Testing-Based Abstractions for Value-Passing Systems*

Rance Cleaveland** and James Riely***

[1] Dept. of Computer Science, N.C. State University, Raleigh, NC 27695-8206, USA
[2] Dept. of Computer Science, University of N.C., Chapel Hill, NC 27599-3175, USA
email: rance@csc.ncsu.edu, riely@cs.unc.edu

Abstract. This paper presents a framework for the abstract interpretation of processes that pass values. We define a process description language that is parameterized with respect to the set of values that processes may exchange and show that an abstraction over values induces an abstract semantics for processes. Our main results state that if the abstract value interpretation safely/optimally approximates the ground interpretation, then the resulting abstracted processes safely/optimally approximate those derived from the ground semantics (in a precisely defined sense). As the processes derived from an abstract semantics in general have far fewer states than those derived from a concrete semantics, our technique enables the automatic analysis of systems that lie beyond the scope of existing techniques.

1 Introduction

Research over the past decade points to the practical viability of automatically verifying concurrent finite-state systems. Algorithms have been proposed for determining whether such systems enjoy properties specified by formulas in various temporal logics [4, 5, 8, 25, 27, 28] and for computing whether or not two systems exhibit the same (or related) observable behavior [2, 6, 18, 21]. Tools built around implementations of these algorithms have been applied to the analysis of a variety of different kinds of systems [7, 11, 12, 22, 23, 24]. When communicating processes are capable of exchanging values taken from an infinite set, however, the resulting system is usually not finite-state, and the automatic analysis routines mentioned above, which rely to some extent on an enumeration of system states, are not directly applicable. Even when the set of values is finite (as is the case, for example, in communication protocols, where packets typically have a fixed width) automatic analysis rapidly becomes impractical as the size of the value set increases. On the other hand, many system properties are largely insensitive to the specific values that systems pass. Some, such as deadlock-freedom and fairness constraints, do not refer to specific values at all, while others are only sensitive to certain aspects of data. These observations suggest that it may be possible to reduce the analysis of value-passing systems to simpler systems that exchange more abstract values.

In this paper, we present a framework for generating *abstractions* of communicating systems based on abstractions of the values exchanged by processes. Such abstracted systems in general have many fewer states than the systems from which they are constructed while retaining some measure of information about the behavior of the original system.

* This paper appeared in B. Jonsson and J. Parrow, editors, *CONCUR '94*, volume 836 of *Lecture Notes in Compter Science*, pages 417–432, Uppsala, Sweden, August 1994.

** Research supported by NSF/DARPA Grant CCR-9014775, NSF Grant CCR-9120995, ONR Young Investigator Award N00014-92-J-1582, and NSF Young Investigator Award CCR-9257963.

*** Research supported by ONR Contract N00014-92-C-0182.

This last fact permits the analysis of a smaller, abstracted system in lieu of the original one, with results obtained from the former also guaranteed to be applicable to the latter. Our work is inspired by that done in the *abstract interpretation* of sequential programming languages [9], which has led to the development of techniques that permit certain properties of sequential programs to be deduced automatically at compile-time. In particular, our approach is similar to work on *factored semantics* described in [17], although our setting is operational rather than denotational. More specifically, we give a semantics for a language similar to CSP [16] in which the core semantics is explicitly parameterized with respect to a *value interpretation* that defines the meaning of data. We also define the conditions under which an abstract value interpretation is *safe* relative to the original interpretation and under which a safe abstraction is said to be *optimal*. We then prove the main results of this paper: safe (optimal) abstract value interpretations yield safe (optimal) abstract process interpretations using our parameterized semantics.

The remainder of the paper is organized as follows. The next section formally introduces value interpretations. Section 3 then gives the syntax and semantics of processes and defines a preorder on processes, indicating when one process approximates another. The semantic relation is a variant of the testing/failures preorders [13], which we also argue preserves both liveness and safety properties of processes. In Section 4 we extend the abstraction functions defined in Section 2 to process terms (syntax) and labeled transition systems (semantics) so that the main results may be formally stated. The section following presents these results, while Section 6 contains a simple example illustrating how they may be applied to the analysis of concurrent systems. Section 7 contains our conclusions and directions for future work.

2 Values and value abstractions

In the next section we give the syntax and semantics of VPL_I—Value-Passing Language with value interpretation I. VPL_I is a simple variant of the language defined by Hennessy and Ingólfsdóttir in [14]; the difference lies in the fact that VPL_I is explicitly parameterized with respect to an interpretation I of values. In this section we introduce value interpretations and show how traditional notions from *abstract interpretation* may be adapted to our setting.

2.1 Value interpretations

In order to define the syntax of VPL_I, we first fix a syntax for constructing boolean and value expressions that is parameterized with respect to the values that may appear in these expressions. That is, the syntax of the expression language contains "holes" to be filled by members of the value set; changing the value set then simply changes the choices for filling the holes. In this setting, a *value interpretation* for a set of values should allow us to "evaluate" expressions.

To formalize these notions, let $(x, y \in)$ Var be a countable set of variable symbols (we write "$(x, y \in)$ Var" to indicate that the symbols x and y will range over the set Var), and let let Σ_{Expr} and Σ_{BExpr} be fixed signatures containing no 0-ary function symbols (constants). The idea is that Σ_{Expr} and Σ_{BExpr} contain constructors for building expressions and boolean expressions, respectively. Also let Val be a nonempty set of values that is disjoint from Var. Then $(e \in) Expr_{Val}$ contains the elements of the term algebra built from $\Sigma_{Expr} \cup Val \cup Var$, with the arity of elements in Val and Var taken to be 0, and $(be \in)$ $BExpr_{Val}$ represents the set of elements of the term algebra $\Sigma_{BExpr} \cup Val \cup Var$. $Expr_{Val}$ and $BExpr_{Val}$ comprise the set of value and boolean expressions, respectively, that may be built from variables and elements of Val using the constructors in Σ_{Expr} and Σ_{BExpr}.

We use $e_1[e_2/x]$ to represent the expression obtained by simultaneously replacing all (free) occurrences of x in e_1 by e_2 in the usual sense, and $CExpr_{Val}$ (resp. $CBExpr_{Val}$) to denote the subset of $Expr_{Val}$ ($BExpr_{Val}$) closed with respect to Var. Also let $\wp(S)$ denote the power set of set S. We may now define value interpretations as follows.

Definition 1. *A* value interpretation *is a triple* $I = \langle Val_I, \mathbb{B}_I[\![\cdot]\!], \mathbb{E}_I[\![\cdot]\!]\rangle$ *where* $(v \in)\, Val_I$ *is a countable set of values,* $\mathbb{B}_I[\![\cdot]\!] : CBExpr_{Val_I} \mapsto (\wp(Bool) - \{\emptyset\})$, *and*

$$\mathbb{E}_I[\![\cdot]\!] : CExpr_{Val_I} \mapsto (\wp(Val_I) - \{\emptyset\}).$$

We usually write $Expr_I$, $CExpr_I$, $BExpr_I$ and $CBExpr_I$ in lieu of $Expr_{Val_I}$, et cetera; we also do not distinguish between values and their syntactic representation.

One noteworthy aspect of this definition is that the "valuation" functions $\mathbb{E}_I[\![\cdot]\!]$ and $\mathbb{B}_i[\![\cdot]\!]$ may be nondeterministic; that is, $\mathbb{E}_i[\![e]\!]$ may return a set of possible values. The utility of this will be made apparent in Section 2.3.

The remainder of this section is devoted to a discussion of *abstractions* over value interpretations. In particular, we show how traditional concepts from abstract interpretation —abstraction, concretization, safety and optimality— may be adapted to our setting. We consider each of these in turn.

2.2 Abstraction and concretization

Abstract interpretation may be seen as the generalized theory of static analysis of programs. The motivation for its study arises from practical considerations in the design of compilers. One would like compilers to generate code that is as efficient and free of runtime errors as possible; however, many of the analyses required are undecidable in general. Abstract interpretation provides a basis for analyzing abstracted versions of programs in order to provide partial information about their behavior.

