}^{\alpha}\ell_{\Box}(y, x) = \ell(y, x)$ so that $\mathrm{R}^{\ell}(y, x)$, whence $y \in \mathsf{pre}_{\mathrm{R}^{\ell}}(X)$.

Suppose $x \cap \mathbf{S} \in S_A$ and $y \cap \mathbf{S} \in S_A$. If in addition $\mathrm{R}^\ell(y,x)$, then we are ready. Suppose $\neg\mathrm{R}^\ell(y,x)$. So, $y \models {}^\alpha\ell_\Box(y \cap \mathbf{S}, x \cap \mathbf{S})$ but not $y \models \ell(y \cap \mathbf{S}, x \cap \mathbf{S})$. We look for $x' \in X$ such that $y \models \ell(y \cap \mathbf{S}, x' \cap \mathbf{S})$. As X is independent of $S_A \cup \mathrm{Suc}(S_A)$ while $x \cap \mathbf{S} \in S_A \cup \mathrm{Suc}(S_A)$, we can choose $x' \in X$ so that $x' \cap (\mathcal{S} \setminus \mathbf{S}) = x \cap (\mathcal{S} \setminus \mathbf{S})$ and $x' \cap \mathbf{S} \in S_A \cup \mathrm{Suc}(S_A)$. Furthermore, by totality of ℓ, we can choose x' in such a way that $y \models \ell(y \cap \mathbf{S}, x' \cap \mathbf{S})$. So, $\mathrm{R}^\ell(y,x')$ holds and, hence, $y \in \mathsf{pre}_{\mathrm{R}^\ell}(X)$. $\Box$

For a given S_A as above, define $\Gamma\colon 2^{2^{\mathcal{S}}} \to 2^{2^{\mathcal{S}}}$ such that $\Gamma(\mathbf{T})$ is the (unique) largest subset of $\mathbf{T}$ that is independent of $S_A \cup \mathrm{Suc}(S_A)$. So, for the example in Figure 3, $\Gamma(\{\ \{s_i,t_j\} \mid (i < 3 \wedge j = 0) \vee (i < 4 \wedge j = 1)\ \}) = \{\ \{s_i,t_j\} \mid i < 4 \wedge j = 1\ \}$. We have

Corollary 6.8 *For any $X \subseteq 2^{\mathcal{S}}$, $\mathsf{pre}_{{}^\alpha\mathrm{R}^\ell_\Box}(\Gamma(X)) \subseteq \mathsf{pre}_{\mathrm{R}^\ell}(X)$.*
Hence, $\mathsf{pre}_{{}^\alpha\mathrm{R}^\ell_\Diamond} \cup (\mathsf{pre}_{{}^\alpha\mathrm{R}^\ell_\Box} \circ \Gamma)$ is a safe approximation of $\tau_\Diamond$.

