Compositional Checking of Satisfaction*

Henrik Reif Andersen Glynn Winskel

Department of Computer Science, Aarhus University, Denmark

Abstract: We present a compositional method for deciding whether a process satisfies an assertion. Assertions are formulae in a modal ν-calculus, and processes are drawn from a very general process algebra inspired by CCS and CSP. Well-known operators from CCS, CSP, and other process algebras appear as derived operators.

The method is *compositional in the structure of processes* and works purely on the syntax of processes. It consists of applying a sequence of *reductions*, each of which only take into account the top-level operator of the process. A reduction transforms a satisfaction problem for a composite process into equivalent satisfaction problems for the immediate subcomponents.

Using process variables, systems with undefined subcomponents can be defined, and given an overall requirement to the system, *necessary and sufficient conditions* on these subcomponents can be found. Hence the process variables make it possible to specify and reason about what are often referred to as *contexts*, *environments*, and *partial implementations*.

As reductions are algorithms that work on syntax, they can be considered as forming a bridge between traditional non-compositional model checking and compositional proof systems.

1 Introduction

In this paper we present a compositional method for deciding whether a finite state process satisfies a specification. Processes will be described in a very general and rich process algebra, which includes common operators from process algebras as CCS and CSP. This algebra contains primitive operators to reflect sequentiality (by the well-known operation of prefixing), nondeterministic choice, asynchronous and synchronous parallel composition, recursion, relabelling, and restriction. Specifications will be drawn from a modal ν-calculus with negation, in which a variety of properties can be specified. These include the usual *liveness*, *safety*, and *fairness* properties, as well as all operators from ordinary linear and branching time temporal logics (see e.g. [Sti91] and [Dam90]).

The method we advocate is *compositional in the structure of processes* and works purely on the syntactical level without any explicit references to the underlying transition system. Compositionality is important for at least the following two reasons. Firstly, it makes the verification *modular*, so that when changing a part of a system only the part of the verification concerning that particular component must be redone. Secondly, when designing a system or *synthesizing* a process the compositionality makes it possible to have undefined parts of a process and still be able to reason about it. For instance, it might be possible to reveal inconsistencies in the specification or prove that with the choices already taken in the design no component supplied for the missing parts will ever be able to make the overall system satisfy the original specification.

This approach is unlike traditional model checking where a transition system model of a process is built and the specification formula is checked by applying some algorithm to the transition system. There are several versions of this basic idea in the literature, e.g. Emerson and Lei [EL86], Clarke et al [CES86], Stirling and Walker [SW89], Larsen [Lar88], Winskel [Win89], Cleaveland [Cle90], and Arnold and Crubille [AC88]. Recently there have been attempts to extend some of these methods based on transition systems to compositional methods by Clarke,

*This work is supported by the ESPRIT Basic Research Actions CEDISYS and CLICS, and for the first author also by the Danish Natural Science Research Council.

Long, and McMillan [CLM89] and Larsen and Xinxin [LX90], but none of these are compositional in the structure of processes.

Our method consists of applying a sequence of *reductions* each of which removes the top-most operator of the process, i.e. a reduction transforms a satisfaction problem for a composite process to satisfaction problems for the immediate subcomponents of the process – without inspecting these. Starting with a process term one can repeatedly use the reductions until a trivial process, for which satisfaction is easily decided, or a variable remains.

2 The Languages

2.1 Syntax

Assume given a set of *state names Nam*, and a finite set of *actions Act*. Processes are denoted by syntactic terms t constructed from the following grammar:

$$t ::= nil \mid at \mid t_0 + t_1 \mid t_0 \times t_1 \mid t \restriction \Lambda \mid t\{\Xi\} \mid rec\ P.t \mid P,$$

where P is an element in *Nam*, i.e. a state identifier. The usual notion of free and bound will apply to state identifiers P, so that P will be bound in *rec P.t* but free in $P + nil$.

Nil is the inactive process, and at is the usual prefix and $t_0 + t_1$ the usual sum operations known from CCS. The product term $t_0 \times t_1$ denotes a very general kind of parallel composition which allows the components t_0 and t_1 to proceed both synchronously and asynchronously. The exact semantics is defined below.

A state identifier P in the body of *rec P.t* works as a *recursion point*, and in effect will behave as the normal recursion in CCS: A term *rec P.t* has the same behaviour as the *unfolded* term $t[rec\ P.t/P]$ (the result of substituting *rec P.t* for all free occurrences of P in t). We impose the syntactic restriction on recursive terms, that no product must appear in the body, which ensures all definable processes are finite state, and for technical reasons we also require every occurrence of P in *rec P.t* to be *strongly guarded*, i.e. appear immediately under a prefix.

In the prefix at, a denotes an action in *Act*. For a given set of actions *Act* we define a set of *composite actions*. Let $*$ be a distinguished symbol not contained in *Act*. The symbol $*$ is called the *idling action* and interpreted as 'no action' or 'inaction'. Define Act_* to be the least set including $Act \cup \{*\}$ and such that $\alpha, \beta \in Act_*$ implies $\alpha \times \beta \in Act_*$ taking $* \times * = *$. Now $\Xi : Act_* \rightharpoonup Act$, is a *relabelling* which is a partial function, with finite domain, mapping non-idling actions to non-idling actions. This relabelling can be extended to a total function on Act_* by taking it to behave as the identity outside the domain. The term $t \restriction \Lambda$ is a *restriction* where Λ is a finite subset of Act_*.

Properties of processes are denoted by assertions A from a modal ν-calculus:

$$A ::= \neg A \mid A_0 \vee A_1 \mid \langle\alpha\rangle A \mid X \mid \nu X.A \mid (t : A),$$

where X ranges over a set of assertion variables. In the maximal fixed-point formula $\nu X.A$ any free occurrence of X must be within an even number of negations in order to guarantee the existence of a unique maximal fixed-point. The action name α belongs to the set of composite actions Act_*. The *correctness assertion* $(t : A)$ denotes true if t satisfies A and false otherwise. An assertion is said to be *pure* if it does not contain any correctness assertions.

