# A Theory of Normed Simulations

DAVID GRIFFIOEN and FRITS VAANDRAGER
University of Nijmegen

In existing simulation proof techniques, a single step in a lower-level specification may be simulated by an extended execution fragment in a higher-level one. As a result, it is cumbersome to mechanize these techniques using general purpose theorem provers. Moreover, it is undecidable whether a given relation is a simulation, even if tautology checking is decidable for the underlying specification logic. This paper studies various types of *normed simulations*. In a normed simulation, each step in a lower-level specification can be simulated by at most one step in the higher-level one, for any related pair of states. In earlier work we demonstrated that normed simulations are quite useful as a vehicle for the formalization of refinement proofs via theorem provers. Here we show that normed simulations also have pleasant theoretical properties: (1) under some reasonable assumptions, it is decidable whether a given relation is a normed forward simulation, provided tautology checking is decidable for the underlying logic; (2) at the semantic level, normed forward and backward simulations together form a complete proof method for establishing behavior inclusion, provided that the higher-level specification has finite invisible nondeterminism.

Categories and Subject Descriptors: F.1.1 [**Computation by abstract devices**]: Models of Computation—*Automata*; F.3.1 [**Logics and meanings of programs**]: Specifying and Verifying and Reasoning about Programs

General Terms: Theory, Verification

Additional Key Words and Phrases: Automata, backward simulations, computer aided verification, forward simulations, history variables, normed simulations, prophecy variables, refinement mappings

## 1. INTRODUCTION

Simulation relations and refinement functions are widely used to prove that a lower-level specification of a reactive system correctly implements a higher-level one [Jonsson 1994; Lynch 1996; Roever and Engelhardt 1998]. Proving soundness and completeness of proof rules for simulation and refinement has attracted the attention of many researchers in the past two or three decades [Milner 1971; Lamport 1983; Jonsson 1985; Lynch and Tuttle 1987; Stark 1988; Klarlund and Schneider 1989; 1993; Jonsson 1990; 1991; Abadi and Lamport 1991; Lynch and Vaandrager 1995]. The usefulness of all these proof methods was demonstrated by their proposers, who applied them to often highly nontrivial case studies. However, all these refine-

Author's address: Nijmeegs Instituut voor Informatica en Informatiekunde, University of Nijmegen, P.O. Box 9010, 6500 GL Nijmegen, The Netherlands, E-mail `griffioen42@zonnet.nl` and `fvaan@cs.kun.nl`. A preliminary version of this paper appeared as Sections 1 and 2 in [Griffioen and Vaandrager 1998]. Research supported by the Netherlands Organization for Scientific Research (NWO) under contract SION 612-316-125.

ment/simulation proofs were done manually, and they were typically quite long and tedious. The field has come to realize that if we want to scale up these methods to larger examples, it really matters that the semantical analysis can be carried out with the help of a software tool that requires little or no human intervention. This led Wolper [1997] to propose the following criterion for "formal" methods

> **Criterion of Semantical Computational Support:** *A formal method provides semantical computational support of it allows software tools for checking semantical properties of specifications.*

Several incomplete refinement/simulation proof rules have been mechanized successfully [Helmink et al. 1994; Nipkow and Slind 1995; Devillers et al. 2000]. A mechanization of a complete set of simulation rules is reported by Sogaard-Andersen et al. [1993], but in this approach the verification process is highly interactive and it does not satisfy Wolper's criterion of semantical computational support. In fact, we believe it will be difficult to efficiently mechanize any of the above mentioned complete proof methods using a general purpose theorem prover: too much user interaction will be required. Earlier [Griffioen and Vaandrager 1998; Griffioen 2000][Chapter 6], we proposed a proof method based on *normed simulations* and showed that it can be mechanized efficiently using PVS. In the present paper we study the theoretical properties of normed simulations. In particular, we establish that normed forward and backward simulations together form a complete proof method for establishing behavior inclusion. Before we discuss the technical contributions of this paper in more detail, we first describe the problem that arises in the mechanization of existing complete proof methods, and how this can be solved using normed simulations.

Technically, a *simulation* (or *refinement*) is a relation (or function) $R$ between the states of a lower-level specification $A$ and a higher-level specification $B$, that satisfies a condition like

$$(s, u) \in R \;\wedge\; s \xrightarrow{a}_A t \;\Rightarrow\; \exists v : u \xrightarrow{a}_B v \;\wedge\; (t, v) \in R \tag{1}$$

(If lower-level state $s$ and higher-level state $u$ are related, and in $A$ there is a transition from $s$ to $t$, then there is a matching transition in $B$ from $u$ to a state $v$ that relates to $t$; see also Figure 1.) The existence of a simulation implies that any behavior of $A$ can also be exhibited by $B$.

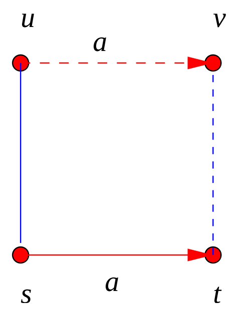


Fig. 1. Transfer condition (1).

The main reason why simulations are useful is that they reduce *global* reasoning about behaviors and executions to *local* reasoning about states and transitions. However, to the best of our knowledge, all complete simulation proof methods that appear in the literature fall back on some form of global reasoning in the case of specifications containing internal (or stuttering) transitions. The usual transfer condition for *forward simulations* [Lynch and Vaandrager 1995], for instance, says

$$(s,u) \in R \,\wedge\, s \xrightarrow{a}_A t \;\Rightarrow\; \exists \text{ execution fragment } \alpha : \mathit{first}(\alpha) = u \\ \wedge\; \mathit{trace}(\alpha) = \mathit{trace}(a) \,\wedge\, (t, \mathit{last}(\alpha)) \in R \tag{2}$$

(Each lower-level transition can be simulated by a sequence of higher-level transitions which, apart from the action that has to be matched, may also contain an arbitrary number of internal "$\tau$" transitions; see also Figure 2.) Thus the research program to reduce global reasoning to local reasoning has not been carried out to its completion. In manual proofs of simulation relations, this is usually not