The formal foundations of abstract interpretation lie in the definition of *abstraction* and *concretization* functions between "concrete" and "abstract" domains. Intuitively, the concrete domain contains the meanings of "real" programs, while the abstract domain includes the meanings of programs obtained by abstracting away some of the detail from concrete programs. Formally, let $\langle C, \sqsubseteq_C\rangle$ and $\langle A, \sqsubseteq_A\rangle$ be preorders.[3]. C may be thought of as representing the possible concrete meanings a program might have, with $c_1 \sqsubseteq_C c_2$ holding if c_1 contains "less information" than c_2, while A represents the corresponding set of abstract meanings. Then an abstract interpretation of C in A may be given as a pair of functions, α and γ, that constitute a *Galois insertion.*

Definition 2. *Given preorders* $\langle C, \sqsubseteq_C\rangle$ *and* $\langle A, \sqsubseteq_A\rangle$, *we say that* α *and* γ *form a Galois insertion, written* $\langle C, \sqsubseteq_C\rangle \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} \langle A, \sqsubseteq_A\rangle$, *when:*

- $\alpha : C \mapsto A$ *and* $\gamma : C \mapsto A$ *are monotonic,*
- $\forall c \in C:\ \gamma \circ \alpha(c) \sqsubseteq_C c$, *and*
- $\forall a \in A:\ \alpha \circ \gamma(a) =_A a$, *where* $a =_A a'$ *iff* $a \sqsubseteq_A a'$ & $a' \sqsubseteq_A a$.

Function α is usually called the *abstraction function*, while γ is called the *concretization function.* The Galois insertion requirements may be seen as an assertion that α and γ are compatible. In particular, the second condition indicates that α does not "add information" to its argument, while the third indicates that γ "preserves the information" of its argument. It should be noted that our use of the symbol $\sqsubseteq$ is contrary to tradition in abstract interpretation; thus, the second condition would usually be written as $\gamma \circ \alpha(c) \sqsupseteq_C c$.

[3] Traditionally, these are taken to be lattices, but we require this slightly weaker formulation in Section 4

Our notation is chosen to be consistent with traditional information-theoretic orderings on processes.

In the setting of value interpretations there is no *a priori* notion of ordering on values; however, we can define still abstraction functions as follows. Given two sets of values, Val_C (for *concrete*) and Val_A (for *abstract*), we call a total surjection $\alpha\colon Val_C \mapsto Val_A$ an *abstraction* function. As an example, let Val_C be the set of natural numbers and Val_A be the set $\{\mathsf{neg}, 0, \mathsf{pos}\}$. Then we may define the usual abstraction $\alpha\colon Val_C \mapsto Val_A$ by taking $\alpha(c) = \mathsf{neg}$ if $c < 0$, $\alpha(0) = 0$, and $\alpha(c) = \mathsf{pos}$ if $c > 0$.

An abstraction function naturally generalizes to expressions: in $\alpha(e)$ each occurrence of a value c in e is replaced by $\alpha(c)$. When no confusion can arise, we abuse notation slightly by using α to refer any such "lifted" instance of α.

Definition 3. *The following functions are defined by induction on the structure of their domains:* $\alpha : Expr_C \mapsto Expr_A$, $\alpha : CExpr_C \mapsto CExpr_A$, $\alpha : BExpr_C \mapsto BExpr_A$, *and* $\alpha\colon CBExpr_C \mapsto CBExpr_A$.

We write $\alpha : C \mapsto A$ refer to the entire family of functions induced by $\alpha : Val_C \mapsto Val_A$ between value interpretations C and A.

In order to apply traditional abstract interpretation techniques to value interpretations, we also need a concretization function for each abstraction α. Values are unordered, however, and the inverse of α, a natural choice for γ, is not in general a function. The powerset of values, on the other hand, does have a natural information-theoretic ordering. Suppose $V_C \subseteq Val_C$; then V_C represents the potential results of evaluating an expression in the concrete interpretation C, and likewise for Val_A. The smaller the set, the more the information available about the actual value returned: indeed, usually we expect that $\mathbb{E}_C[\![\cdot]\!]$ is deterministic and total and thus maps each expression to a singleton set. So we take as our preorder $\langle \wp(C) - \{\emptyset\}, \supseteq\rangle$, where $V_C \supseteq V'_C$ means that V_C contains less information than V'_C as it contains more elements. We may now define abstraction and concretization functions on these domains as follows.

Definition 4. *For* $\alpha : Val_C \mapsto Val_A$, $V_C \subseteq Val_C$ *and* $V_A \subseteq Val_A$, *define the lifted abstraction and concretization functions as:* $\alpha(V_C) = \{a \mid \exists c \in V_C : \ \alpha(c) = a\}$, *and* $\gamma(V_A) = \{c \mid \exists a \in V_A\colon \alpha(c) = a\}$.

These functions turn out to form a Galois insertion.

Lemma 5.
$$\langle \wp(Val_C), \supseteq\rangle \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} \langle \wp(Val_A), \supseteq\rangle$$

In a similar way we can define lifted abstractions and concretizations on all of the syntactic categories of VPL_I. In each case, the result is a Galois insertion on the preorder over sets induced by the superset relation. For example, we have that:

$$\langle \wp(CExpr_C), \supseteq\rangle \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} \langle \wp(CExpr_A), \supseteq\rangle$$

2.3 Safety

In traditional abstract interpretation, after giving a Galois insertion $\langle S_C, \sqsubseteq_C\rangle \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} \langle S_A, \sqsubseteq_A\rangle$ one then gives abstract versions $f_A : S_A \mapsto S_A$ for each operation $f_C : S_C \mapsto S_C$ used in defining the semantics of a language. Of course, one would wish that an abstraction f_A of f_C be compatible with f_C, in some sense. This notion is made precise by defining f_A to be a *safe* approximation of f_C if for all $c \in S_C$, $(f_A \circ \alpha)(c) \sqsubseteq (\alpha \circ f_C)(c)$. Intuitively, f_A is safe if it can never "add information" to the results produced by f_C; that is, $\alpha \circ f_C$ produces the most precise abstract information for any value $c \in S_C$.

In our setting, there are no specific operators with respect to which safety can be defined, since the exact syntax of expressions is not specified. Instead, our definition of safety uses the evaluation functions of the interpretation.

Definition 6. *A value abstraction* $\alpha : C \mapsto A$ *is* safe *iff for all* $e \in CExpr_C$ *and* $be \in CBExpr_C$*,*

$$\mathbb{B}_A[\![\alpha(be)]\!] \supseteq \mathbb{B}_C[\![be]\!] \quad \textit{and} \quad \mathbb{E}_A[\![\alpha(e)]\!] \supseteq \alpha(\mathbb{E}_C[\![e]\!]).$$

In other words, one interpretation is *safe* relative to another if for all terms, the former yields no more precise a result than the former. Returning to our example, let $e = (1 + (-2))$. The most precise abstract information about e would be $\alpha(\mathbb{E}_C[\![e]\!]) = \{\mathsf{neg}\}$, whereas $\mathbb{E}_A[\![\alpha(e)]\!] = \{\mathsf{neg}, 0, \mathsf{pos}\}$. In this case, the abstract semantics can yield no more precise an answer; if it did, it would get at least some answer "wrong", since $\alpha(1 + (-2)) = \alpha(2 + (-1)) = \alpha(1 + (-1)) = \mathsf{pos} + \mathsf{neg}$.

2.4 Optimality

While safety is a necessary condition for an abstraction of an operator to be useful, it is not sufficient. For example, one can give an abstract operator that is trivially safe: just map each abstract value to a least value in the abstract domain. While safe, this operator does not convey useful information about the concrete operation it is supposed to approximate. In our previous example, the semantic function that maps all expressions to $\{\mathsf{neg}, 0, \mathsf{pos}\}$ would be an example of such a trivial, yet safe, semantics.

At the opposite extreme from the trivial semantics is the optimal (or *induced*) semantics. In the traditional setting, f_A is said to be *optimal* for f_C if f_A is the most precise safe approximation of f_C. Formally [17], we may say that f_A is optimal if $f_A(a) =_A \alpha \circ f_C \circ \gamma(a)$, where $=_A$ is the equivalence induced by $\sqsubseteq_A$.

In order to formalize this notion in our setting, we first must extend our semantic functions to operate over *sets* of terms.