Figure 3 shows an example in which both existential pre-abstractions are needed. Write ${}^\alpha\mathsf{pre}_\Diamond$ and ${}^\alpha\mathsf{pre}_\Box$ for the approximations to pre induced by Definition 6.4; write ${}^\alpha\overline{\mathsf{pre}}_\Diamond$ for ${}^\alpha\mathsf{pre}_\Diamond \cup ({}^\alpha\mathsf{pre}_\Box \circ \Gamma)$. The states within the dashed boundary are abstracted. So, ${}^\alpha\mathrm{R}^\ell_\Box$ will remove the t_1 label between s_1 and s_2 and the labels on the self-loops at s_1 and s_2 while ${}^\alpha\mathrm{R}^\ell_\Diamond$ will remove the corresponding transitions. In computing ${}^\alpha[\![\Phi]\!]$ we use the iterative scheme of page 5. We start with ${}^\alpha[\![\Phi]\!]^1\bot = {}^\alpha[\![t_3]\!] = \{S \subseteq \mathcal{S} \mid t_3 \in S\}$. For the set on the right-hand-side we just write t_3; this being its characteristic predicate. As ${}^\alpha\mathsf{pre}_\Diamond(t_3) = t_2 \vee (t_3 \wedge (s_0 \vee s_3))$, ${}^\alpha\mathsf{pre}_\Box(t_3) = t_2 \vee t_3$ and $\Gamma(t_3) = t_3$, we obtain ${}^\alpha[\![\Phi]\!]^2\bot = t_3 \vee {}^\alpha\overline{\mathsf{pre}}_\Diamond(t_3) = t_3 \vee {}^\alpha\mathsf{pre}_\Diamond(t_3) \vee {}^\alpha\mathsf{pre}_\Box(\Gamma(t_3)) = t_2 \vee t_3$. Next, we compute ${}^\alpha[\![\Phi]\!]^3\bot = t_3 \vee {}^\alpha\overline{\mathsf{pre}}_\Diamond(t_2 \vee t_3)$. Because both ${}^\alpha\mathsf{pre}_\Diamond$ and Γ distribute over disjunction, this equals $t_3 \vee {}^\alpha\mathsf{pre}_\Diamond(t_2) \vee {}^\alpha\mathsf{pre}_\Diamond(t_3) \vee {}^\alpha\mathsf{pre}_\Box(\Gamma(t_2) \vee \Gamma(t_3))$. As ${}^\alpha\mathsf{pre}_\Diamond(t_2) = \emptyset$, $\Gamma(t_2) = t_2$ and ${}^\alpha\mathsf{pre}_\Box(t_2 \vee t_3) = (t_1 \wedge s_1) \vee t_2 \vee t_3$, we get $(t_1 \wedge s_1) \vee t_2 \vee t_3$ as the result. Now, $\Gamma(t_2 \vee t_3 \vee (s_1 \wedge t_1)) = t_2 \vee t_3$ so that for computing ${}^\alpha\overline{\mathsf{pre}}_\Diamond$ at $t_2 \vee t_3 \vee (s_1 \wedge t_1)$, ${}^\alpha\mathsf{pre}_\Box$ does not contribute. However, we do have $\{s_0,t_0\} \in {}^\alpha\mathsf{pre}_\Diamond(\{s_1,t_1\})$ and by this $\{s_0,t_0\} \in {}^\alpha[\![\Phi]\!]^4\bot$.

7 Experimental results

We applied the abstraction techniques developed above to verify a production cell [DHKS95]. Figure 4 depicts the system to model.

The task of the production cell is to press metal blanks. The production cell consists of two conveyor belts (the *feed belt* in the front and the *deposit belt* in the back of the picture), a rotary table (right to the feed belt), a press (to the right of the picture), a robot with two independent movable arms (between rotary table and press) and a crane (under the top of the picture). In addition, we assume that there is a user who puts a metal blank on the feed belt.

The controller of the production cell notifies the user when a new metal blank can be put on the feed belt. This will transport it to the rotary table. The table will rotate and lift the blank up, so it can be picked up by the first robot arm. The robot puts the blank into the press. After pressing the blank, the second robot arm fetches the pressed blank and puts it

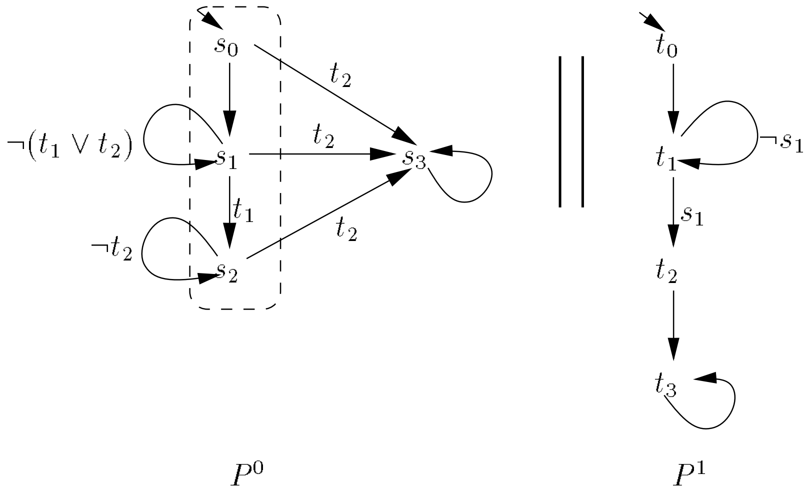

Figure 3: $({}^{\alpha}P^0 \parallel P^1) \models \Phi$ with $\Phi \overset{D}{=} \mu x.(t_3 \vee \Diamond x)$