A lot of derived operators can easily be defined in terms of the core language and will be used throughout the paper:

$$\begin{array}{lcllcl} [\alpha]A & = & \neg\langle\alpha\rangle\neg A & \mu X.A & = & \neg\nu X.\neg A[\neg X/X] \\ T & = & \nu X.X & A \rightarrow B & = & \neg A \vee B \\ F & = & \neg T & A \leftrightarrow B & = & (A \rightarrow B) \wedge (B \rightarrow A) \end{array}$$

Here we have used the notation $A[B/X]$ which denotes the assertion resulting from substituting B for all free occurrences of X in A. We will say that an assertion A is *closed* if it contains no free variables. Furthermore for a finite set $K \subseteq Act_*$ we define $\langle K\rangle A = \bigvee_{\kappa\in K}\langle\kappa\rangle A$ where disjunction over an empty set gives false (F).

The correctness assertions $(t : A)$ are atoms in a propositional logic which will be used to express reductions. A grammar for the logic is:

$$L ::= T \mid \neg L \mid L_0 \vee L_1 \mid (t : A)$$

In the logical language L we are able to express complex relationships between properties of different processes. For example

$$(p + q : \langle\alpha\rangle A) \leftrightarrow (p : \langle\alpha\rangle A) \vee (q : \langle\alpha\rangle A),$$

expresses a very simple example of a reduction. It states that the process $p + q$ can do an α and get into a state that satisfies A if and only if p or q can do an α and get into a state that satisfies A. It is a reduction because the formula is valid for all p's and q's, and the validity of $(p + q : \langle\alpha\rangle A)$ is reduced to validity of correctness assertions over the subterms p and q. Although this reduction is almost trivial, in general, it might be quite difficult to get reductions. Consider for example the problem of choosing a B such that

$$(rec\ P.t : \nu X.A) \leftrightarrow (t : B)$$

holds. The aim of this paper is to describe a method for supplying such a B and analogous assertions for all the other operators.

2.2 Semantics

In order to define the semantics we first recall some well-known definitions of transition systems.

Definition 1 A *transition system* T is a triple $(S, L, \rightarrow)$ where S is a set of states, L a set of labels, and $\rightarrow \subseteq S \times L \times S$ a transition relation. The *set of reachable states* R_p from a state $p \in S$ is defined as the least subset of S containing p and closed under $\xrightarrow{L}$, where $\xrightarrow{L} = \bigcup_{l\in L} \xrightarrow{l}$. A *pointed transition system* T is a quadruple $(S, L, \rightarrow, i)$ where $(S, L, \rightarrow)$ is a transition system, $i \in S$ is an initial state, and all states in S must be reachable from i, i.e. S must equal R_i.

Given a pointed transition system $T = (S, L, \rightarrow, i)$ the *rooting of* T is a pointed transition system $\underline{T} = (S \cup \{\underline{i}\}, L, \rightarrow', \underline{i})$ where $\underline{i}$ is a new state assumed not to be in S, and the transition relation $\rightarrow' \subseteq (S \cup \{\underline{i}\}) \times L \times (S \cup \{\underline{i}\})$ is defined by:

$$\rightarrow' \;=\; \rightarrow \cup \{(\underline{i}, \alpha, q) \mid i \xrightarrow{\alpha} q\}.$$

Pictorially the rooting of a pointed transition system is constructed by adjoining a new initial state with the same out-going transitions as the old initial state.

The rooting of a transition system T is just as good as T with respect to satisfaction in our logic. A claim made precise by the rooting lemma below.

The semantics of process terms is given by the transition system $\mathcal{T} = (\mathcal{S}, Act_*, \rightarrow)$, where $\mathcal{S}$ is the set of closed process terms, Act_* the set of composite actions, and $\rightarrow \subseteq \mathcal{S} \times Act_* \times \mathcal{S}$ is the transition relation given as the least relation satisfying the following rules.

$$p \xrightarrow{*} p \qquad a\,p \xrightarrow{a} p \qquad \frac{p \xrightarrow{\alpha} p'}{p + q \xrightarrow{\alpha} p'} \qquad \frac{q \xrightarrow{\alpha} q'}{p + q \xrightarrow{\alpha} q'}$$

$$\frac{p \xrightarrow{\alpha} p' \quad q \xrightarrow{\beta} q'}{p \times q \xrightarrow{\alpha\times\beta} p' \times q'} \qquad \frac{t[rec\ P.t/P] \xrightarrow{\alpha} t'}{rec\ P.t \xrightarrow{\alpha} t'}$$

$$\frac{p \xrightarrow{\alpha} p'}{p\{\Xi\} \xrightarrow{\beta} p'\{\Xi\}} \quad \Xi(\alpha) = \beta \qquad \frac{p \xrightarrow{\alpha} p'}{p \restriction \Lambda \xrightarrow{\alpha} p' \restriction \Lambda} \quad \alpha \in \Lambda$$

Note in particular the rule for product. One of the components in the product may *idle* by means of the idling action $*$ allowing the other component to proceed independently, as in the transition

$$p \xrightarrow{\alpha \times *} p'$$

where the left component of p performs an α-action and the right component idles.

For a transition system $T = (S, L, \rightarrow)$ an assertion A denotes a *property* of T which we take to be a subset of S, hence the set of all properties of T is the powerset $\mathcal{P}(S)$. As assertions may contain free variables we introduce the notion of an environment which describes the interpretation of the variables. An *environment of assertions* for T is a map

$$\phi : Var_A \rightarrow \mathcal{P}(S)$$

which assigns properties to assertion variables. The environment $\phi[U/X]$ is like ϕ except that the variable X is mapped to U.