Definition 7. *For* $BS \subseteq CBExpr_I$ *and* $ES \subseteq CExpr_I$ *define:*

$$\mathbb{B}_I[\![BS]\!] = \{v \mid \exists be \in BS\colon\ v \in \mathbb{B}_I[\![be]\!]\}, \quad \textit{and}$$
$$\mathbb{E}_I[\![ES]\!] = \{v \mid \exists e \in ES\colon\ v \in \mathbb{E}_I[\![e]\!]\}.$$

Given $\alpha\colon C \mapsto A$*, we say that* A *is* optimal *for* C *iff for all* $e \in CExpr_C$ *and* $be \in CBExpr_C$*:*

$$\mathbb{B}_A[\![be]\!] = \alpha(\mathbb{B}_C[\![\gamma(be)]\!]) \quad \textit{and} \quad \mathbb{E}_A[\![e]\!] = \alpha(\mathbb{E}_C[\![\gamma(e)]\!]).$$

2.5 Preview

This section has introduced value interpretations and described the conditions under which an abstract value interpretation is considered safe and optimal. In the next section we present a process description language defined parametrically with respect to value interpretations. The semantics of the language is given as a mapping from process terms to labeled transition systems, and a preorder is defined on these semantic objects. In Section 4 we construct a Galois insertion between concrete and abstract labeled transition systems. Finally, in Section 5, we extend the definitions of safety and optimality to the process description language and prove the main results: if an abstract value interpretation is safe (optimal), then the process interpretation constructed using the abstract semantics will be safe (optimal).

3 Processes

This section introduces the syntax and semantics of VPL_I and defines a semantic preorder relating processes given in the language.

3.1 Syntax

In addition to the sets of value- and boolean-expression constructors mentioned in the previous section, the definition of process terms is parameterized with respect to countable sets $(P, Q \in)$ PN of process names and $(c \in)$ $Chan$ of channel names. We use L to range over finite subsets of $Chan$. Given a value interpretation I, we define the set of possible *communications* as $(a, b \in) Comm_I = \{c?v, c!v \mid c \in Comm \;\&\; v \in Val_I\}$. The set of actions, $(\lambda \in) Act_I = Comm_I \cup \{\tau\}$, includes also the hidden action τ. Intuitively, $c?v$ represents the act of receiving value v on channel c, while $c!v$ corresponds to the output of v on c. The action τ represents an internal computation step. Finally, if $(f \in) Chan \mapsto Chan$ then $\hat{f} \in Act_I \mapsto Act_I$ is defined by $\hat{f}(\tau) = \tau$, $\hat{f}(c!v) = f(c)!v$, and $\hat{f}(c?v) = f(c)?v$. That is, $\hat{f}$ relabels the channel components of actions.

The syntax of $(t \in) VPL_I$ may now be given by the following grammar:

$$\begin{array}{rl} t ::= & \mathsf{nil} \;\Big|\; c?x.t \;\Big|\; c!e.t \;\Big|\; be \rhd t_1 \diamond t_2 \;\Big|\; t_1 [\!] t_2 \;\Big|\; t_1 \oplus t_2 \\ & \Big|\; t_1 | t_2 \;\Big|\; t \backslash L \;\Big|\; t[f] \;\Big|\; P(\overline{e}) \;\Big|\; (\mathsf{rec}\; P(\overline{x}).t)\,(\overline{e}) \end{array}$$

The notation $\overline{x}$ indicates a vector of variables, likewise $\overline{e}$ a vector of expressions. For term $(\mathsf{rec}\; P(\overline{x}).t)\,(\overline{e})$ to be well formed we require that $\overline{x}$ and $\overline{e}$ have the same number of elements and that each occurrence of P in t be applied to this same number of arguments. The term $c?x.t$ binds x in t, while the term $(\mathsf{rec}\; P(\overline{x}).t)\,(\overline{e})$ binds P and $\overline{x}$ in t. We assume the usual definitions of substitution (for process names and for variables) and closed terms; we denote the set of closed terms of VPL_I as $(p, q \in) Proc_I$ and call such terms *processes*.

3.2 Semantics

Before presenting the formal semantics of processes, we first give some intuition as to their behavior. Term nil represents the terminated process. The process $c?x.t$ is capable of receiving a value on channel c and subsequently behaves as t with the received value substituted for x. If the expression e is a constant (that is, $e \in Val_I$), then process $c!e.p$ will output e on channel c; otherwise, $c!e.p$ may spontaneously evolve to $c!v.p$ for any v in $\mathbb{E}_I[\![e]\!]$. We write the conditional as $be \rhd p \diamond q$; this process may have one or two possible internal moves, depending on the valuation of be. We use the symbol $[\!]$ to denote external choice and $\oplus$ to denote internal choice. $p|q$ denotes the parallel composition of p and q. The process $p\backslash L$ behaves as p with the exception that communication along channels in L is forbidden, and $p[f]$ behaves as p with the channels relabeling by f. Finally, $(\mathsf{rec}\; P(\overline{x}).p)\,(\overline{e})$ may spontaneously unfold, substituting $\overline{e}$ for $\overline{x}$.

This intuition is formalized in the transition relation $(\longrightarrow_I) \subseteq (Proc_I \times Act_I \times Proc_I)$, where $p \xrightarrow{\lambda}_I q$ holds if p is capable of executing action λ and evolving to q. To define this transition relation we first need some auxiliary notation. We use the overbar to indicate complementary communications; thus $\overline{c?v} = c!v$ and $\overline{c!v} = c?v$. Let the function $\mathsf{name} : (Act \mapsto Chan \cup \{\tau\})$ map communications to the channels on which they occur and τ to itself; for example, $\mathsf{name}(c?v) = c$ and $\mathsf{name}(\tau) = \tau$. The formal definition of $\longrightarrow_I$ is given in Table 1. We write $p \xrightarrow{\lambda}_I$ to abbreviate $(\exists q : \; p \xrightarrow{\lambda}_I q)$ and $p \stackrel{\lambda}{\not\longrightarrow}$ to abbreviate $\neg(p \xrightarrow{\lambda}_I)$. If $p \stackrel{\tau}{\not\longrightarrow}$ we say that p is *stable*.

Using this operational semantics, we may now defined a mapping from process terms to *labeled transition systems* as follows.

$$\textit{In}) \quad c?x.t \xrightarrow{c?v}_I t[v/x]\,,\ \textbf{for all}\ v \in \mathit{Val}_I$$

$$\textit{Out}) \quad c!e.p \xrightarrow{\tau}_I c!v.p\,,\ \ \textbf{if}\ e \notin \mathit{Val}_I \wedge v \in \mathbb{E}_I[\![e]\!] \qquad c!v.p \xrightarrow{c!v}_I p$$

$$\textit{Rec}) \quad (\mathsf{rec}\ P(\overline{x}).t)\,(\overline{e}) \xrightarrow{\tau}_I (t[\mathsf{rec}\ P(\overline{x}).t/P])[\overline{e}/\overline{x}]$$

$$\textit{Int}) \quad p \oplus q \xrightarrow{\tau}_I p \qquad\qquad p \oplus q \xrightarrow{\tau}_I q$$

$$\textit{Cond}) \quad be\ \rhd\ p\ \diamond q \xrightarrow{\tau}_I p\,,\ \textbf{if}\ tt \in \mathbb{B}_I[\![be]\!] \qquad be\ \rhd\ p\ \diamond q \xrightarrow{\tau}_I q\,,\ \textbf{if}\ f\!f \in \mathbb{B}_I[\![be]\!]$$

$$\textit{Ext}) \quad \frac{p \xrightarrow{\tau}_I p'}{p \mathbin{[\!]} q \xrightarrow{\tau}_I p' \mathbin{[\!]} q} \qquad \frac{q \xrightarrow{\tau}_I q'}{p \mathbin{[\!]} q \xrightarrow{\tau}_I p \mathbin{[\!]} q'} \qquad \frac{p \xrightarrow{a}_I p'}{p \mathbin{[\!]} q \xrightarrow{a}_I p'} \qquad \frac{q \xrightarrow{a}_I q'}{p \mathbin{[\!]} q \xrightarrow{a}_I q'}$$

$$\textit{Par}) \quad \frac{p \xrightarrow{\lambda}_I p'}{p|q \xrightarrow{\lambda}_I p'|q} \qquad \frac{q \xrightarrow{\lambda}_I q'}{p|q \xrightarrow{\lambda}_I p|q'} \qquad \frac{p \xrightarrow{a}_I p' \quad q \xrightarrow{\overline{a}}_I q'}{p|q \xrightarrow{\tau}_I p'|q'}$$

$$\textit{Res}) \quad \frac{p \xrightarrow{\lambda}_I p'}{p\backslash L \xrightarrow{\lambda}_I p'\backslash L}\ \mathsf{name}(\lambda) \notin L$$

$$\textit{Ren}) \quad \frac{p \xrightarrow{\lambda}_I p'}{p[f] \xrightarrow{f(\lambda)}_I p'[f]}$$