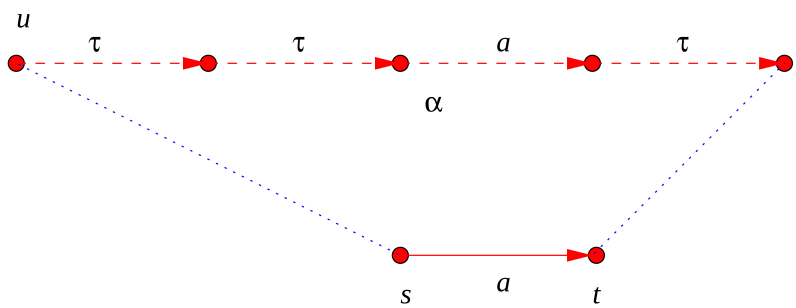


Fig. 2. Transfer condition (2).

a problem: in practice lower-level transitions are typically matched by at most one higher-level transition; moreover humans tend to be quite good in reasoning about sequences, and move effortlessly from transitions to executions and back. In contrast, it turns out to be rather cumbersome to formalize arguments involving sequences using existing theorem provers [Devillers et al. 1997]. In fact, in several papers in which formalizations of simulation proofs are described, the authors only consider a restricted type of simulation in which each lower-level transition is matched by at most one higher-level transition [Helmink et al. 1994; Nipkow and Slind 1995; Devillers et al. 2000]. However, there are many examples of situations where these restricted types of simulations cannot be applied. In approaches where the full transfer condition (2) is formalized [Søgaard-Andersen et al. 1993], the user has to supply the simulating execution fragments $\alpha$ to the prover explicitly, which makes the verification process highly interactive. Jonsson [1990] presents a variant of the completeness theorem of Abadi and Lamport [1991] in terms of certain forward and backward simulations in which lower-level transitions are matched by at most one higher-level transition. However, his completeness result is only partial in the sense that he requires that the higher-level automaton contains no non-trivial $\tau$-steps. In our view this restriction is problematic, especially in a stepwise refinement approach where the higher-level specification in one design step may be the lower-level specification from a previous design step. All the complications that we address in our paper are due to the possible presence of internal actions in the higher-level automaton.

In this paper, we study a simulation proof method which remedies the above problems. The idea is to define a function $n$ that assigns a norm $n(s \xrightarrow{a} t, u)$, in some well-founded domain, to each pair of a transition in $A$ and a state of $B$. If $u$ has to simulate transition $s \xrightarrow{a} t$ then it may either do nothing (if $a$ is internal and $t$ is related to $u$), or it may do a matching $a$-transition, or it may perform an internal transition $u \xrightarrow{b} v$ such that the norm decreases, i.e.,

$$n(s \xrightarrow{a} t, v) \ < \ n(s \xrightarrow{a} t, u).$$

We establish that *normed forward simulations* and *normed backward simulations* together constitute a complete proof method for establishing trace inclusion. In addition we show how *history* and *prophecy relations* (which are closely related to history and prophecy variables [Abadi and Lamport 1991]) can be enriched with a norm function, to obtain another complete proof method in combination with a simple notion of refinement mapping.

The preorders generated by normed forward simulations are strictly finer than the preorders induced by Lynch and Vaandrager's forward simulations [1995]. In fact, we will characterize normed forward simulations in terms of *branching forward simulations* [Glabbeek and Weijland 1996], and present a similar characterization for the backward case. It is possible to come up with a variant of normed forward simulation that induces the same preorder as forward simulations, but technically this is somewhat more involved [Griffioen 2000][Section 6.5.10].

When proving invariance properties of programs, one is faced with two problems. The first problem is related to the necessity of proving tautologies of the assertion logic, whereas the second manifests in the need of finding sufficiently strong invariants. In order to address the first problem, powerful decision procedures have been incorporated in theorem provers such as PVS [Owre et al. 1995]. If tautology checking is decidable then it is decidable whether a given state predicate is valid for the initial states and preserved by all transitions. The task of finding such a predicate, i.e. solving the second problem, is in most cases still the responsibility of the user, even though some very powerful heuristics have been devised to support and automate the search [Bensalem et al. 1996; Manna et al. 1998; Lakhnech et al. 2001; Bensalem et al. 2000]. Analogously, if specifications $A$ and $B$, a conjectured forward simulation relation $R$ and norm function $n$ can all be expressed within a decidable assertion logic, and if the specification of $B$ only contains a finite number of deterministic transition predicates, then it is decidable whether the pair $(R, n)$ is a normed forward simulation. This result, which does not hold for earlier approaches such as [Lynch and Vaandrager 1995], is a distinct advantage of normed forward simulations.

The idea of using norm functions to prove simulation relations was also developed by Groote and Springintveld [1995], who used it to prove branching bisimilarity in the context of the process algebra $\mu$CRL. However, their norm function is defined on the states of $B$ only and does not involve the transitions of $A$. As a consequence, their method does not always apply to diverging processes. Norm functions very similar to ours were also studied by Namjoshi [1997]. He uses them to obtain a characterization of the stuttering bisimulation of Browne et al. [1988], which is the equivalent of branching bisimulation in a setting where states rather than actions are labeled [De Nicola and Vaandrager 1995]. Neither Groote and Springintveld [1995], nor Namjoshi [1997] address effectiveness issues. Although we present normed simulations in a setting of labeled transition systems, it should not be difficult to transfer our results to a process algebraic setting such as that of Groote and Springintveld [1995] or a state based setting such as Namjoshi's [1997]. Inspired by our approach, norm functions have been used by Baier and Stoelinga [2000] to define a new bisimulation equivalence for probabilistic systems.

In this paper, we only present maximally simple examples to illustrate the various definitions and results. Earlier [Griffioen and Vaandrager 1998; Griffioen 2000][Chapter 6], we used normed simulations in a substantial case study, namely the verification of the leader election protocol that is part of the IEEE 1394 "Firewire" standard. This verification has been mechanically checked using PVS.[1]

In the presentation of our results, we will closely follow Lynch and Vaandrager [1995] and stick to their notations. In fact, our aim will be (amongst others) to derive analogous results to theirs, only for different types of simulations. However, we decided not to present normed versions of their forward-backward and backward-forward simulations of, since these simulations have thus far not been used in practice and technically this would bring nothing new. Apart from the notion of a norm function, a major technical innovation in the present paper is a new, simple definition of execution correspondence [Gawlick et al. 1993; Søgaard-Andersen et al. 1993], and the systematic use of this concept in the technical development. Although here we only address simulation proof techniques for establishing safety, we expect that based on the execution correpondence lemma's that we prove it will be easy to generalize our results to a setting with liveness properties. We leave it as a topic for future research to substantiate this claim.