Formally, relative to the transition system $T = (S, L, \rightarrow)$ the assertion A denotes the property $[\![A]\!]_T \phi$ defined inductively on the structure of A.

$$\begin{array}{lcl}
[\![\neg A]\!]_T \phi & = & S \setminus [\![A]\!]_T \phi \\
[\![A_0 \vee A_1]\!]_T \phi & = & [\![A_0]\!]_T \phi \cup [\![A_1]\!]_T \phi \\
[\![\langle\alpha\rangle A]\!]_T \phi & = & \{s \in S \mid \exists s' \in S.\ s \xrightarrow{\alpha} s' \ \&\ s' \in [\![A]\!]_T \phi\} \\
[\![X]\!]_T \phi & = & \phi(X) \\
[\![\nu X.A]\!]_T \phi & = & \nu U \subseteq S.\ \psi(U) \\
 & & \quad \text{where } \psi : U \mapsto [\![A]\!]_T \phi[U/X] \\
[\![(t : A)]\!]_T \phi & = & \begin{cases} S & \textit{if } t \in [\![A]\!]_T \phi \\ \emptyset & \textit{otherwise} \end{cases}
\end{array}$$

The powerset $\mathcal{P}(S)$ ordered by inclusion is a complete lattice and as we require all variables to appear under an even number of negations the map ψ will always be monotonic, so by Tarski's lemma [Tar55] ψ will have a maximum fixed-point (the *largest postfixed point*) which we denote by $\nu U \subseteq S.\ \psi(U)$.

Define $[\![A]\!]\phi = [\![A]\!]_T \phi$. This gives the standard *global interpretation* of assertions over all states S.

For a transition system $T = (S, L, \rightarrow)$, and a subset Q of S we have the induced transition system

$$T_Q = (Q, L, \rightarrow \cap (Q \times L \times Q)),$$

which is T restricted to the set of states Q. Writing $[\![A]\!]_Q \phi$ for $[\![A]\!]_{T_Q} \phi$ we get a *local interpretation* of A. For particular choices of the subset Q the local and global interpretations coincide, as is captured by the locality lemma below. Let ϕ_Q denote the environment which on the variable X gives $\phi(X) \cap Q$.

Lemma 2 *(Locality lemma.)*
Let $T = (S, L, \rightarrow)$ be a transition system. Given an assertion A, an environment ϕ, and a subset Q of S. Suppose Q satisfies the closedness criterion: Q is closed under $\xrightarrow{K}$, where K is the set of actions appearing inside diamonds in A. Then the following equality holds

$$[\![A]\!]_{T_Q} \phi_Q \ = \ [\![A]\!]_T \phi \cap Q.$$

With the transition system T one particularly interesting choice of Q is the set of reachable states R_p from a state p which by definition satisfies the closedness criterion of the locality lemma. Suppose we wanted to check whether a particular state p belongs to the set of states denoted by an assertion A, then by the locality lemma we obtain:

$$\begin{array}{rl} p \in [\![A]\!]\,\phi & \textit{iff}\ \ p \in [\![A]\!]\,\phi \cap R_p \\ & \textit{iff}\ \ p \in [\![A]\!]_{R_p}\,\phi_{R_p} \end{array}$$

As mentioned previously, the rooting of a transition system T is 'just as good as' T with respect to satisfaction in our logic – which is the intuitive content of the following lemma.

Lemma 3 *(Rooting lemma.)*
Given a pointed transition system, $T = (S_T, L_T, \rightarrow_T, i_T)$, with the rooting $\underline{T}$. Let $r : \mathcal{P}(S_T) \rightarrow \mathcal{P}(S_{\underline{T}})$ be the map on properties that take the initial state of T to the two copies of it in $\underline{T}$ and take all other states to their obvious counterparts. Let $\phi : Var_A \rightarrow \mathcal{P}(S)$ be an environment of assertions. Then

$$r([\![A]\!]_T\,\phi) \;=\; [\![A]\!]_{\underline{T}}\,(r \circ \phi).$$

The connection given by the rooting lemma between pointed transition systems T and their rootings $\underline{T}$ is very useful: The set of states satisfying an assertion will be the same in both interpretations up to application of the map r. In particular the initial state of T will satisfy A if and only if the initial state of $\underline{T}$ satisfies A; an observation central to our development of reductions in section 3.

There is another technical lemma stating a close relationship between syntactic and semantic substitution on assertions which will be used frequently in the proofs.

Lemma 4 *(Substitution lemma.)*
For B a closed assertion, X a variable, A an arbitrary, pure assertion, and ϕ an environment for T, we have

$$[\![A[B/X]]\!]_T\,\phi = [\![A]\!]_T\,\phi\,[[\![B]\!]_T\,\phi/X].$$

For the propositional logic we define the satisfaction predicate $\models_\phi$ relative to an environment ϕ:

$$\begin{array}{ll} \models_\phi T & \textit{always} \\ \models_\phi \neg L & \textit{iff}\ \ \textit{not} \models_\phi L \\ \models_\phi L_0 \vee L_1 & \textit{iff}\ \ \models_\phi L_0\ \textit{or}\ \models_\phi L_1 \\ \models_\phi t : A & \textit{iff}\ \ t \in [\![A]\!]\,\phi \end{array}$$

Furthermore we define the derived predicate $\models$ as:

$$\models L \ \ \textit{iff for all}\ \phi\ \ \models_\phi L.$$

Taking $\bullet$ to be the trivial transition system with one state (denoted $\bullet$) and no transitions, we observe that the set of assertions built from correctness assertions, negations, and conjunctions when interpreted over $\bullet$ is essentially a copy of the logic L, i.e. for such an assertion A we have $[\![A]\!]_\bullet\,\phi = \{\bullet\}$ if and only if $\models_\phi A$ where A is interpreted as a formula in the propositional logic.