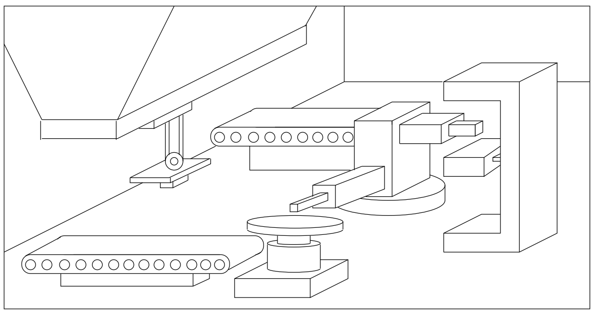

Figure 4: A production cell

on the deposit belt, which will transport the blank to the crane. The crane closes the cycle and moves the blank from the deposit belt to the feed belt in order to press the blank again.

We have modeled the production cell with StateCharts [Har87]. StateCharts is a graphical specification language. It is based on automata that can be composed in parallel and that form a hierarchy. The parallel composition leads to a (synchronous) lock-step semantics, where parallel components have to make a step if they can, otherwise they stutter. Hence, the induced transition relation is total, which is required by our framework. The parallel components of the production cell communicate by identifying states of other components, e.g., a transition from one state to another one is only possible if another process is in a certain state. By this mechanism the parallel components will be synchronized. This fragment of Statecharts corresponds to the programs we introduced in this paper. A detailed description of the translation scheme for full StateCharts into BDDs can be found in [HK94].

Since the production cell contains several physical components which have to be controlled independently, the general structure of the StateChart specification is a parallel decomposition of the controller w.r.t. the physical components. The single sub-controllers for the components are again decomposed for the single task the sub-controller has to satisfy, e.g., the sub-controller for the rotary table has again sub-controllers for the rotation and the vertical movement.

The verification of the production cell distinguishes between safety and liveness properties. Typical safety properties are that the table does not rotate too far in one of the two directions thus avoiding damage. For all controller components these kind of safety properties can be checked in isolation; i.e., the property that the table only rotates in its specified area is independent of the behavior of the robot. An abstraction of all sub-controllers except the controller for the table as described above preserves the necessary information to verify the safety property.

Liveness properties are more complex to verify because they are in general not restricted to the behavior of some few processes. For example the property that along all executions eventually the crane puts a blank on the front belt needs all processes, because this property simulates a complete run through the production cell. Using abstractions one has to split such a specification into several parts which together imply the original property but each of which only needs part of the model. However, we are able to verify the several parts individually.

The verification results are shown in the Tables 2–4 below. The first column gives the property that is checked. It is expressed using the CTL operators G ("globally", i.e., for all states along a path), F ("eventually", i.e., there exists a state along a path) as well as the Unless operator (also called weak until) defined by $\varphi_1 \mathsf{Unless}\, \varphi_2 \equiv (\varphi_1 \mathsf{U} \varphi_2) \vee \mathsf{G}\varphi_1$, where U is the until operator. A and E denote the universal resp. existential path quantifiers. These formulae are easily translated into the μ-calculus. For example, $\mathsf{A}(\varphi_1 \mathsf{Unless}\, \varphi_2) \equiv \nu x.(\varphi_2 \vee (\varphi_1 \wedge \Box x))$. The meanings of the propositions occurring in these formulae are explained in Table 1.

The second column indicates which parts of the system have been abstracted. The entire system consists of a parallel composition of processes modelling the feed belt, rotary table, robot, press, deposit belt, crane and user; in the tables these are identified by their first letters. We always abstract from entire processes. E.g., an entry *rpd* in the second column means that robot, press and deposit belt have been abstracted from. Because of the tight interaction of the processes this means that the existential abstractions of the processes remove *all* transitions. Indeed, none of these properties could be verified without the optimization from Section 6.2.