Table 1. Transition Rules for VPL_I, where $I = \langle\, \mathit{Val}_I, \mathbb{B}_I[\![\cdot]\!], \mathbb{E}_I[\![\cdot]\!]\rangle$

Definition 8. *A (rooted)* labeled transition system *over value interpretation I is a triple $\langle \Sigma, \sigma^0, \rightarrowtail \rangle$, where Σ is a set of states, $\sigma^0 \in \Sigma$ is an initial state, and $(\rightarrowtail) \subseteq (\Sigma \times Act_I \times \Sigma)$ is a transition relation. Let $(\mathcal{M}, \mathcal{N} \in) LTS_I$ be the set of all such labeled transition systems.*

We can now define the meaning of a process by mapping it to an element of LTS_I. Let $p \in Proc_I$; then $\mathbb{P}_I[\![p]\!] = \langle Proc_I, p, \longrightarrow_I \rangle$. We sometimes refer to $\mathbb{P}_I[\![p]\!]$ as the model *of p.*

As a matter of practical concern, we note that unreachable states and the edges connecting them may safely be eliminated $\mathbb{P}_I[\![p]\!]$.

3.3 Semantic ordering

In order to reason about the relative expressiveness of abstract process semantics, we need a preorder on transition systems that reflects the notion of approximation: if $\mathcal{M}$ is smaller than $\mathcal{N}$ in the preorder, then the behavior of $\mathcal{M}$ should approximate that of $\mathcal{N}$. For this purpose we use a variant of the *must preorder* of [14]. In addition to having a pleasing operational justification based on process testing, this preorder may be seen to relate processes on the basis of the safety and liveness properties that they enjoy. In order to define this relation, we first introduce the following definitions, which borrow heavily from [6, 13].

Definition 9. *Let I be an interpretation, let $\mathcal{M} = \langle \Sigma_{\mathcal{M}}, \sigma^0_{\mathcal{M}}, \rightarrowtail_{\mathcal{M}} \rangle$ be a transition system in LTS_I, let $\sigma, \varrho \in \Sigma_{\mathcal{M}}$, and let $s \in Comm^*_I$.*

- *The* trace relation, $(\Longrightarrow_{\mathcal{M}}) \subseteq (\Sigma_{\mathcal{M}} \times Comm_I^* \times \Sigma_{\mathcal{M}})$, *is defined inductively on the structure of* $Comm_I^*$ *as follows.*
  (a) $(\stackrel{\epsilon}{\Longrightarrow}_{\mathcal{M}}) = (\stackrel{\tau}{\longrightarrow}_{\mathcal{M}})^*$, *where* $(\stackrel{\tau}{\longrightarrow}_{\mathcal{M}})^*$ *is the transitive and reflexive closure of* $\stackrel{\tau}{\longrightarrow}_{\mathcal{M}}$.
  (b) $(\stackrel{as}{\Longrightarrow}_{\mathcal{M}}) = (\stackrel{\epsilon}{\Longrightarrow}_{\mathcal{M}}) \circ (\stackrel{a}{\longrightarrow}_{\mathcal{M}}) \circ (\stackrel{s}{\Longrightarrow}_{\mathcal{M}})$, *where* $\circ$ *denotes relational composition.*

- *The* convergence relation, $\downarrow^{\mathcal{M}} \subseteq (\Sigma_{\mathcal{M}} \times Comm_I^*)$, *is defined inductively as follows.*
  (a) $\sigma \downarrow^{\mathcal{M}} \epsilon$ *iff there is no infinite sequence* $\langle \sigma_i \rangle_{i \geq 1}$ *with* $\sigma \stackrel{\tau}{\longrightarrow}_{\mathcal{M}} \sigma_1$ *and* $\sigma_i \stackrel{\tau}{\longrightarrow}_{\mathcal{M}} \sigma_{i+1}$.
  (b) $\sigma \downarrow^{\mathcal{M}} as$ *iff* $\sigma \downarrow^{\mathcal{M}} \epsilon$ *and* $(\sigma \stackrel{a}{\Longrightarrow}_{\mathcal{M}} \varrho$ *implies* $\varrho \downarrow^{\mathcal{M}} s)$.

- *Let* $Event = \{c?, c! \mid c \in Chan\}$ *be the set of* events *and* $\sigma \in \Sigma_{\mathcal{M}}$. *Then the set events in which a process may initially engage is given by*

$$\mathsf{init}_{\mathcal{M}}(\sigma) = \{c? \mid \exists v \in Val_I\colon\ \sigma \stackrel{c?v}{\longrightarrow}_{\mathcal{M}}\} \cup \{c! \mid \exists v \in Val_I\colon\ \sigma \stackrel{c!v}{\longrightarrow}_{\mathcal{M}}\}.$$

  The acceptance set *of* σ *after a trace* s *is defined as follows.*

$$\mathsf{acc}_{\mathcal{M}}(\sigma, s) = \{\mathsf{init}_{\mathcal{M}}(\varrho) \mid \sigma \stackrel{s}{\Longrightarrow}_{\mathcal{M}} \varrho \ \ \&\ \ \varrho \stackrel{\tau}{\not\longrightarrow}_{\mathcal{M}}\}$$

- *Let* $AS, BS \subseteq \wp(Event)$*; then* $AS \subset\subset BS$ *iff* $\forall A \in AS\colon\ \exists B \in BS\colon\ B \subseteq A$.

Thus $\sigma \stackrel{s}{\Longrightarrow}_{\mathcal{M}} \varrho$ holds if σ can perform the actions listed in s with any number of intervening τ actions and end up as ϱ. We abbreviate $(\exists \varrho\colon\ \sigma \stackrel{s}{\Longrightarrow}_{\mathcal{M}} \varrho)$ as $\sigma \stackrel{s}{\Longrightarrow}_{\mathcal{M}}$. The predicate $\sigma \downarrow^{\mathcal{M}} s$ holds if σ is incapable of infinite internal computation at any point during its "executions" of s. The set $\mathsf{init}(\sigma)$ is the set of initial events of σ; we emphasize that this set includes no references to values. The acceptance set $\mathsf{acc}(\sigma, s)$ represents the set of "event capabilities" of σ after s. Note that the set of events does not depend on the value interpretation. Each set AS in $\mathsf{acc}(\sigma, s)$ corresponds to a state that σ may reach by executing s and contains the set of next possible events in that state. The fact that $\mathsf{acc}(\sigma, s)$ may contain more than one such set indicates that nondeterministic choices may occur during the execution of s; the more sets $\mathsf{acc}(\sigma, s)$ contains, the more nondeterministic σ is in its execution of s. Finally, the ordering $\subset\subset$ relates acceptance sets on the basis of their relative nondeterminism; intuitively, $AS \subset\subset BS$ if AS represents a "less nondeterministic" set of processes.

Notation. We write $\mathcal{M} \stackrel{s}{\Longrightarrow}$ for $\sigma^0_{\mathcal{M}} \stackrel{s}{\Longrightarrow}_{\mathcal{M}}$ and $\mathsf{lang}(\mathcal{M})$ for $\{s \mid \mathcal{M} \stackrel{s}{\Longrightarrow}\}$, the *language* of $\mathcal{M}$. We also write $\mathcal{M} \downarrow s$ for $\sigma^0_{\mathcal{M}} \downarrow^{\mathcal{M}} s$, and $\mathsf{acc}(\mathcal{M}, s)$ for $\mathsf{acc}_{\mathcal{M}}(\sigma^0_{\mathcal{M}}, s)$.

The *specification preorder* relates transition systems on the basis of their nondeterminism. Formally, it is defined as follows.

Definition 10. *Let* $\mathcal{M}, \mathcal{N} \in LTS_I$.