3 Reductions

Our method for compositional checking of satisfaction is based on the notion of a *reduction*, which we explain in terms of the prefix operator.

Given a pure and closed assertion A and a prefix at we would like to find a propositional expression B over atoms $(t : B_i)$ such that the following holds:

$$\models (at : A) \leftrightarrow B$$

Having found such a B the validity of $(at : A)$ has been *reduced* to validity of a propositional expression containing only atoms on the subterm t. In order words: B is a *necessary* and *sufficient* condition on the subterm t ensuring that at satisfies A. By the word *reduction* we will henceforth understand *an algorithmic description of how to find B given A and at.*

It is not obvious that such a B exists. Although we can easily express the set of processes that will make the correctness assertion valid as

$$\{t \in \mathcal{S} \mid \models at : A\},$$

it is not necessarily the case that this set can be expressed *within the logic* as an assertion B over atoms $(t : B_i)$ such that

$$\{t \in \mathcal{S} \mid \models B\} \;=\; \{t \in \mathcal{S} \mid \models at : A\}.$$

In general, the ability to do so, will depend on the expressive power of the logic, and the kind of operation for which we are trying to find a reduction. We will show that for our modal logic and all operators of our process algebra, such a B does indeed exist, and furthermore we give for each operator an algorithm that computes one particular choice of B.

In providing this B the most difficult part concerns – not surprisingly – the fixed-points. The single most important property of fixed-points around which all the reductions are centered, is expressed by the reduction lemma. Recall that a map on a complete meet semilattice is ω-anticontinuous if it preserves meets of all decreasing ω-chains.

Lemma 5 *(Reduction lemma.)*
Suppose D and E are powersets over countable sets, and $in : D \to E$ an ω-anticontinuous function with $in(\top_D) = \top_E$. Suppose $\psi : E \to E$ and $\theta : D \to D$ are both monotonic and have the property

$$\psi \circ in = in \circ \theta.$$

We can then conclude that

$$\nu\psi = in(\nu\theta).$$

To understand the role of the reduction lemma, take E to be the lattice of properties of a compound process and D to be a lattice built from properties of immediate subprocesses. The lemma allows us to express a fixed-point property of the original compound process in terms of fixed-points of functions over properties of its immediate subcomponents via the transformation *in*.

For example, the properties of a process at can be identified with certain subsets of the states $S_{\underline{at}}$ in the rooting of the transition system pointed by at, and the properties of t with subsets of the states S_t of the transition system pointed by t. Now we take the transformation to be

$$in : \mathcal{P}(S_t) \times \mathcal{P}(\{\bullet\}) \to \mathcal{P}(S_{\underline{at}})$$

where $in(V_0, V_1) = V_0 \cup \{\underline{at} \mid \bullet \in V_1\}$. The role of the extra product component is to record whether or not the property holds at the initial state $\underline{at}$ of $S_{\underline{at}}$. (The rooting is required to ensure that the initial state at is not confused with later occurrences.)[1]

[1] Because of the isomorphism $\mathcal{P}(A_0) \times \cdots \times \mathcal{P}(A_n) \times \cdots \cong \mathcal{P}(A_0 + \cdots + A_n + \cdots)$ we can still meet the conditions of the reduction lemma when D is a countable product of powersets of countable sets.

An assertion with a free variable occurring positively essentially denotes a monotonic function $\psi : \mathcal{P}(S_{\underline{at}}) \to \mathcal{P}(S_{\underline{at}})$. The definition of the reduction is given by structural induction on assertions ensuring that assertions denoting such functions ψ, and their reductions denoting monotonic functions $\theta : \mathcal{P}(S_t) \times \mathcal{P}(\{\bullet\}) \to \mathcal{P}(S_t) \times \mathcal{P}(\{\bullet\})$ are related by *in* in the manner demanded by the transformation lemma. The lemma then allows the reduction to proceed for fixed-points. As this case of prefixing makes clear reductions of fixed-points can be simultaneous fixed-points. However the use of Bekić's theorem ([Bek84]) replaces the simultaneous fixed-points by fixed-points in the individual components. In the case where these individual components lie in powersets of singletons they end up being replaced by boolean values for closed assertions.

In the course of this definition by structural induction we will be faced with the problem of giving a reduction for assertion variables. One solution to this problem can be found by introducing a syntactic counterpart of *in* called *IN* and define a *change of variables* σ to be a map taking all variables X to $IN(X_0, X_1)$. An application of such a substitution to an assertion A has to satisfy certain technical requirements: It should be *fresh i.e.* for an assertion A when (i) for all variables X at which σ is defined the free variables in $\sigma(X)$ are disjoint from those in A, and (ii) for distinct variables X and X', at which σ is defined, the free variables in $\sigma(X)$ and $\sigma(X')$ are disjoint. It is emphasised that while the syntactic counterparts *IN* of the transformations play the important part in reductions of expressing relationships between variables they do *not* appear in the reductions themselves.

Reductions for all operators can be established along the lines sketched. Each operator involves a judicious choice of *in*, which *IN* is to denote. In the following sections we present this choice and the accompanying reductions.