## 2. PRELIMINARIES

In this section, we briefly recall some basic concurrency theory definitions [Lynch and Vaandrager 1995]. An *automaton* (or *labeled transition system*) $A$ consists of

—a (possibly infinite) set $states(A)$ of states,

—a nonempty set $start(A) \subseteq states(A)$ of start states,

—a set $acts(A)$ of actions that includes the internal (or stuttering) action $\tau$, and

—a set $steps(A) \subseteq states(A) \times acts(A) \times states(A)$ of steps.

Write $s \xrightarrow{a}_A t$ as a shorthand for $(s, a, t) \in steps(A)$. We let $ext(A)$, the *external actions*, denote $acts(A) - \{\tau\}$. An *execution fragment* of $A$ is a finite or infinite alternating sequence, $s_0 a_1 s_1 a_2 s_2 \cdots$, of states and actions of $A$, beginning with a state, and if it is finite also ending with a state, such that for all $i > 0$, $s_{i-1} \xrightarrow{a_i} s_i$. An *execution* of $A$ is an execution fragment that begins with a start state. We denote by $execs^*(A)$ and $execs(A)$ the sets of finite and all executions of $A$, respectively. A state $s$ of $A$ is *reachable* if $s$ occurs as the last state in some finite execution $\alpha$ of

[1]Actually, we discovered the notion of a normed simulation while formalizing the correctness proof of this leader election protocol.

$A$. In this case we write *reachable*$(A, s)$. Also, we write *reachable*$(A)$ for the set of reachable states of $A$.

The *trace* of an execution fragment $\alpha$, notation *trace*$(\alpha)$, is the subsequence of non-$\tau$ actions occurring in $\alpha$. A finite or infinite sequence $\beta$ of external actions is a *trace* of $A$ if $A$ has an execution $\alpha$ with $\beta = \mathit{trace}(\alpha)$. Write *traces*$^*(A)$ and *traces*$(A)$ for the sets of finite and all traces of $A$, respectively. Write $A \leq_{*\mathrm{T}} B$ if $\mathit{traces}^*(A) \subseteq \mathit{traces}^*(B)$, and $A \leq_{\mathrm{T}} B$ if $\mathit{traces}(A) \subseteq \mathit{traces}(B)$.

Suppose $A$ is an automaton, $s$ and $t$ are states of $A$, and $\beta$ is a finite sequence over *ext*$(A)$. We say that $(s, \beta, t)$ is a *move* of $A$, and write $s \overset{\beta}{\Longrightarrow}_A t$, or just $s \overset{\beta}{\Longrightarrow} t$ when $A$ is clear, if $A$ has a finite execution fragment $\alpha$ that starts in $s$, has trace $\beta$ and ends in $t$.

Three restricted kinds of automata play an important role in this paper:

(1) $A$ is *deterministic* if $|\mathit{start}(A)| = 1$, and for any state $s$ and any finite sequence $\beta$ over *ext*$(A)$, there is at most one state $t$ such that $s \overset{\beta}{\Longrightarrow} t$. A deterministic automaton is characterized uniquely by the properties that $|\mathit{start}(A)| = 1$, every $\tau$-step is of the form $(s, \tau, s)$ for some $s$, and for each state $s$ and each action $a$ there is at most one state $t$ such that $s \xrightarrow{a}_A t$.
(2) $A$ has *finite invisible nondeterminism (fin)* if *start*$(A)$ is finite, and for any state $s$ and any finite sequence $\beta$ over *ext*$(A)$, there are only finitely many states $t$ such that $s \overset{\beta}{\Longrightarrow}_A t$.
(3) $A$ is a *forest* if, for each state $s$ of $A$, there is exactly one execution that leads to $s$. A forest is characterized uniquely by the property that all states of $A$ are reachable, start states have no incoming steps, and each of the other states has exactly one incoming step.

The relation *after*$(A)$ consists of the pairs $(\beta, s)$ for which there is a finite execution of $A$ with trace $\beta$ and last state $s$:

$$\mathit{after}(A) \triangleq \{(\beta, s) \mid \exists \alpha \in \mathit{execs}^*(A) : \mathit{trace}(\alpha) = \beta \text{ and } \mathit{last}(\alpha) = s\}.$$

(Here *last* denotes the function that returns the last element of a finite, nonempty sequence.) We also define *past*$(A)$ to be the inverse of *after*$(A)$, $\mathit{past}(A) \triangleq \mathit{after}(A)^{-1}$; this relates a state $s$ of $A$ to the traces of finite executions of $A$ that lead to $s$.

The following elementary lemma by Lynch and Vaandrager [1995] states that for the restricted kinds of automata defined above, the relations *after* and *past* satisfy certain nice properties.

LEMMA 2.1.

(1) *If $A$ is deterministic then after$(A)$ is a function from traces$^*(A)$ to states$(A)$.*
(2) *If $A$ has fin then after$(A)$ is* image-finite*, i.e., each trace in the domain of after$(A)$ is only related to finitely many states.*
(3) *If $A$ is a forest then past$(A)$ is a function from states$(A)$ to traces$^*(A)$.*

## 3. STEP REFINEMENTS AND EXECUTION CORRESPONDENCE

In this section, we present *step refinements*, the simplest notion of simulation that we consider in this paper. In order to prove soundness of step refinements, we also introduce the auxiliary notion of *execution correspondence*. This notion plays a key role in this paper; the technical lemmas that we prove in this section will also be used repeatedly in subsequent sections.

### 3.1 Step Refinements

Let $A$ and $B$ be automata. A *step refinement* from $A$ to $B$ is a partial function $r$ from $states(A)$ to $states(B)$ that satisfies the following two conditions:

(1) If $s \in start(A)$ then $s \in domain(r)$ and $r(s) \in start(B)$.
(2) If $s \xrightarrow{a}_A t \wedge s \in domain(r)$ then $t \in domain(r)$ and
  (a) $r(s) = r(t) \wedge a = \tau$, or
  (b) $r(s) \xrightarrow{a}_B r(t)$.

Note that, by a trivial inductive argument, the set of states for which $r$ is defined contains all the reachable states of $A$ (and is thus an *invariant* of this automaton). We write $A \leq_{\mathrm{R}} B$ if there exists a step refinement from $A$ to $B$.