- $\mathcal{M} \sqsupseteq_I^{may} \mathcal{N}$ *iff* $\mathsf{lang}(\mathcal{N}) \subseteq \mathsf{lang}(\mathcal{M})$.
- $\mathcal{M} \sqsubseteq_I^{must} \mathcal{N}$ *iff for all* s $\mathcal{M} \downarrow s$ *implies* $(\mathcal{N} \downarrow s$ *and* $\mathsf{acc}(\mathcal{N}, s) \subset\subset \mathsf{acc}(\mathcal{M}, s))$.
- $\mathcal{M} \sqsubseteq_{\approx I} \mathcal{N}$ *iff* $\mathcal{M} \sqsupseteq_I^{may} \mathcal{N}$ *and* $\mathcal{M} \sqsubseteq_I^{must} \mathcal{N}$.
- $\mathcal{M} \approx_I \mathcal{N}$ *iff* $\mathcal{M} \sqsubseteq_{\approx I} \mathcal{N}$ *and* $\mathcal{N} \sqsubseteq_{\approx I} \mathcal{M}$.

It is traditional to abbreviate $\mathbb{P}_I[\![p]\!] \sqsubseteq_{\approx I} \mathbb{P}_I[\![q]\!]$, as $p \sqsubseteq_{\approx I} q$.

We now compare our semantics with the one given in [14]. There, value interpretations are assumed to be deterministic. Table 2 gives their formulation of the semantics. This definition of $\hookrightarrow$ may be substituted into our definitions for $\mathbb{P}_I[\![\cdot]\!]$ to generate new transition systems for processes and hence a new preorder on processes that we denote $\ll_I$. If the valuation functions are deterministic, then the preorders relate exactly the same terms.

Theorem 11. *Let* I *be such that the range of* $\mathbb{E}_I[\![\cdot]\!]$ *is* $\{\{v\} \mid v \in Val_I\}$ *and the range of* $\mathbb{B}_I[\![\cdot]\!]$ *is* $\{\{tt\}, \{ff\}\}$. *Then* $p \sqsubseteq_{\approx I} q$ *iff* $p \ll_I q$

All rules but (Out) and $(Cond)$ from table 1 with $\hookrightarrow$ replacing $\longrightarrow_I$

$$Out')\quad c!e.p \overset{c!v}{\hookrightarrow} p \ \textbf{ if } \ \mathbb{E}_I[\![e]\!] = \{v\}$$

$$Cond')\ \frac{p \overset{\lambda}{\hookrightarrow} p'}{be \rhd p \diamond q \overset{\lambda}{\hookrightarrow} p'}\ \mathbb{B}_I[\![be]\!] = \{tt\} \qquad \frac{q \overset{\lambda}{\hookrightarrow} q'}{be \rhd p \diamond q \overset{\lambda}{\hookrightarrow} q'}\ \mathbb{B}_I[\![be]\!] = \{f\!f\}$$

Table 2. Traditional semantics of value passing

We close this section by remarking on connections between $\sqsubsetsim_I$ and safety and liveness properties. Olderog and Hoare [20] present a framework for the consideration of safety and liveness in the context of labeled transition systems; they define a preorder that is similar to the specification preorder and show that if one transition system is less than another, then the higher one enjoys all the safety and liveness properties satisfied by the lower one, where safety and liveness properties are expressed in terms of traces, acceptance sets, et cetera. It can be shown that very similar results hold for $\sqsubsetsim_I$. For example, one may define a safety property $\mathcal{S}$ as any prefix-closed subset of $Comm_I^*$ (ie, the set of traces where the "bad thing" has not happened) and stipulate that a transition system $\mathcal{M}$ satisfies $\mathcal{S}$ iff $\mathsf{lang}(\mathcal{M}) \subseteq \mathcal{S}$. Then it follows that if $\mathcal{M} \sqsubsetsim_I \mathcal{N}$ and $\mathcal{M}$ satisfies $\mathcal{S}$, then $\mathcal{N}$ must satisfy $\mathcal{S}$ also.

In a similar vein, Olderog and Hoare characterize liveness properties as sets of transition systems.[4] Then $\mathcal{M}$ satisfies liveness property $\mathcal{L}$ iff for all *deterministic behaviors* $\mathcal{D}$ of $\mathcal{M}$, there exists a transition system $\mathcal{L}_i \in \mathcal{L}$ such that $\mathcal{L}_i \sqsubseteq^{must} \mathcal{D}$. Here a deterministic behavior is a restriction of a transition system such that the acceptance set after each trace contains a single set of events. If $\mathcal{M} \sqsubsetsim \mathcal{N}$ then one may show that the set of deterministic behaviors of $\mathcal{M}$ is also "less than" the deterministic behaviors of $\mathcal{N}$; this, along with the fact that the specification preorder is finer than the must preorder, allows us to conclude that if $\mathcal{M}$ satisfies a liveness property $\mathcal{L}$, then so does $\mathcal{N}$.

4 Abstractions of transition systems

In this section we show that, given a Galois insertion on sets of values, we may construct a Galois insertion on transition systems ordered by the specification preorder. As a consequence of this, we have (Corollary 16) that reasoning conducted on abstract transition systems carries over to their concretized counterparts. Corollary 17 then shows that abstractions may be composed, a result of practical importance in that it licenses the use of intermediate abstractions in reasoning about more abstract properties of systems.

Note first that Definitions 3 and 4 can be extended to define abstraction and concretization functions on all the sets of syntactic objects of VPL_I. Thus we have, for example:

$$\alpha\colon Comm_C^* \mapsto Comm_A^*,\quad \alpha\colon Act_C \mapsto Act_A,\quad \text{and}\quad \alpha\colon Proc_C \mapsto Proc_A.$$

Likewise we can extend Lemma 5 to cover process terms, so that

$$\langle \wp(Proc_C), \supseteq\rangle \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} \langle \wp(Proc_A), \supseteq\rangle.$$

[4] It is worth noting that deadlock-freedom is expressed as a liveness property in this framework, which is also powerful enough to express concepts such as eventuality and boundedness.

The definition of appropriate "liftings" of α and γ to transition systems, however, is less immediate. We first define abstraction. Since states in arbitrary transition systems have no structure, the labels on transitions are the only natural candidates for abstraction.

Definition 12. *Let $\mathcal{N} = \langle \Sigma_{\mathcal{N}}, \sigma^0_{\mathcal{N}}, \rightarrowtail_{\mathcal{N}} \rangle$ be a transition system in LTS_C. The abstraction of $\mathcal{N}$ to LTS_A is the transition system $\alpha(\mathcal{N}) = \langle \Sigma_{\mathcal{N}}, \sigma^0_{\mathcal{N}}, \rightarrowtail_{\mathcal{M}} \rangle$, where*

$$\sigma \overset{\alpha(\lambda)}{\rightarrowtail}_{\mathcal{M}} \varrho \ \textit{ iff } \ \sigma \overset{\lambda}{\rightarrowtail}_{\mathcal{N}} \varrho.$$

Clearly this abstraction preserves as much of the original meaning of the process as possible; in fact, these abstractions will be used as the standard by which to judge the abstract semantics of processes presented in the next section.

The definition of concretization for transition systems is more difficult. We want to define γ so that $\langle LTS_C, \sqsubseteq_C \rangle \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} \langle LTS_A, \sqsubseteq_A \rangle$. Our solution is to introduce new states into the concretization of a transition system. Each edge $\sigma \overset{\lambda}{\rightarrowtail} \varrho$ of the original system is replaced by two sets of edges: the first is a set of τ edges from σ to one of the new states; the second is a set of edges with labels from $\gamma(\lambda)$ that map back to ϱ. To ensure that the specification preorder is preserved, however, some care must be taken with the definitions. Throughout the remainder of this section, let $\mathcal{M} = \langle \Sigma_{\mathcal{M}}, \sigma^0_{\mathcal{M}}, \rightarrowtail_{\mathcal{M}} \rangle$ be a transition system over value interpretation A, and let $\sigma, \varrho \in \Sigma_{\mathcal{M}}$. We first introduce the concept of an image product.