3.1 Prefix

The reduction for prefix is defined inductively on the structure of assertions and shown in figure 1. Note that $\mathrm{red}^0(at : A; \sigma)$ just renames the variables of A from X to X_0. The transformation *in* was explained in the previous section.[2]

$$
\begin{array}{lcl}
\mathrm{red}^0(at : X; \sigma) & = & X_0 \\
 & & \text{where } \sigma(X) = IN(X_0, X_1) \\
\mathrm{red}^0(at : \nu X.A; \sigma) & = & \nu X_0.\mathrm{red}^0(at : A; \sigma) \\
 & & \text{where } \sigma(X) = IN(X_0, X_1) \\
\mathrm{red}^0(at : \langle\alpha\rangle A; \sigma) & = & \langle\alpha\rangle \mathrm{red}^0(at : A; \sigma) \\
\mathrm{red}^0(at : \neg A; \sigma) & = & \neg \mathrm{red}^0(at : A; \sigma) \\
\mathrm{red}^0(at : A \vee B; \sigma) & = & \mathrm{red}^0(at : A; \sigma) \vee \mathrm{red}^0(at : B; \sigma) \\
 & & \\
\mathrm{red}^1(at : X; \sigma) & = & X_1 \\
 & & \text{where } \sigma(X) = IN(X_0, X_1) \\
\mathrm{red}^1(at : \nu X.A; \sigma) & = & \mathrm{red}^1(at : A; \sigma)[\mathrm{red}^0(at : \nu X.A; \sigma)/X_0][T/X_1] \\
 & & \text{where } \sigma(X) = IN(X_0, X_1) \\
\mathrm{red}^1(at : \langle\alpha\rangle A; \sigma) & = & \begin{cases} t : \mathrm{red}^0(at : A; \sigma) & \text{if } \alpha = a \\ F & \text{if } \alpha \neq a \end{cases} \\
\mathrm{red}^1(at : \neg A; \sigma) & = & \neg \mathrm{red}^1(at : A; \sigma) \\
\mathrm{red}^1(at : A \vee B; \sigma) & = & \mathrm{red}^1(at : A; \sigma) \vee \mathrm{red}^1(at : B; \sigma)
\end{array}
$$

Figure 1: Reduction for prefix defined inductively on the structure of assertions.

[2] For this and the following reductions we have that $\mathrm{red}(at : \langle * \rangle A; \sigma) = red(at : A; \sigma)$ and henceforth we will omit these trivial cases from the presentation.

The reduction is constructed in such a way that the two components are related to A through *in* by

$$[\![A[\sigma]]\!]_{\underline{at}}\,\phi \;=\; in([\![\mathrm{red}^0(at:A;\sigma)]\!]_t\,\phi, [\![\mathrm{red}^1(at:A;\sigma)]\!]_\bullet\,\phi), \tag{1}$$

where σ is a change of variables for A. From the rooting lemma we know that

$$at \in [\![A]\!]_{at}\,\phi \;\; \textit{iff} \;\; \underline{at} \in [\![A]\!]_{\underline{at}}\,\phi$$

and from the definition of *in* and (1) we get

$$at \in [\![A]\!]_{at}\,\phi \;\; \textit{iff} \;\; \bullet \in [\![\mathrm{red}^1(at:A;\sigma)]\!]_\bullet\,\phi.$$

As $\mathrm{red}^1(at:A;\sigma)$ consists of correctness assertions, negations, and conjunctions only, we can consider it to be a formula in our propositional logic, yielding our reduction

$$\models (at:A) \leftrightarrow \mathrm{red}^1(at:A;\sigma).$$

Theorem 6 *(Reduction for prefix.) Given a closed, pure assertion A, a change of variables σ which is fresh for A, and an arbitrary process term t, then* $\models (at:A) \leftrightarrow \mathrm{red}^1(at:A;\sigma)$.

3.2 Nil

The reduction for nil is defined inductively on the structure of assertions and shown in figure 2. The definitions of $\neg$ and $\vee$ are similar to the definitions for prefix and therefore omitted. The transformation $in : \mathcal{P}(\{\bullet\}) \to \mathcal{P}(\{nil\})$ is just the direct image of the obvious isomorphism between $\{\bullet\}$ and $\{nil\}$. Note that the reduction for *nil* is quite trivial and just gives true (T) or false (F).

$$\begin{array}{lcl} \mathrm{red}(nil:X;\sigma) & = & Y \text{ where } \sigma(X) = IN(Y) \\ \mathrm{red}(nil:\nu X.A;\sigma) & = & \mathrm{red}(nil:A;\sigma)[T/Y] \text{ where } \sigma(X) = IN(Y) \\ \mathrm{red}(nil:\langle\alpha\rangle A;\sigma) & = & F \end{array}$$

Figure 2: Reduction for nil.

Theorem 7 *(Reduction for nil.) Given a closed, pure assertion A and a change of variables σ which is fresh for A, then* $\models (nil:A) \leftrightarrow \mathrm{red}(nil:A;\sigma)$.

3.3 Sum

The reduction for sum is presented in figure 3. The definitions for $\neg$ and $\vee$ are omitted as they are similar to the definitions for prefix.

To understand the transformation first note that we have a map $j : S_{t_0} + S_{t_1} \to S_{\underline{t_0+t_1}}$ taking the initial states of t_0 and t_1 to the state $t_0 + t_1$ in $S_{\underline{t_0+t_1}}$ and taking all other states to their obvious counterparts. Let $f : (S_{t_0} + S_{t_1}) + \{\bullet\} \to S_{\underline{t_0+t_1}}$ be the map that takes $\bullet$ to the initial state of $S_{\underline{t_0+t_1}}$ and on $S_{t_0} + S_{t_1}$ behaves like j. We take the transformation to be

$$in : \mathcal{P}(S_{t_0} + S_{t_1}) \times \mathcal{P}(\{\bullet\}) \to \mathcal{P}(S_{\underline{t_0+t_1}})$$

where $in(V_0, V_1) = \{j(s) \mid s \in V_0\} \cup \{\underline{t_0 + t_1} \mid \bullet \in V_1\}$.