The third and fourth columns give the maximal number of BDD nodes that are needed to verify the property without resp. with application of our abstraction technique; the last column is the reduction factor between these numbers.

Table 3 shows that formulae with existential path quantifiers can also be verified while Table 4 gives results for formulae with mixed path quantifiers.

The most notable feature of these tables is that we are able to achieve a 17-fold reduction on the average in the number of BDD nodes needed to verify the specifications.

proposition	meaning
init	the entire system is in its initial state
f-go	the feed belt is running
f-in	there is a metal blank on the right end of the feed belt (near the rotary table)
t-hpos-f	the horizontal position of the rotary table is such that it can accept a metal blank from the feed belt
t-empty	there is no metal blank on the rotary table
t-full	there is a metal blank on the rotary table
t-vpos-bot	the vertical position of the rotary table is at its lowest point
t-vpos-err	the vertical position of the rotary table is such that it may cause damage

Table 1: Meanings of the propositions.

8 Conclusions

We have developed a compositional abstract interpretation scheme for model checking full propositional μ-calculus over parallel processes. The technique shows quite good performance in non-trivial examples, resulting in substantial reductions in the number of BDD nodes. The programming model that we consider is general and underlies many programming languages, hence we expect our results to be generally applicable. We have extended these abstraction techniques to directly apply to a larger subset of the StateChart language; notably event generation and event triggers (including triggering on the absence of events).

Future work includes the refinement of these techniques to cover a larger class of program abstractions.

property	abstract from	without abstraction	with	red. fact.
init $\Rightarrow$ AG $\neg$t-vpos-err	*frpdcu*	36300	2210	16
init $\Rightarrow$ AF t-full	*rpd*	48700	5720	9
init $\Rightarrow$ AF (f-in $\wedge$ f-go)	*trpd*	43700	2840	15
init $\Rightarrow$ AG (f-in $\Rightarrow$ A (f-in Unless (f-in $\wedge$ t-vpos-bot $\wedge$ t-hpos-f)))	*rpdc*	36900	2840	13
init $\Rightarrow$ AG ((t-vpos-bot $\wedge$ t-hpos-f $\wedge$ t-empty) $\Rightarrow$ A ((t-vpos-bot $\wedge$ t-hpos-f) Unless (f-in $\wedge$ f-go $\wedge$ t-vpos-bot $\wedge$ t-hpos-f)))	*rpdcu*	84100	3680	23
init $\Rightarrow$ AF (t-vpos-bot $\wedge$ t-hpos-f $\wedge$ t-empty)	*frpdcu*	38400	2880	13
init $\Rightarrow$ AG ((t-vpos-bot $\wedge$ t-hpos-f $\wedge$ f-go $\wedge$ f-in) $\Rightarrow$ AF t-full)	*frpdcu*	48700	2590	19

Table 2: Results for universal properties.

property	abstract from	without abstraction	with	red. fact.
init $\Rightarrow$ EG $\neg$t-vpos-err	*frpdcu*	36500	2220	16
init $\Rightarrow$ EF t-full	*rpd*	52500	9540	6
init $\Rightarrow$ EF (f-in $\wedge$ f-go)	*trpd*	101000	3190	32
init $\Rightarrow$ EG (f-in $\Rightarrow$ E (f-in Unless (f-in $\wedge$ t-vpos-bot $\wedge$ t-hpos-f)))	*rpdc*	39100	4060	10
init $\Rightarrow$ EG ((t-vpos-bot $\wedge$ t-hpos-f $\wedge$ t-empty) $\Rightarrow$ E ((t-vpos-bot $\wedge$ t-hpos-f) Unless (f-in $\wedge$ f-go $\wedge$ t-vpos-bot $\wedge$ t-hpos-f)))	*rpdcu*	42500	3200	13
init $\Rightarrow$ EF (t-vpos-bot $\wedge$ t-hpos-f $\wedge$ t-empty)	*frpdcu*	84500	2500	34
init $\Rightarrow$ EG ((t-vpos-bot $\wedge$ t-hpos-f $\wedge$ f-go $\wedge$ f-in) $\Rightarrow$ EF t-full)	*frpdcu*	53600	2310	23