Definition 13. *For each state σ, define the set of concretized consequents, $\mathsf{con}(\sigma)$ to be*

$$\mathsf{con}(\sigma) = \{\{\langle \sigma, \lambda, \varrho \rangle \colon \lambda' \in \gamma(\widehat{\lambda})\} \colon \langle \sigma, \widehat{\lambda}, \varrho \rangle \in (\rightarrowtail_{\mathcal{M}})\},$$

and let $X = \{\chi_1, \chi_2, \ldots\} = \mathsf{con}(\sigma)$. The image product *of σ is then defined:*

$$\mathsf{ip}(\sigma) = \{S \colon \ S \subseteq (\cup X) \ \textit{ and } \ |S \cap \chi_k| = 1\}$$

The image product of a state is the set of all possible combinations of concretized edges leaving the state. As an example of an image product, consider a state σ with two edges, $(\sigma \overset{a!\top}{\rightarrowtail}_{\mathcal{M}} \varrho_1)$ and $(\sigma \overset{b!\top}{\rightarrowtail}_{\mathcal{M}} \varrho_2)$. If $\gamma(\top) = \{0, 1\}$, then the image product of σ is the set

$$\begin{aligned} \{ \ &\{\langle \sigma, a!1, \varrho_1 \rangle, \langle \sigma, b!0, \varrho_2 \rangle\}, \{\langle \sigma, a!1, \varrho_1 \rangle, \langle \sigma, b!1, \varrho_2 \rangle\}, \\ &\{\langle \sigma, a!0, \varrho_1 \rangle, \langle \sigma, b!0, \varrho_2 \rangle\}, \{\langle \sigma, a!0, \varrho_1 \rangle, \langle \sigma, b!1, \varrho_2 \rangle\} \ \}. \end{aligned}$$

We can now formally define the concretization of a transition system as a bipartite graph. One set of states, including the root, is taken directly from the original system, while the other set of states is constructed using the image product.

Definition 14. *Let $\mathcal{M} = \langle \Sigma_{\mathcal{M}}, \sigma^0_{\mathcal{M}}, \rightarrowtail_{\mathcal{M}} \rangle$ be a transition system in LTS_A. The concretization of $\mathcal{M}$ to LTS_C is the transition system $\gamma(\mathcal{M}) = \langle \Sigma_{\mathcal{N}}, \sigma^0_{\mathcal{M}}, \rightarrowtail_{\mathcal{N}} \rangle$, where $\Sigma_{\mathcal{N}}$ and $\rightarrowtail_{\mathcal{N}}$ are defined as follows.*

$$\begin{aligned} \Sigma_{\mathcal{N}} \ &= \ \Sigma_{\mathcal{M}} \cup \left(\textstyle\bigcup_{\sigma \in \Sigma_{\mathcal{M}}} \mathsf{ip}(\sigma) \right) \\ \sigma \overset{\tau}{\rightarrowtail}_{\mathcal{N}} \pi \ &\textit{iff} \ \pi \in \mathsf{ip}(\sigma) \\ \pi \overset{\lambda}{\rightarrowtail}_{\mathcal{N}} \varrho \ &\textit{iff} \ (\exists \sigma \colon \ \langle \sigma, \lambda, \varrho \rangle \in \pi) \end{aligned}$$

From the definitions, we can derive the following.

$$\begin{array}{llll} \forall s \in Comm_C^*, \mathcal{N} \in LTS_C \colon & s \in \mathsf{lang}(\mathcal{N}) & \textit{implies} & \alpha(s) \in \mathsf{lang}(\alpha(\mathcal{N})), \\ \forall \widehat{s} \in Comm_A^*, \mathcal{M} \in LTS_A \colon & \widehat{s} \in \mathsf{lang}(\mathcal{M}) & \textit{iff} & \exists s \in \gamma(\widehat{s}) \colon \ s \in \mathsf{lang}(\gamma(\mathcal{M})). \end{array}$$

Similar results hold also for convergence and for acceptance sets, allowing us to conclude that α and γ do indeed form a Galois insertion over transition systems.

Theorem 15. $\langle LTS_C, \sqsubseteq_{\approx C} \rangle \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} \langle LTS_A, \sqsubseteq_{\approx A} \rangle$.

The following corollary implies in essence that if an "abstract" property holds of an abstracted system then the corresponding "concretized" property holds for the original system. Corollary 17 then shows how abstractions may be composed.

Corollary 16. *$\mathcal{M} \sqsubseteq_{\approx A} \alpha(\mathbb{P}_C[\![p]\!])$ iff $\gamma(\mathcal{M}) \sqsubseteq_{\approx C} \mathbb{P}_C[\![p]\!]$*

Corollary 17. *Let $\alpha = \delta \circ \beta$, where $\alpha : Val_C \mapsto Val_A$, $\beta : Val_C \mapsto Val_I$, and $\delta : Val_I \mapsto Val_A$. Then for $\mathcal{C} \in LTS_C$, $\mathcal{I} \in LTS_I$ and $\mathcal{A} \in LTS_A$, we have $\mathcal{I} \sqsubseteq_{\approx I} \beta(\mathcal{C})$ iff $\delta(\mathcal{I}) \sqsubseteq_{\approx A} \alpha(\mathcal{C})$. Moreover, if there exists $\gamma : A \mapsto I$ such that $\langle \wp(Val_I), \supseteq \rangle \underset{\gamma}{\overset{\delta}{\rightleftarrows}}$ then $\langle \wp(Val_A), \supseteq \rangle$,*

$$\mathcal{A} \sqsubseteq_{\approx A} \alpha(\mathcal{C}) \quad \textit{iff} \quad \gamma(\mathcal{A}) \sqsubseteq_{\approx I} \beta(\mathcal{C}).$$

The significance of Corollary 17 is twofold. First, it states that intermediate abstractions can be used to prove more abstract properties. Second, it states that properties that hold for the most abstract model also hold for models at intermediate levels of abstraction. This suggests, for example, that an interpretation that distinguishes some values may be used to prove properties that ignore values altogether. Thus, in order to prove properties of a concrete system, users of our framework may employ many abstraction functions, starting with the most abstract; if the desired then can be proven at the most abstract level, then the task is done, otherwise more and more concrete models may be used.

5 Abstract semantics

In the previous section we showed how to abstract the model of a process in such a way that properties of the abstract model hold also for the original. This technique, however, requires that the concrete model be constructed, an impossibility in the case that the concrete model is infinite state. In this section we advocate an alternative method: rather than abstracting the concrete model, one simply constructs a model using the abstract semantics of Table 1. We show that if the value abstraction is *safe*, then properties of the resulting abstract model will also hold for the concrete model. The advantage of this approach is clear: the concrete model need never be constructed.

5.1 Safety

Theorem 18. *If α is a safe value abstraction from C to A (Definition 6), then for all $p \in Proc_C$:*

$$\mathbb{P}_A[\![\alpha(p)]\!] \sqsubseteq_{\approx A} \alpha(\mathbb{P}_C[\![p]\!]).$$

Proof. By the definition of $\sqsubseteq_{\approx}$ it suffices to show that $\mathbb{P}_A[\![\alpha(p)]\!]$ is *may*-greater and *must*-less than $\alpha(\mathbb{P}_C[\![p]\!])$. Let $\mathcal{N} = \alpha(\mathbb{P}_C[\![p]\!])$ with transition relation $\rightarrowtail_{\mathcal{N}}$; by definition 8, the transition relation of $\mathbb{P}_A[\![\alpha(p)]\!]$ is $\longrightarrow_A$.

The *may*-inclusion requirement is satisfied iff iff $p \overset{s}{\Longrightarrow}_C$ implies $\alpha(p) \overset{\alpha(s)}{\Longrightarrow}_A$. We establish this last implication in Theorem 21 below.

Regarding the *must*-inclusion requirement, we have the following proof obligation.

$$\alpha(p) \downarrow^A \alpha(s) \text{ implies } \; p \downarrow^C s \;\; \& \;\; \mathsf{acc}_C(p, s) \subset\subset \mathsf{acc}_A(\alpha(p), \alpha(s))$$

Theorems 22 and 23, below, establish that this obligation is indeed met. □

In the rest of this subsection we sketch the proofs of Theorems 21-23. To begin with we must establish a relationship between the transition relations $\longrightarrow_C$ and $\longrightarrow_A$. Lemma 19 states that every edge in $\longrightarrow_C$ is matched by an edge in $\longrightarrow_A$. Lemma 20 shows that if p is a stable process under $\longrightarrow_C$, then $\alpha(p)$ can reach (via $\longrightarrow_A$) some stable state whose initial event capabilities are a subset of those available to p.