Theorem 8 *(Reduction for sum.) Given a closed, pure assertion A, a change of variables σ which is fresh for A, and arbitrary process terms t_0 and t_1, then*

$$\models (t_0 + t_1 : A) \leftrightarrow \mathrm{red}^1(t_0 + t_1 : A;\sigma).$$

$$
\begin{array}{lcl}
\mathrm{red}^0(t_0 + t_1 : X; \sigma) & = & X_0 \\
 & & \text{where } \sigma(X) = IN(X_0, X_1) \\
\mathrm{red}^0(t_0 + t_1 : \nu X.A; \sigma) & = & \nu X_0.\mathrm{red}^0(t_0 + t_1 : A; \sigma) \\
 & & \text{where } \sigma(X) = IN(X_0, X_1) \\
\mathrm{red}^0(t_0 + t_1 : \langle\alpha\rangle A; \sigma) & = & \langle\alpha\rangle \mathrm{red}^0(t_0 + t_1 : A; \sigma) \\
 & & \\
\mathrm{red}^1(t_0 + t_1 : X; \sigma) & = & X_1 \\
 & & \text{where } \sigma(X) = IN(X_0, X_1) \\
\mathrm{red}^1(t_0 + t_1 : \nu X.A; \sigma) & = & \mathrm{red}^1(t_0 + t_1 : A; \sigma)[\mathrm{red}^0(t_0 + t_1 : \nu X.A; \sigma)/X_0][T/X_1] \\
 & & \text{where } \sigma(X) = IN(X_0, X_1) \\
\mathrm{red}^1(t_0 + t_1 : \langle\alpha\rangle A; \sigma) & = & (t_0 : \langle\alpha\rangle A^0) \vee (t_1 : \langle\alpha\rangle A^0) \\
 & & \text{where } A^0 = \mathrm{red}^0(t_0 + t_1 : A; \sigma)
\end{array}
$$

Figure 3: Reduction for sum.

3.4 Relabelling

For relabelling we take the transformation to be $in : \mathcal{P}(R_t) \to \mathcal{P}(R_{t\{\Xi\}})$ where $in(V) = \{p\{\Xi\} \mid p \in V\}$.

$$
\begin{array}{lcl}
\mathrm{red}(t\{\Xi\} : X; \sigma) & = & Y \text{ where } \sigma(X) = IN(Y) \\
\mathrm{red}(t\{\Xi\} : \nu X.A; \sigma) & = & \nu Y.\mathrm{red}(t\{\Xi\} : A; \sigma) \text{ where } \sigma(X) = IN(Y) \\
\mathrm{red}(t\{\Xi\} : \langle\alpha\rangle A; \sigma) & = & \langle\Xi^{-1}(\alpha)\rangle \mathrm{red}(t\{\Xi\} : A; \sigma)
\end{array}
$$

Figure 4: Reduction for relabelling.

Theorem 9 *(Reduction for relabelling.) Assume A closed and pure, a change of variables σ which is fresh for A, and an arbitrary process term t, then $\models (t\{\Xi\} : A) \leftrightarrow (t : \mathrm{red}(t\{\Xi\} : A; \sigma))$.*

3.5 Restriction

For restriction we take the transformation to be $in : \mathcal{P}(R_t) \to \mathcal{P}(R_{t\restriction\Lambda})$ where $in(V) = \{p \restriction \Lambda \mid p \in V\} \cap R_{t\restriction\Lambda}$.

$$
\begin{array}{lcl}
\mathrm{red}(t \restriction \Lambda : X; \sigma) & = & Y \text{ where } \sigma(X) = IN(Y) \\
\mathrm{red}(t \restriction \Lambda : \nu X.A; \sigma) & = & \nu Y.\mathrm{red}(t \restriction \Lambda : A; \sigma) \text{ where } \sigma(X) = IN(Y) \\
\mathrm{red}(t \restriction \Lambda : \langle\alpha\rangle A; \sigma) & = & \begin{cases} \langle\alpha\rangle \mathrm{red}(t \restriction \Lambda : A; \sigma) & \text{if } \alpha \in \Lambda \\ F & \text{if } \alpha \notin \Lambda \end{cases}
\end{array}
$$

Figure 5: Reduction for restriction.

Theorem 10 *(Reduction for restriction.) Assume A closed and pure, a change of variables σ which is fresh for A, and an arbitrary process term t, then $\models (t \restriction \Lambda : A) \leftrightarrow (t : \mathrm{red}(t \restriction \Lambda : A; \sigma))$.*

3.6 Recursion

In order to define the reduction for recursion, we will need to extend our assertion language with an assertion $\widehat{P}$ to identify recursion points. The semantics of $\widehat{P}$ is simply:[3]

[3]The general semantics should be $[\![\widehat{P}]\!]_T \phi = \{P, \underline{P}\} \cap S_T$, but due to our requirement of guardedness, we will never be involved with rooting a state identifier, so the stated semantics is sufficient.

$$[\![\widehat{P}]\!]_T \phi \;=\; \{P\} \cap S_T.$$

It can be verified that the locality and the rooting lemma still hold. All the reductions mentioned in the previous sections should be extended to take care of the assertions $\widehat{P}$ and this is easily done – they should all give F. Furthermore, we add a reduction for P, which is like the one for *nil*, except that it gives T on $\widehat{P}$.

For the first time we will need to put in extra correctness assertions in our reductions, which furthermore might contain free assertion variables. These correctness assertions can however be closed by a *closure lemma* and then 'pulled out' by a *purifying lemma* yielding an expression which belongs to the propositional language without any correctness assertions appearing inside other assertions, hence being applicable for further reductions.

Theorem 11 *(The purifying lemma.)*
Let A be an assertion with all correctness assertions closed and let t be a process term. Then there exists an expression B over unnested correctness assertions such that, $\models (t : A) \leftrightarrow B$.