As far as we know, the notion of step refinements was first proposed by Nipkow and Slind [1995]. However, if we insist on the presence of stuttering steps $s \xrightarrow{\tau} s$ for each state $s$ (a common assumption in models of reactive systems) then clause (2a) in the above definition becomes superfluous and the notion of a step refinement reduces to that of a homomorphism between reachable subautomata [Ginzburg 1968]. Step refinements are slightly more restrictive than the *possibility mappings* of Lynch and Tuttle [1987] (called *weak refinements* by Lynch and Vaandrager [1995]). In the case of a possibility mapping each (reachable) step of $A$ may be matched by a sequence of steps in $B$ with the same trace. This means that in the above definition condition (2) is replaced by:

2. If $s \xrightarrow{a}_A t \wedge s \in domain(r)$ then $t \in domain(r)$ and $B$ has an execution fragment $\alpha$ with $first(\alpha) = r(s)$, $trace(\alpha) = trace(a)$ and $last(\alpha) = r(t)$.

Observe that, unlike step refinements, possibility mappings do not reduce global reasoning to local reasoning.

*Example* 3.1. Figure 3 illustrates the notion of a step refinement. Note that the $\tau$-steps in $A$ are not matched by any step in $B$. Also the $c$-step in $A$ is not matched by any step in $B$: both source and target states of this step are outside the domain of the step refinement. This is allowed since both states are unreachable. Observe that there is no step refinement from $B$ to $A$, but that there exists a possibility mapping from $B$ to $A$.

Figure 4 gives another example. In this case there is a step refinement from $A'$ to $B'$ but not from $B'$ to $A'$. There is not even a possibility mapping from $B'$ to $A'$.

The following proposition states a basic sanity property of step refinements.

PROPOSITION 3.2. $\leq_{\mathrm{R}}$ *is a preorder (i.e., is transitive and reflexive).*

PROOF. The identity function from $states(A)$ to itself trivially is a step refinement from $A$ to itself. Hence $\leq_{\mathrm{R}}$ is reflexive. Transitivity follows from the observation that if $r$ is a step refinement from $A$ to $B$ and $r'$ is a step refinement from $B$ to $C$, then the function composition $r' \circ r$ is a step refinement from $A$ to $C$. □

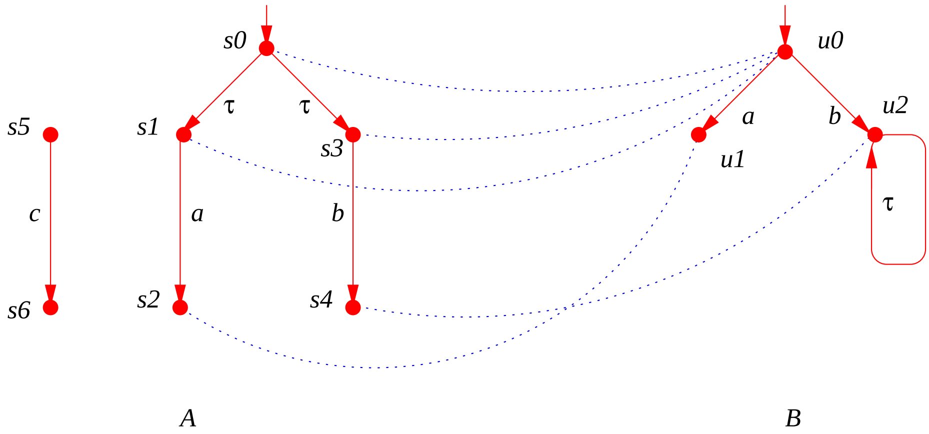


Fig. 3. A step refinement.

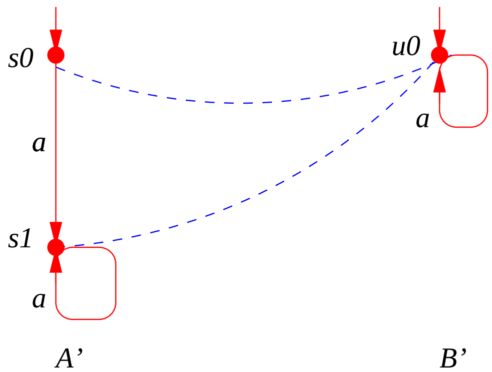


Fig. 4. Another step refinement.

### 3.2 Execution Correspondence

If there exists a step refinement from $A$ to $B$ then we can construct, for each execution fragment of $A$, a corresponding execution fragment of $B$ with the same trace. The notion of 'corresponding' is formalized below.

Suppose $A$ and $B$ are automata, $R \subseteq \mathit{states}(A) \times \mathit{states}(B)$, and $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$ and $\alpha' = u_0 b_1 u_1 b_2 u_2 \cdots$ are execution fragments of $A$ and $B$, respectively. Let $\mathit{index}(\alpha)$ and $\mathit{index}(\alpha')$ denote the index sets of $\alpha$ and $\alpha'$. Then $\alpha$ and $\alpha'$ *correspond via* $R$ and are $R$*-related*, notation $(\alpha, \alpha') \in R$, if there exists an *index relation* over $R$, i.e., a relation $I \subseteq \mathit{index}(\alpha) \times \mathit{index}(\alpha')$ such that (1) if two indices are related by $I$ then the corresponding states are related by $R$; (2) $I$ is monotone; (3) each index of $\alpha$ is related to an index of $\alpha'$ and vice versa; (4) sides of "squares" always have the same label and sides of "triangles" are labeled with $\tau$. Formally we require, for $i, i' \in \mathit{index}(\alpha)$ and $j, j' \in \mathit{index}(\alpha')$,

(1) $(i, j) \in I \quad \Rightarrow \quad (s_i, u_j) \in R$

(2) $(i,j) \in I \wedge (i',j') \in I \wedge i < i' \;\Rightarrow\; j \leq j'$
(3) $I$ and $I^{-1}$ are total
(4) $(i,j) \in I \wedge (i+1, j+1) \in I \Rightarrow a_{i+1} = b_{j+1}$
$(i,j) \in I \wedge (i+1, j) \in I \Rightarrow a_{i+1} = \tau$
$(i,j) \in I \wedge (i, j+1) \in I \Rightarrow b_{j+1} = \tau$

We write $(A, B) \in R$ if for every execution $\alpha$ of $A$ there is an execution $\alpha'$ of $B$ such that $(\alpha, \alpha') \in R$, and $[A, B] \in R$ if for every finite execution $\alpha$ of $A$ there is a finite execution $\alpha'$ of $B$ with $(\alpha, \alpha') \in R$. Figure 5 illustrates the correspondence between two executions of automata $A$ and $B$ from Figure 3.