Lemma 19. $\forall p, q, \lambda$: $\; p \xrightarrow{\lambda}_C q$ *implies* $\; \alpha(p) \xrightarrow{\alpha(\lambda)}_A \alpha(q)$

Lemma 20. $\forall p$: $\; p \overset{\tau}{\not\longrightarrow}_C$ *implies* $\; (\exists \widehat{p}\colon \alpha(p) \overset{\epsilon}{\Longrightarrow}_A \widehat{p} \overset{\tau}{\not\longrightarrow}_A$ *and* $\mathsf{init}_A(\widehat{p}) \subseteq \mathsf{init}_C(p))$

Lemma 19 is proved by induction over the structure of process terms. The proof makes use of the safety of α and requires that substitution be well behaved with respect to abstraction. The proof of Lemma 20 also proceeds by structural induction and uses Lemma 19.

Theorem 21 establishes that every trace of p is matched by a trace of $\alpha(p)$. Note that the converse does not hold in general since the abstraction of the conditional may introduce new traces into the language of $\alpha(p)$. The proof is by induction on the length of the trace s; both the basis and induction steps follow immediately from Lemma 19.

Theorem 21. $\forall p, q, s$: $\; p \overset{s}{\Longrightarrow}_C q$ *implies* $\; \alpha(p) \overset{\alpha(s)}{\Longrightarrow}_A \alpha(q)$

Theorem 22 states that if the abstraction of a process converges on a trace then so must the original process. Again the converse does not hold in general, as can be seen by considering the process $((1 = 1) \rhd \mathsf{nil} \diamond \mathsf{rec}\ P.P)$ under an abstraction that evaluates all boolean expressions to $\{tt, ff\}$. Theorem 23 provides the final piece of the puzzle.

Theorem 22. $\forall p, s$: $\; \alpha(p) \downarrow^A \alpha(s)$ *implies* $\; p \downarrow^C s$

Theorem 23. $\forall p, s$: $\; \alpha(p) \downarrow^A \alpha(s)$ *implies* $\; \mathsf{acc}_C(p, s) \subset\subset \mathsf{acc}_A(\alpha(p), \alpha(s))$

Assuming $\alpha(p) \downarrow^A \alpha(s)$, the proof obligation for Theorem 23 can be written as:

$$p \overset{s}{\Longrightarrow}_C q \overset{\tau}{\not\longrightarrow}_C \;\text{ implies }\; (\exists \widehat{q}\colon \alpha(p) \overset{\alpha(s)}{\Longrightarrow}_A \widehat{q} \overset{\tau}{\not\longrightarrow}_A \;\&\; \mathsf{init}_A(\widehat{q}) \subseteq \mathsf{init}_A(q)).$$

The proof is again by induction on s. The basis case $(s = \epsilon)$ requires a further induction on the length of the longest initial τ-sequence of p. That there can be no infinite τ sequence is established by the premise and the fact that the model of a process is image finite (see [14]).

5.2 Optimality

In order to prove optimality, we must first lift $\mathbb{P}_I[\![\cdot]\!]$ to sets of process terms. To this end we introduce the following operator on transition systems.

Definition 24. *Given a set of transition systems* $\{\mathcal{M}_1, \mathcal{M}_2, \ldots\}$ *where* $\mathcal{M}_k = \langle \Sigma_k, \sigma_k^0, \rightarrowtail_k \rangle \in LTS_I$, *define the* internal sum *of the set to be:* $\bigoplus_k \mathcal{M}_k = \langle \Sigma, \sigma^0, \rightarrowtail \rangle$, *where* σ^0 *is a fresh state,* $\Sigma = \bigcup_k \{\langle k, \sigma \rangle \mid \sigma \in \Sigma_k\} \cup \{\sigma^0\}$, *and* $\rightarrowtail$ *is defined as follows.*

$$\sigma^0 \overset{\tau}{\rightarrowtail} \langle k, \sigma \rangle \text{ iff } \sigma = \sigma_k^0$$
$$\langle k, \sigma \rangle \overset{\lambda}{\rightarrowtail} \langle j, \varrho \rangle \text{ iff } k = j \;\;\&\;\; \sigma \overset{\lambda}{\rightarrowtail}_k \varrho$$

The internal sum of a set of processes is the greatest lower bound of these processes with respect to the specification preorder. The meaning of a set of process terms can now be defined, for $PS \subseteq Proc_I$, as follows.

$$\mathbb{P}_I[\![PS]\!] = (\bigoplus_{p \in PS} \mathbb{P}_I[\![p]\!])$$

The following theorem establishes that, given an optimal abstract value interpretation, $\mathbb{P}_A[\![\cdot]\!]$ is optimal for $\mathbb{P}_C[\![p]\!]$. It follows from Theorems 15 and 18.

Theorem 25. *If α is an optimal value abstraction from C to A (Definition 7), then for all $p \in Proc_A$:*

$$\mathbb{P}_A[\![p]\!] \approx_A \alpha(\mathbb{P}_C[\![\gamma(p)]\!]).$$

5.3 Exact analysis

Even if an abstract semantics is not optimal, there may still be processes for which the abstract semantics is "exact". To end the section, we give a sufficient condition for establishing that this is the case. The condition is a natural generalization of *data-independence* as studied by Wolper [29]. We need the following definitions.

Definition 26. *Let $\alpha : C \mapsto A$ and $f : Val_C \mapsto Val_C$. Then f* respects α *if for every v in Val_C, $\alpha(v) = \alpha(f(v))$. Extend f to process terms as for α in Definition 3. Then a process $p \in Proc_C$ is* α-independent *if for all f respecting α, $\alpha(\mathbb{P}_C[\![p]\!]) \approx_A \alpha(\mathbb{P}_C[\![f(p)]\!])$.*

Intuitively, p is α-independent if its behavior modulo α is independent of specific values, modulo α.

Theorem 27. *If $\alpha : C \mapsto A$ is safe and $p \in Proc_C$ is α-independent, then*

$$\mathbb{P}_A[\![\alpha(p)]\!] \approx_A \alpha(\mathbb{P}_C[\![p]\!]).$$

6 Example

In this section we give a small example illustrating the utility of our results. Consider the following system consisting of a router and two processing units. The router waits for a value, which is a natural number, to arrive on its in channel; it then routes the (halved) value to the "left" processing unit if the original value is even and to the right otherwise. (Thus the least significant bit of the value may be thought of as an "address".) Assume that the value interpretation C is the standard one for natural numbers. The VPL process describing this system may be given as follows.

$$\begin{aligned}
\mathsf{Router} &= \mathsf{in}?(v).((v \bmod 2) = 0) \ \triangleright\ \mathsf{left}!(v/2).\mathsf{Router} \diamond \mathsf{right}!(v/2).\mathsf{Router} \\
\mathsf{Unit}_0 &= \mathsf{in}?(v).\mathsf{out}!(f(v)).\mathsf{Unit}_0 \\
\mathsf{Unit}_1 &= \mathsf{in}?(v).\mathsf{out}!(g(v)).\mathsf{Unit}_1 \\
\mathsf{System} &= (\mathsf{Router} \mid \mathsf{Unit}_0[\mathsf{left}/\mathsf{in}] \ \mid\ \mathsf{Unit}_1[\mathsf{right}/\mathsf{in}]) \backslash \{\mathsf{left}, \mathsf{right}\}
\end{aligned}$$

We would like to determine whether the above system is deadlock-free. Unfortunately, its state space is infinite, and naive state-space enumeration techniques would not terminate. The results in this paper suggest, however, that if we can come up with a safe abstraction on values and establish that the resulting abstracted process is deadlock-free, then so is the original system. That is, letting A be the abstract interpretation and α the abstraction from C to A, it follows from the fact that $\mathbb{P}_A[\![p]\!] \sqsubseteq_A \alpha(\mathbb{P}_C[\![p]\!])$ that $\mathbb{P}_C[\![p]\!]$ deadlocks if and only if $\alpha(\mathbb{P}_C[\![p]\!])$ does.

Consider the trivial abstract value space A in which all concrete values are collapsed into a single abstract value 0, every expression evaluates to 0, and every boolean evaluates to the set $\{ff, tt\}$. The abstraction function α that maps the concrete interpretation into this interpretation is clearly safe. When we apply this abstraction to the above system, we get a system that is semantically equivalent to the following.

$$\begin{aligned}
\mathsf{Router}_A &= \mathsf{in?0.(left!0.Router} \oplus \mathsf{right!0.Router)} \\
\mathsf{Unit}_A &= \mathsf{in?0.out!0.Unit}_A \\
\mathsf{System}_A &= (\mathsf{Router} \mid \mathsf{Unit}_A[\mathsf{left/in}] \mid \mathsf{Unit}_A[\mathsf{right/in}]) \backslash \{\mathsf{left}, \mathsf{right}\}
\end{aligned}$$

This system is finite-state, and using reachability analysis one may determine that it is deadlock-free. Accordingly, it follows that the original system is also deadlock-free.