Table 3: Results for existential properties.

property	abstract from	without abstraction	with	red. fact.
init $\Rightarrow$ AG ((t-vpos-bot $\wedge$ t-hpos-f $\wedge$ t-empty) $\Rightarrow$ E ((t-vpos-bot $\wedge$ t-hpos-f) Unless (f-in $\wedge$ f-go $\wedge$ t-vpos-bot $\wedge$ t-hpos-f)))	*rpdcu*	42500	3160	13
init $\Rightarrow$ EG ((t-vpos-bot $\wedge$ t-hpos-f $\wedge$ t-empty) $\Rightarrow$ A ((t-vpos-bot $\wedge$ t-hpos-f) Unless (f-in $\wedge$ f-go $\wedge$ t-vpos-bot $\wedge$ t-hpos-f)))	*rpdcu*	84100	3420	25
init $\Rightarrow$ EG ((t-vpos-bot $\wedge$ t-hpos-f $\wedge$ f-go $\wedge$ f-in) $\Rightarrow$ AF t-full)	*frpdcu*	50000	2450	20
init $\Rightarrow$ AG ((t-vpos-bot $\wedge$ t-hpos-f $\wedge$ f-go $\wedge$ f-in) $\Rightarrow$ EF t-full)	*frpdcu*	52500	2210	24

Table 4: Results for mixed properties.

References

[LGS+95] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, Vol. 6, Iss. 1, Januari 1995.

[BCM+92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–171, June 1992. Special Issue: Selections from 1990 IEEE Symposium on Logic in Computer Science.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, August 1986.

[Bry92] Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[CBM90] O. Coudert, C. Berthet, and J. C. Madre. Verifying temporal properties of sequential machines without building their state diagrams. In R. P. Kurshan and E. M. Clarke, editors, *Proceedings of the 1990 Workshop on Computer-Aided Verification*, June 1990.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings 4th ACM Symposium on Principles of Programming Languages (POPL)*, pages 238–252, Los Angeles, California, 1977.

[CGL94] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM TOPLAS*, 16(5):1512–1542, September 1994.

[CR94] R. Cleaveland and J. Riely. A testing-based abstractions for value-passing systems. In B. Jonsson and J. Parrow, editors, *CONCUR'94: Concurrency Theory*, Lecture Notes in Computer Science 836, pages 417–432. Springer-Verlag, August 1994.

[Dam94] M. Dam. CTL* and ECTL* as fragments of the modal μ-calculus. *Theoretical Computer Science*, 126(1):77–97, 1994.

[DGG94] D. Dams, O. Grumberg, and R. Gerth. Abstract interpretation of reactive systems: Abstractions preserving $\forall$CTL*, $\exists$CTL* and CTL*. In E.-R. Olderog, editor, *Proceedings of the IFIP WG2.1/WG2.2/WG2.3 Working Conference on Programming Concepts, Methods and Calculi (PROCOMET)*, IFIP Transactions, Amsterdam, June 1994. North-Holland/Elsevier. Full version available as Computing Science Note 95/16, Eindhoven University of Technology, Dept. of Math. and Computing Science.

[DHKS95] W. Damm, H. Hungar, P. Kelb, and R. Schlör. Using graphical specification languages and symbolic model checking in the verification of a production cell. In C. Lewerenz and T. Lindner, editors, *Formal Development of Reactive Systems: Case Study "Production Cell"*, volume 891 of *Lecture Notes in Computer Science*. Springer, 1995.

[Har87] David Harel. StateCharts: A Visual Formalism for Complex Systems. *Science of Computer Programming 8*, 1987.

[HK94] J. Helbig and P. Kelb. An OBDD-Representation of StateCharts. In *Proceedings European Design Automation Conference (EDAC)*, 1994.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[Tar55] A. Tarski. A Lattice-Theoretical Fixpoint Theorem and its Applications. In *Pacific Journal of Mathematics 5*, pages 285–309, 1955.