Moreover, the proof of the lemma gives an algorithm for computing such a B. The closure lemma can be found in [Win90].

$$\begin{array}{lcl}
\mathrm{red}(\mathit{rec}\ P.t : X; \sigma) & = & Y \ \text{ where } \sigma(X) = \mathit{IN}(Y) \\
\mathrm{red}(\mathit{rec}\ P.t : \nu X.A; \sigma) & = & \nu Y.\mathrm{red}(\mathit{rec}\ P.t : A; \sigma) \ \text{ where } \sigma(X) = \mathit{IN}(Y) \\
\mathrm{red}(\mathit{rec}\ P.t : \langle\alpha\rangle A; \sigma) & = & \langle\alpha\rangle A' \vee (\widehat{P} \wedge (t : \langle\alpha\rangle A')) \\
 & & \text{where } A' = \mathrm{red}(\mathit{rec}\ P.t : A; \sigma)
\end{array}$$

Figure 6: Reduction for recursion. The definitions for $\neg$ and $\vee$ are omitted as they again are similar to the definitions for prefix.

Take $f : S_{\underline{t}} \to S_{\mathit{rec}\ P.t}$ to be the map that takes $\underline{t}$ to $\mathit{rec}\ P.t$ and all other states s to $s[\mathit{rec}\ P.t/P]$. The transformation for recursion $in : \mathcal{P}(S_{\underline{t}}) \to \mathcal{P}(S_{\mathit{rec}\ P.t})$ is defined to be the direct image of f.

Theorem 12 *(Reduction for recursion.) Given a closed, pure assertion A, a change of variables σ which is fresh for A, and an arbitrary process term t then*

$$\models (\mathit{rec}\ P.t : A) \leftrightarrow (t : \mathrm{red}(\mathit{rec}\ P.t : A; \sigma)).$$

3.7 Product

A reduction for a product $q \times p$ should be an assertion B over atoms $(q : B_i)$ and $(p : C_j)$ such that

$$\models q \times p : A \ \ \mathit{iff} \ \ \models B.$$

Unfortunately, if we insist on finding such a B without inspecting either p or q, we can get a very complex expression which, in the case of fixed-points will even become infinite unless assumptions on the possible sizes of p and q are made (cf. the remarks at the end of [Win90]). In [Win90] it is shown how a very reasonable sized B can be found, when the assertion language is restricted rather severely, excluding disjunctions, negations, minimal fixed-points, and general box formulas, but still having maximal fixed-points, diamond formulas, a strong version of box formulas, and conjunctions.

Here we present another approach. We give a reduction when p is a process term without restrictions and relabellings, i.e. we find a B (depending on p) s.t.

$$\models q \times p : A \quad \textit{iff} \quad \models q : B.$$

Let $R_p = \{p_1, \ldots, p_n\}$ be the finite set of reachable states of p in some fixed enumeration. We define the map $in : \underbrace{\mathcal{P}(R_q) \times \ldots \times \mathcal{P}(R_q)}_{n} \to \mathcal{P}(R_{q \times p})$ as

$$in(U_{p_1}, \ldots, U_{p_n}) = (U_{p_1} \times p_1) \cup \ldots \cup (U_{p_n} \times p_n).$$

As usual we have a change of variables σ with $\sigma(X) = IN(X_{p_1}, \ldots, X_{p_n})$. As a notational convenience we write A/p for $\mathrm{red}(q \times p : A; \sigma)$ omitting the σ which is always assumed to map an X into $X_{p_1}, \ldots, X_{p_n}$. The reduction is shown in figure 7.

$$
\begin{array}{lcl}
\neg A/p & = & \neg(A/p) \\
A_0 \vee A_1/p & = & (A_0/p) \vee (A_1/p) \\
X/p & = & X_p \\
\nu X.A/p & = & C_k(\nu(X_{p_1}, \ldots, X_{p_n}).(A/p_1, \ldots, A/p_n)) \\
 & & \text{where } \{p_i\}_i \text{ denotes the set of reachable} \\
 & & \text{states from } p \text{ with } p = p_k. \\
A/q \times r & = & (A/r)/q \\
 & & \text{with the actions in the modalities of } A \text{ reassociated} \\
\langle \alpha \times \beta \rangle A/nil & = & \begin{cases} \langle \alpha \rangle (A/nil) & \textit{if } \beta = * \\ F & \textit{if } \beta \neq * \end{cases} \\
\langle \alpha \times \beta \rangle A/\gamma q & = & \begin{cases} \langle \alpha \rangle (A/\gamma q) & \textit{if } \beta = * \\ \langle \alpha \rangle (A/q) & \textit{if } \beta = \gamma \\ F & \textit{otherwise} \end{cases} \\
\langle \alpha \times \beta \rangle A/q + r & = & (\langle \alpha \times \beta \rangle A/q) \vee (\langle \alpha \times \beta \rangle A/r) \\
\langle \alpha \times \beta \rangle A/rec\ P.t & = & \langle \alpha \times \beta \rangle A/t[rec\ P.t/P]
\end{array}
$$

Figure 7: Reduction for product. $C_k(\nu \underline{X}.\underline{A})$ denotes the k'th component of the n-ary fixed-point $\nu \underline{X}.\underline{A}$, closed by repeated application of Bekić's theorem[4].

Theorem 13 *(Reduction for product.)*
Assume given a pure and closed assertion A, a change of variables σ, and a term p with no restrictions and relabellings. We then have for an arbitrary term q:

$$\models (q \times p : A) \leftrightarrow (q : red(q \times p : A; \sigma)).$$

The case of the maximal fixed-point is established by repeated application of Bekić's theorem, and the resulting assertion might become rather complex, as in the worst case a fixed-point will appear for each reachable state of p, and on top of this, Bekić's theorem might increase the size of the assertion considerably. We are currently investigating methods to control the potential blow-up in general. We present in the next section an example, that indicates that in practice this need not be the case.

[4]Termination is ensured by the well-founded order consisting of the number of products in the process term combined lexicographically with the structure of assertions again combined lexicographically with the maximal depth to a prefix in the process term

4 Examples

It is an important property of all our reductions (except product) that they only dependent on the top-most operator of the process term, hence we can leave part of a process unspecified and still apply the reductions. Technically this can be done by adding *process variables* to our language of processes. Given an assertion and a process with variables, we can then compute a propositional expression with correctness assertions over the variables, expressing what relationship there should be between them in order to make the process satisfy the assertion. In this way the reductions compute what corresponds to weakest preconditions in Hoare logic.