7 Discussion

In this paper we have shown how abstractions on values may be used to generate abstractions on processes that pass values. We defined the semantics of processes parametrically with respect to a value interpretation and showed that safe value abstractions induce safe abstract semantic functions and optimal value abstractions likewise induce optimal semantic functions. We proved our results relative to the *specification preorder* which preserves not only safety properties, but also liveness properties.

One may use our technique to simplify the task of reasoning about value-passing systems as follows. Given a system and a safety or liveness property, one may first attempt to establish satisfaction using the most abstract value interpretation that is exact with respect to the specification. If satisfaction can be shown, the task is finished; otherwise, one can select a less abstract interpretation and repeat the analysis. The hope is that one finds a value interpretation that is concrete enough to prove the property desired, yet abstract enough so that satisfaction is (rapidly) computable. This process would be facilitated by an environment providing a library of interpretations, along with tools capable of analyzing processes to suggest which of the more concrete interpretations available should be chosen in the event that verification fails, and we would like to pursue the development of such an environment as future work. It would also be interesting to characterize the properties preserved by the specification preorder in terms of a temporal logic; one candidate would appear to be linear time temporal logic without a next operator. We also would like to investigate the addition of values with structure to our model. The extension to disjoint sums (for example, the set of integers and characters) is straight-forward. More challenging are sets of values whose elements are ordered, as are the denotations of functions in domain theory. A solution here, however, would open up the possibility of treating higher-order value passing languages. To this end it would be useful to cast our results in terms of *acceptance trees* [14]; this rephrasing should not present difficulties. Finally, we intend to further explore the connections between our approach and effect systems.

Related Work. Existing work on abstractions of concurrent systems has focused on the development of abstraction techniques that preserve certain classes of formulas in various temporal logics [1, 3, 10]. The frameworks of these papers differ in details, but each considers how to generate, from an abstraction on values, abstractions on Kripke structures that preserve various fragments of the temporal logic CTL^*. Their programming models have focused on shared memory, whereas our considers value-passing; in addition, their semantics are based on the *simulation* preorder, which is incomparable to the preorder used here [26]. Consequently, the "properties" that are preserved would in general be different. Characterizing these differences precisely remains a topic that needs to be addressed.

The goals of our work are also similar to those of Hennessy and Lin in their work on *symbolic bisimulations* [15]. Central to their work is the notion of a symbolic transition system, which is a transition system with sets of free variables as states and guarded expressions as edges. Symbolic transition systems are often finite, but even trivial recursive processes whose arguments vary from call to call may have infinite symbolic transition systems, rendering their technique ineffective. For example $(\text{rec } P(x).c!x.P(x+1)\ )(0)$ has an infinite symbolic transition system; our method will produce a finite transition system for this process, given a finite abstract value set.

Our work is also related to that done by Nielson and Nielson on effect systems for CML [19]. An *effect system* is an extension of a conventional type system that describes the side-effects (in our case, events) that a program may have. In the case that the properties of interest can be described using the trivial abstraction, our method reduces to an effect system for process, in the spirit of the Nielsons' work. Their language is much more complex, supporting higher-order and structured values. However, our abstractions preserve more of the behavior of the original process than do theirs; for example, their abstractions reduce internal to external non-determinism.

As for more applied work, Yeh and Young [30] have used an approach that can be seen as an instance of ours to ours for verifying properties of Ada programs. Their success points to the practical importance of our technique.

References

1. S. Bensalem, A. Bouajjani, C. Loiseaux, and J. Sifakis. Property-preserving simulations. In *Proceedings of the Workshop on Computer-Aided Verification*, volume 663 of *LNCS*, pages 260—273. Springer-Verlag, June 1992.
2. B. Bloom and R. Paige. Computing ready simulations efficiently. In *Proceedings of the North American Process Algebra Workshop*, Workshops in Computing, pages 119–134, Stony Brook, New York, August 1992. Springer-Verlag.
3. Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. In *Proceedings ACM POPL*, pages 343–354, January 1992.
4. E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM TOPLAS*, 8(2):244–263, April 1986.
5. R. Cleaveland. Tableau-based model checking in the propositional mu-calculus. *Acta Informatica*, 27(8):725–747, September 1990.
6. R. Cleaveland and M.C.B. Hennessy. Testing equivalence as a bisimulation equivalence. *Formal Aspects of Computing*, 5:1–20, 1993.
7. R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench: A semantics-based tool for the verification of finite-state systems. *ACM TOPLAS*, 15(1):36–72, January 1993.
8. R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. *Formal Methods in System Design*, 2:121–147, 1993.
9. P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In *PLILP '92*, volume 631 of *LNCS*, pages 269–295. Springer-Verlag, August 1992.
10. D. Dams, O. Grumberg, and R. Gerth. Abstract interpretation of reactive systems: Abstractions preserving $\forall$CTL*, $\exists$CTL* and CTL*. In *PROCOMET '94*, IFIP Transactions. North-Holland/Elsevier, June 1994. Full version available from Eindhoven University of Technology.
11. J. Goyer. Communications protocols for the B-HIVE multicomputer. Master's thesis, North Carolina State University, 1991.
12. E. Harcourt, J. Mauney, and T. Cook. Specification of instruction-level parallelism. In *Proceedings of the North American Process Algebra Workshop*, August 1993. Technical Report TR93-1369, Cornell University.
13. M.C.B. Hennessy. *Algebraic Theory of Processes*. MIT Press, Boston, 1988.
14. M.C.B. Hennessy and A. Ingólfsdóttir. A theory of communicating processes with value-passing. *Information and Computation*, 107:202–236, December 1993.

15. M.C.B. Hennessy and H. Lin. Symbolic bisimulations. Technical Report 1/92, Sussex University, 1992.
16. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, London, 1985.
17. N.D. Jones and F. Nielson. *Abstract Interpretation: a Semantics-Based Tool for Program Analysis.* Handbook of Theoretical Computer Science. Oxford, To appear.
18. P. Kanellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, May 1990.
19. F. Nielson and H. Nielson. From CML to process algebras. In E. Best, editor, *Proceedings of CONCUR '93*, volume 715 of *LNCS*, pages 493–508, Hildesheim, Germany, August 1993. Springer-Verlag.
20. E.-R. Olderog and C.A.R. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica*, 23:9–66, 1986.
21. R. Paige and R.E. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16(6):973–989, December 1987.
22. J. Parrow. Verifying a CSMA/CD-protocol with CCS. In *Proceedings of the IFIP Symposium on Protocol Specification, Testing and Verification*, pages 373–387, Atlantic City, New Jersey, June 1988. North-Holland.
23. J. Richier, C. Rodgriguez, J. Sifakis, and J. Voiron. Verification in XESAR of the sliding window protocol. In *Proceedings of the IFIP Symposium on Protocol Specification, Testing and Verification*, pages 235–250, Zurich, May 1987. North-Holland.
24. V. Roy and R. de Simone. Auto/Autograph. In *Computer-Aided Verification '90*, pages 235–250, Piscataway, New Jersey, July 1990. American Mathematical Society.
25. C. Stirling and D. Walker. Local model checking in the modal mu-calculus. In *TAPSOFT '89*, volume 352 of *LNCS*, pages 369–383, Barcelona, March 1989. Springer-Verlag.
26. R. van Glabbeek. The linear time – branching time spectrum. In *Proceedings of CONCUR '90*, volume 458 of *LNCS*, pages 278–297. Springer-Verlag, August 1990.
27. M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the Symposium on Logic in Computer Science*, pages 332–344, Cambridge, Massachusetts, June 1986. Computer Society Press.
28. G. Winskel. A note on model checking the modal ν-calculus. In *Proceedings ICALP*, volume 372 of *LNCS*, pages 761–772, Stresa, Italy, July 1989. Springer-Verlag.
29. P. Wolper. Expressing interesting properties of programs in propositional temporal logic. In *Proceedings ACM POPL*, pages 184–193, January 1986.
30. W.J. Yeh and M. Young. Compositional reachability analysis using process algebra. In *TAV '91*, pages 49–59. ACM SIGSOFT, ACM Press, October 1991.