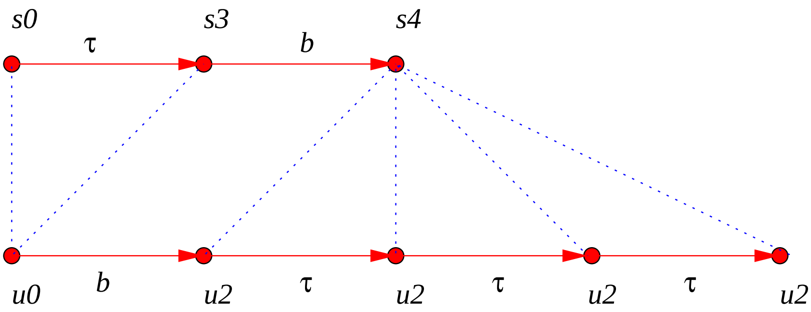


Fig. 5. Execution correspondence.

Another notion of correspondence has been presented by Sogaard-Andersen, Lynch et al. [1993; 1993] and formalized by Mueller [1998]. Within the theory of I/O automata, execution correspondence plays a crucial role in proofs of preservation of both safety and liveness properties. Our notion is more restrictive than earlier work [Gawlick et al. 1993; Søgaard-Andersen et al. 1993], but technically simpler. Moreover it has the advantage that it preserves 'until' properties. In this paper, we only study safety properties and it suffices to know that corresponding executions have the same trace. The latter fact is established in the next lemma.

Lemma 3.3. *(Corresponding execution fragments have the same trace)*

(*1*) *Suppose $I$ is an index relation as above and $(i,j) \in I$. Then*

$$trace(s_0 a_1 s_1 \cdots a_i s_i) = trace(u_0 b_1 u_1 \cdots b_j u_j).$$

(*2*) *If $(\alpha, \alpha') \in R$ then $trace(\alpha) = trace(\alpha')$.*

Proof. For (1), suppose $(i,j) \in I$. By induction on $i + j$ we prove

$$trace(s_0 a_1 s_1 \cdots a_i s_i) \;=\; trace(u_0 b_1 u_1 \cdots b_j u_j).$$

If $i + j = 0$ then both $i$ and $j$ are 0. Clearly, $trace(s_0) = trace(u_0) = \lambda$.

For the induction step, suppose $i + j > 0$. For reasons of symmetry we may assume, without loss of generality, that $i > 0$. Let $j'$ be the largest index with $j' \leq j$ and $(i-1, j') \in I$. (By monotonicity, $i-1$ can only be related to indices less than or equal to $j$, and by totality there is at least one such an index.) We distinguish between three cases:

(1) $j' = j$. Then by condition (4b), $a_i = \tau$. By induction hypothesis,

$$\mathit{trace}(s_0 a_1 s_1 \cdots a_{i-1} s_{i-1}) = \mathit{trace}(u_0 b_1 u_1 \cdots b_j u_j).$$

Hence $\mathit{trace}(s_0 a_1 s_1 \cdots a_i s_i) = \mathit{trace}(u_0 b_1 u_1 \cdots b_j u_j)$.

(2) $j' = j - 1$. Then by condition (4a), $a_i = b_j$. By induction hypothesis,

$$\mathit{trace}(s_0 a_1 s_1 \cdots a_{i-1} s_{i-1}) = \mathit{trace}(u_0 b_1 u_1 \cdots b_{j-1} u_{j-1}).$$

Hence $\mathit{trace}(s_0 a_1 s_1 \cdots a_i s_i) = \mathit{trace}(u_0 b_1 u_1 \cdots b_j u_j)$.

(3) $j' < j - 1$. Then by conditions (2) and (3), $(i, j-1) \in I$. By condition (4c), this implies $b_j = \tau$. By induction hypothesis,

$$\mathit{trace}(s_0 a_1 s_1 \cdots a_i s_i) = \mathit{trace}(u_0 b_1 u_1 \cdots b_{j-1} u_{j-1}).$$

Hence $\mathit{trace}(s_0 a_1 s_1 \cdots a_i s_i) = \mathit{trace}(u_0 b_1 u_1 \cdots b_j u_j)$.

This completes the proof of the induction step.

For (2), suppose that $(\alpha, \alpha') \in R$. Then there exists an index relation $I$ that relates $\alpha$ and $\alpha'$. Using (1) and the fact that both $I$ and $I^{-1}$ are total, it follows that each finite prefix of $\mathit{trace}(\alpha)$ is also a finite prefix of $\mathit{trace}(\alpha')$, and vice versa. This implies $\mathit{trace}(\alpha) = \mathit{trace}(\alpha')$. $\square$

The next corollary will be used repeatedly in the rest of this paper. It states that in order to prove trace inclusion between automata $A$ and $B$ it suffices to find for each execution of $A$ a corresponding execution of $B$. Depending on whether one wants to prove inclusion of all traces or of finite traces only, a stronger respectively weaker type of execution correspondence is required.

COROLLARY 3.4. *(Execution correspondence implies trace inclusion)*

*(1) If $(A, B) \in R$ then $[A, B] \in R$.*
*(2) If $[A, B] \in R$ then $A \leq_{*\mathrm{T}} B$.*
*(3) If $(A, B) \in R$ then $A \leq_{\mathrm{T}} B$.*

PROOF. Statement (1) follows from the definitions. Statements (2) and (3) follow immediately from Lemma 3.3 and the definitions. $\square$

### 3.3 Soundness and Partial Completeness

The next theorem states that if there is a step refinement from $A$ to $B$, it is possible to construct, for each execution of $A$, a corresponding execution of $B$. Using Corollary 3.4, this implies that step refinements constitute a sound technique for proving trace inclusion. In addition, the next theorem also allows us to use step refinements as a sound technique for proving implementation relations between live automata, as in previous work [Gawlick et al. 1993; Søgaard-Andersen et al. 1993; Mueller 1998].