As pointed out in the previous section, the reductions for product has the potential of becoming rather complex. In this section we show by a small example, that in practice, the reductions need not turn out to be too complex.

First we define a binary parallel operator $\|_{K,L}$ which allows its left and right components to independently perform the actions indicated by the sets K and L, except that they are required to synchronise on common actions of K and L. The precise definition is

$$p \|_{K,L} q \stackrel{\text{def}}{=} (p \times q) \upharpoonright \Lambda\{\Xi\}$$

where $\Lambda = \{a{\times}a \mid a \in K \cap L\} \cup \{a{\times}* \mid a \in K \setminus L\} \cup \{*{\times}a \mid a \in L \setminus K\}$ and

$$\begin{array}{lcl} \Xi(a{\times}a) & = & a, \text{ for all } a \in K \cap L \\ \Xi(a{\times}*) & = & a, \text{ for all } a \in K \setminus L \\ \Xi(*{\times}a) & = & a, \text{ for all } a \in L \setminus K \\ \Xi(\alpha) & & \text{undefined otherwise.} \end{array}$$

Now assume that we want to construct a small system consisting of a coffee vending machine and a researcher. The coffee machine should be able to accept money and then supply a cup of coffee. The researcher should be able to pay out money, drink coffee, and publish papers. Suppose we know how the researcher behaves, specified by a process term r, but would like to find out what kind of coffee machine x to put into the system, such that eventually the researcher has no other choice than to publish a paper.

In general a property of the form 'eventually only the action α can happen' can be expressed by the assertion

$$\mu X.\langle - \rangle T \wedge [-\alpha]X$$

where

$$\langle - \rangle A \;=\; \langle Act \rangle A \qquad [-K]A \;=\; [Act \setminus K]A.$$

Our problem can now be restated.

> Assume the actions to be p for publish, c for taking/giving coffee, m for taking/giving money, and define $K = \{m, c\}, L = \{m, c, p\}$. Which values of x make the following correctness assertion valid
>
> $$x \|_{K,L} r : \mu X.\langle - \rangle T \wedge [-p]X? \qquad (2)$$

Suppose the researcher r behaves as $rec\ P.m.c.(m.c.P + p.P)$. Then expanding the definition of $\|_{K,L}$ and applying the reduction for restriction and relabelling, we get the equivalent correctness assertion

$$x \times r : \mu X.\langle m{\times}m, c{\times}c, *{\times}p \rangle T \wedge [m{\times}m, c{\times}c]X$$

and then, by applying the reduction for product, the equivalent

$$x : \mu X.\langle m \rangle T \wedge [m](\langle c \rangle T \wedge [c][m](\langle c \rangle T \wedge [c]X)). \qquad (3)$$

One can now use (3) to verify different proposals for coffee machines, without redoing the first two steps. This might be done by our method, or for closed terms by other model checking algorithms.

An interesting point to note about the assertion in (3) is that, although the researcher r had *four* reachable states, and then potentially four fixed-points could appear, only *one* fixed-point appears in the resulting assertion.

Returning to the example, we can verify that a successful choice of x is $m.c.nil$, i.e. a coffee machine that accepts money and give coffee once, and then breaks down, whereas $rec\ P.m.c.P$ is an unsuccessful choice. Reading the assertion in (3) carefully, we can express the requirement to the machine as 'after having offered a finite and odd number of m's followed by c's, no m should be offered.'

Changing the behaviour of the researcher slightly and taking $r = rec\ P.m.c.P + m.c.p.P$ and performing the reductions for restriction, relabelling, and product, we arrive at the correctness assertion $x : F$, i.e. there are no coffee machines that will make the system fulfill the requirement.

References

[AC88] André Arnold and Paul Crubille. A linear algorithm to solve fixed-point equations on transitions systems. *Information Processing Letters*, 29:57–66, 1988.

[Bek84] H. Bekić. Definable operations in general algebras, and the theory of automata and flow charts. *Lecture Notes in Computer Science*, 177, 1984.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[Cle90] Rance Cleaveland. Tableau-based model checking in the propositional mu-calculus. *Acta Informatica*, 27:725–747, 1990.

[CLM89] E.M. Clarke, D.E. Long, and K.L. McMillan. Compositional model checking. In *Procedings of 4th Annual Symposium on Logic in Computer Science*. IEEE, 1989.

[Dam90] Mads Dam. Translating CTL* into the modal μ-calculus. Technical Report ECS-LFCS-90-123, Laboratory for Foundations of Computer Science, Uni. of Edinburgh, November 1990.

[EL86] E. Allen Emerson and Chin-Luang Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Symposium on Logic in Computer Science, Proceedings*, pages 267–278. IEEE, 1986.

[Lar88] Kim G. Larsen. Proof systems for Hennessy-Milner logic with recursion. In *Proceedings of CAAP*, 1988.

[LX90] Kim G. Larsen and Liu Xinxin. Compositionality through an operational semantics of contexts. In M.S. Paterson, editor, *Proceedings of ICALP*, volume 443 of *LNCS*, 1990.

[Sti91] Colin Stirling. Modal and Temporal Logics. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*. Oxford University Press, 1991.

[SW89] Colin Stirling and David Walker. Local model checking in the modal mu-calculus. In *Proceedings of TAPSOFT*, 1989.

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5, 1955.

[Win89] Glynn Winskel. A note on model checking the modal ν-calculus. In *Proceedings of ICALP*, volume 372 of *LNCS*, 1989.

[Win90] Glynn Winskel. On the compositional checking of validity. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings of CONCUR '90*, volume 458 of *LNCS*, 1990.