THEOREM 3.5. *(Soundness of step refinements)*
*If $r$ is a step refinement from $A$ to $B$ then $(A, B) \in r$.*

PROOF. Suppose $r$ is a step refinement from $A$ to $B$. Let $\alpha = s_0 a_1 s_1 \cdots$ be an execution of $A$. Inductively, we define an execution $\alpha' = u_0 b_1 u_1 \cdots$ of $B$ and an index relation $I$ such that $\alpha$ and $\alpha'$ are $r$-related via $I$.

To start with, define $u_0 = r(s_0)$ and declare $(0, 0)$ to be an element of $I$.

Now suppose $(i, j) \in I$ and $i$ is a nonfinal index of $\alpha$. We distinguish between two cases:

(1) If $r(s_i) \xrightarrow{a_{i+1}}_B r(s_{i+1})$ then define $b_{j+1} = a_{i+1}$, $u_{j+1} = r(s_{i+1})$, and declare $(i + 1, j + 1)$ to be an element of $I$;

(2) otherwise, declare $(i + 1, j)$ to be an element of $I$.

By construction, using the defining properties of a step refinement, it follows that $I$ is an index relation. This implies $(A, B) \in r$. $\square$

Step refinements alone do not provide a complete method for proving trace inclusion. There is a partial completeness result, however.

Theorem 3.6. *(Partial completeness of step refinements)*
*Suppose $A$ is a forest, $B$ is deterministic and $A \leq_{*\mathrm{T}} B$. Then $A \leq_{\mathrm{R}} B$.*

Proof. The relation $r \triangleq \mathit{after}(B) \circ \mathit{past}(A)$ is a step refinement from $A$ to $B$. $\square$

Actually, we can even slightly strengthen the above theorem. It suffices to assume that $A$ restricted to its reachable states is a forest, and that $B$ restricted to its reachable states is deterministic. In Figure 3, automaton $A$ restricted to its reachable states is a forest and automaton $B$ is deterministic. As we observed already, there is a step refinement from $A$ to $B$. Even if we restrict to reachable states, automaton $B$ is not a forest and automaton $A$ is not deterministic. As we observed, there is no step refinement from $B$ to $A$.

In practice, the preconditions of Theorem 3.6 are seldom met. The higher-level specification often is deterministic, but it rarely occurs that the lower-level specification is a forest. Nevertheless, step refinements have been used in several substantial case studies [Helmink et al. 1994; Nipkow and Slind 1995; Devillers et al. 2000].

## 4. NORMED FORWARD SIMULATIONS

Even though there exists no step refinement from automaton $B'$ to automaton $A'$ in Figure 4, these automata do have the same traces. By moving from functions to relations it becomes possible to prove that each trace of $B'$ is also a trace of $A'$. This idea is formalized in the following definition.

A *normed forward simulation* from $A$ to $B$ consists of a relation $f \subseteq \mathit{states}(A) \times \mathit{states}(B)$ and a function $n : \mathit{steps}(A) \times \mathit{states}(B) \to S$, for some well-founded set $S$, such that (here $f[s]$ denotes the set $\{u \mid (s, u) \in f\}$):

(1) If $s \in \mathit{start}(A)$ then $f[s] \cap \mathit{start}(B) \neq \emptyset$.

(2) If $s \xrightarrow{a}_A t \wedge u \in f[s]$ then
   (a) $u \in f[t] \wedge a = \tau$, or
   (b) $\exists v \in f[t] : u \xrightarrow{a}_B v$, or
   (c) $\exists v \in f[s] : u \xrightarrow{\tau}_B v \wedge n(s \xrightarrow{a} t, v) < n(s \xrightarrow{a} t, u)$.

Write $A \leq_{\mathrm{F}} B$ if there exists a normed forward simulation from $A$ to $B$.

The intuition behind this definition is that if $s \xrightarrow{a}_A t$ and $(s, u) \in f$, then either (a) the transition in $A$ is a stuttering step that does not have to be matched, or (b) there is a matching step in $B$, or (c) $B$ can do a stuttering step which decreases the norm. Since the norm decreases at each application of clause (c), this clause can only be applied a finite number of times. In general, the norm function may depend both on the transitions in $A$ and on the states of $B$. However, if $B$ is *convergent*, i.e., there are no infinite $\tau$-paths, then one can simplify the type of the norm function (though not necessarily the definition of the norm function itself) to $n : \mathit{states}(B) \to S$. In fact, in the approach of Groote and Springintveld [1995], which not always applies to divergent processes, the norm function is required to be of this restricted type.

*Example* 4.1. In Figure 4, the relation indicated by the dashed lines, together with an arbitrary norm function, is a normed forward simulation from $B'$ to $A'$.

Consider automata $A$ and $B$ in Figure 3. Let $n$ be the function that assigns norm 1 to state $s0$ and norm 0 to all other states of $A$. Then $n$ together with the relation indicated by the dashed lines constitutes a normed forward simulation from $B$ to $A$.

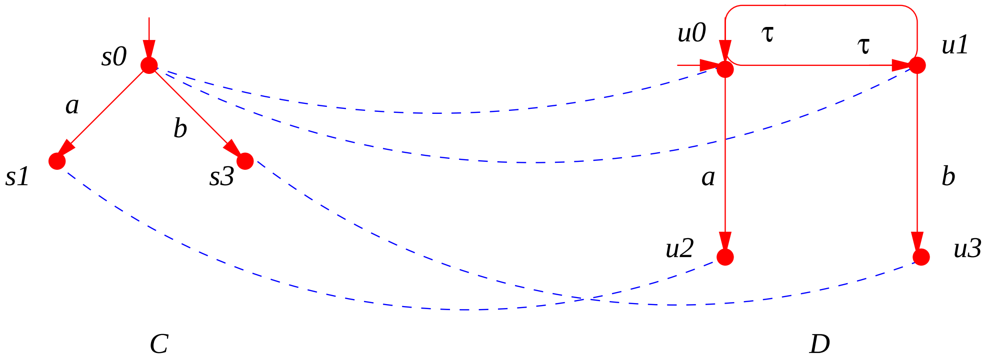


Fig. 6. Norm function must take steps of $C$ into account.

Now consider the automata $C$ and $D$ in Figure 6. Let $m$ be a norm function satisfying

$$\begin{array}{rclrcl} m(s0 \xrightarrow{a} s1, u0) & = & 0 & m(s0 \xrightarrow{a} s1, u1) & = & 1 \\ m(s0 \xrightarrow{b} s3, u0) & = & 1 & m(s0 \xrightarrow{b} s3, u1) & = & 0 \end{array}$$

Then $m$ together with the relation indicated by the dashed lines constitutes a normed forward simulation from $C$ to $D$. It is not hard to see that in this example, where $D$ is not convergent, the norm necessarily depends on the selected step in $C$.

The example of Figure 6 also serves to illustrate the difference between normed forward simulations and the forward simulations that were studied by Jonsson [1990; 1991; 1994]. Essentially, Jonsson's forward simulations are just normed forward simulations, except that there is no norm function and condition 2(c) has been omitted. We leave it to the reader to check that there exists no forward simulation in this sense from $C$ to $D$. This is the case even when we add "stuttering" $\tau$-loops to each state, as required in Jonsson's models.

The next proposition asserts that normed forward simulations indeed generalize step refinements.

PROPOSITION 4.2. $A \leq_{\mathrm{R}} B \Rightarrow A \leq_{\mathrm{F}} B$.

PROOF. Together with an arbitrary norm function, any step refinement (viewed as a relation) is a normed forward simulation. □

The soundness of normed forward simulations is trivially implied by the following lemma and Corollary 3.4.

LEMMA 4.3. *Suppose $(f, n)$ is a normed forward simulation from $A$ to $B$, $A$ has an execution fragment $\alpha$ with first state $s$, and $u$ is a state of $B$ with $u \in f[s]$. Then $B$ has an execution fragment $\alpha'$ that starts in $u$ such that $(\alpha, \alpha') \in f$.*

PROOF. Let $c : \mathit{steps}(A) \times \mathit{states}(B) \to \{L, C, R\} \times \mathit{states}(B)$ be a function such that $c(s \xrightarrow{a} t, u) = (x, v)$ and $u \in f[s]$ implies

(1) If $x = L$ then $u \in f[t] \wedge a = \tau$.
(2) If $x = C$ then $v \in f[t] \wedge u \xrightarrow{a}_B v$.
(3) If $x = R$ then $v \in f[s] \wedge u \xrightarrow{\tau}_B v \wedge n(s \xrightarrow{a} t, v) < n(s \xrightarrow{a} t, u)$.

The existence of $c$, which chooses between a left move (L) of $A$, a common move (C) of $A$ and $B$, or a right move (R) of $B$, is guaranteed by the fact that $(f, n)$ is a normed forward simulation.

Let $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$. Then $s = s_0$. Inductively, we define a sequence $\sigma = z_0 z_1 z_2 \cdots$ of 4-tuples in $\mathsf{N} \times \mathsf{N} \times \mathit{acts}(B) \times \mathit{states}(B)$. The first element in the sequence is $z_0 = (0, 0, \tau, u)$. If $z_k = (i, j, b, u)$ is an element of the sequence, and $i$ is a nonfinal index of $\alpha$, then we define $z_{k+1}$ as follows

(1) If $c(s_i \xrightarrow{a_{i+1}} s_{i+1}, u) = (L, v)$ then $z_{k+1} = (i + 1, j, b, u)$.
(2) If $c(s_i \xrightarrow{a_{i+1}} s_{i+1}, u) = (C, v)$ then $z_{k+1} = (i + 1, j + 1, a_{i+1}, v)$.
(3) If $c(s_i \xrightarrow{a_{i+1}} s_{i+1}, u) = (R, v)$ then $z_{k+1} = (i, j + 1, \tau, v)$.

Suppose that both $(i, j, b, u)$ and $(i', j, b', u')$ occur in sequence $\sigma$. We claim that $b = b'$ and $u = u'$. To see why this is true assume without loss of generality that $(i, j, b, u)$ occurs before $(i', j, b', u')$. Now observe that the values of both the first and second component of elements in $\sigma$ increase monotonically. This means that each successor of $(i, j, b, u)$ up to and including $(i', j, b', u')$ has been obtained from its predecessor by applying rule (1). This implies that the the second respectively third components of all elements in the sequence from $(i, j, b, u)$ until $(i', j, b', u')$ coincide. Hence $b = b'$ and $u = u'$.

Using this property, we can define for each element $(i, j, b, u)$ in $\sigma$, $b_j = b$ and $u_j = u$. Let $\alpha' = u_0 b_1 u_1 b_2 u_2 \cdots$ and let $I = \{(i, j) \mid \exists b, u : (i, j, b, u) \text{ occurs in } \sigma\}$. By construction of $\sigma$, using the properties of $c$, it follows that $\alpha'$ is an execution fragment of $B$ that starts in $u$, and that $I$ is an index relation over $f$. This implies $(\alpha, \alpha') \in f$. □

THEOREM 4.4. *(Soundness of normed forward simulations)*
*If $f$ is a normed forward simulation from $A$ to $B$ then $(A, B) \in f$.*

PROOF. Immediate from the definitions and Lemma 4.3. □

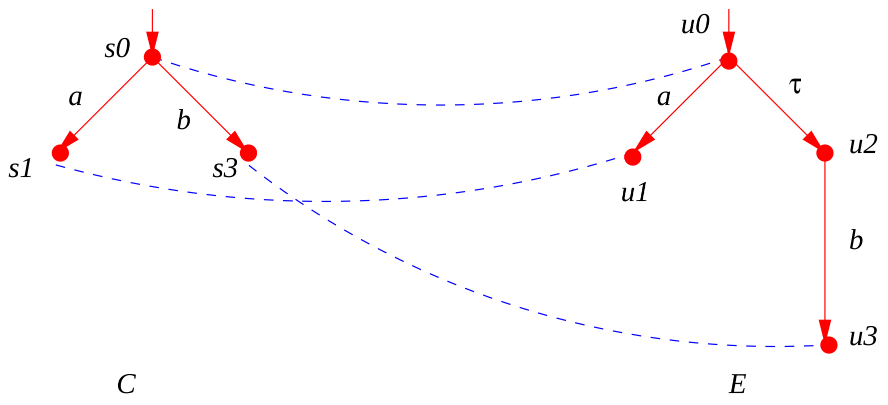


Fig. 7. Difference between forward simulations and normed forward simulations.

*Example* 4.5. Consider automata $C$ and $E$ in Figure 7. There does not exist a normed forward simulation from $C$ to $E$. Such a simulation would have to relate states $s0$ and $u0$. But in order for $E$ to simulate the step $s0 \xrightarrow{b} s3$, it would also have to relates states $s0$ and $u2$. But this is impossible since from state $u2$ there is no way to simulate the step $s0 \xrightarrow{a} s1$.

It turns out that there does exist a *forward simulation* in Lynch and Vaandrager's sense [1995] from $C$ to $E$. In the case of a forward simulation, a step of $A$ may be matched by a sequence of steps in $B$ with the same trace. This means that in the definition of a normed forward simulation condition (2) is replaced by:

2. If $s \xrightarrow{a}_A t \wedge u \in f[s]$ then $B$ has an execution fragment $\alpha$ with $\mathit{first}(\alpha) = u$, $\mathit{trace}(\alpha) = \mathit{trace}(a)$ and $\mathit{last}(\alpha) \in f[t]$.

The dashed lines in Figure 7 indicate a forward simulation from $C$ to $E$.

The automata $A$ and $B$ in Figure 3 provide us with a similar example: there exists a forward simulation from $B$ to $A$, but no normed forward simulation.

The difference between forward simulations and normed forward simulations is very similar to the difference between Milner's *observation equivalence* [1989] and the *branching bisimulation* of Van Glabbeek and Weijland [1996]. In fact, we can characterize normed forward simulations in terms of "branching forward simulations", a notion that is inspired by the branching bisimulations of [Glabbeek and Weijland 1996]. A similar characterization has been obtained by Namjoshi [1997] in the setting of stuttering bisimulations.

Formally, a *branching forward simulation* from $A$ to $B$ is a relation $f \subseteq \mathit{states}(A) \times \mathit{states}(B)$ such that

(1) If $s \in \mathit{start}(A)$ then $f[s] \cap \mathit{start}(B) \neq \emptyset$.

(2) If $s \xrightarrow{a}_A t$ and $u \in f[s]$ then $B$ has an execution fragment that starts in $u$ and that is $f$-related to $s \xrightarrow{a} t$.

The following theorem implies that there exists a normed forward simulation between two automata if and only if there is a branching forward simulation between them.

Theorem 4.6.

(*1*) *Suppose $(f, n)$ is a normed forward simulation from $A$ to $B$. Then $f$ is a branching forward simulation from $A$ to $B$.*

(*2*) *Suppose $f$ is a branching forward simulation from $A$ to $B$. Let $n(s \xrightarrow{a} t, u)$ be 0 if $u \notin f[s]$ and otherwise be equal to the length of the shortest execution fragment that starts in $u$ and that is $f$-related to $s \xrightarrow{a} t$. Then $(f, n)$ is a normed forward simulation from $A$ to $B$.*

Proof. Part (1) follows by Lemma 4.3. The proof of part (2) is routine. □

An interesting corollary of Theorem 4.6 is that if there exists a normed forward simulation between two automata, there is in fact a normed forward simulation with a norm that has the natural numbers as its range.

The proof that branching bisimilarity is an equivalence is known to be tricky [Basten 1996]. Likewise, the proof that branching forward simulations induce a preorder is nontrivial. We first need to define the auxiliary concept of a *reduced* index relation and to prove a lemma about it.

Suppose that $\alpha$ and $\alpha'$ are $R$-related via index relation $I$. We say that $I$ is *reduced* if the following two conditions are satisfied:

(1) If $\alpha$ is finite then $I$ relates the final index of $\alpha$ only to the final index of $\alpha'$.

(2) $I$ is *N-free*: $(i, j) \in I \wedge (i+1, j+1) \in I \quad \Rightarrow \quad (i+1, j) \notin I \wedge (i, j+1) \notin I$.

Observe that if $\alpha$ is finite and $I$ is reduced, then $\alpha'$ is also finite. The following technical lemma states that index relations can always be reduced.

Lemma 4.7. *Suppose that $\alpha$ and $\alpha'$ are $R$-related via index relation $I$. Then $\alpha'$ has a prefix $\alpha''$ that is $R$-related to $\alpha$ via a reduced index relation $J \subseteq I$.*

Proof. If $\alpha$ is infinite then let $\alpha'' = \alpha'$. If $\alpha$ is finite then let $\alpha''$ be the finite prefix of $\alpha'$ up to and including the first state whose index is related by $I$ to the final index of $\alpha$.

Inductively we define a sequence $\sigma = z_0 z_1 z_2 \cdots$ of pairs in $\mathsf{N} \times \mathsf{N}$. The first element of the sequence is $z_0 = (0, 0)$. If $z_k = (i, j)$ is an element of the sequence and $i$ is a nonfinal index then we define $z_{k+1}$ as follows:

(1) $(i+1, j+1) \in I \quad \Rightarrow \quad z_{k+1} = (i+1, j+1)$

(2) $(i+1, j) \in I \wedge (i+1, j+1) \notin I \quad \Rightarrow \quad z_{k+1} = (i+1, j)$

(3) $(i, j+1) \in I \wedge (i+1, j+1) \notin I \quad \Rightarrow \quad z_{k+1} = (i, j+1)$

Note that since $I$ is an index relation, $z_{k+1}$ is properly defined. Let $J = \{(i, j) \mid (i, j) \text{ occurs in } \sigma\}$. It is routine to check that $J \subseteq I$, that $\alpha$ and $\alpha''$ are $R$-related via $J$, and that $J$ is reduced. A tricky point is the totality of $J$ and $J^{-1}$. We prove that $J$ is total by contradiction. Suppose that $J$ is not total. Let $i$ be the smallest index of $\alpha$ with $J[i] = \emptyset$. Let $j$ be the smallest index of $\alpha'$ with $(i, j) \in I$ ($j$ exists since index relation $I$ is total). Let $l$ be the maximal index of $\alpha'$ with $(i-1, l) \in J$ (there is a maximal index since $(i-1, l) \in J$ implies $(i-1, l) \in I$, which implies $l \leq j$ by monotonicity of index relation $I$). Let $z_k = (i-1, l)$. Since $J[i] = \emptyset$,

$z_{k+1} = (i-1, l+1)$. Hence $(i-1, l+1) \in J$. But this contradicts the fact that $l$ be the maximal index of $\alpha'$ with $(i-1, l) \in J$.

In a similar way also the totality of $J^{-1}$ and N-freeness can be proved by contradiction. □

We are now prepared to prove that branching forward simulations (and hence also normed forward simulations) induce a preorder.

Proposition 4.8. *$\leq_{\mathrm{F}}$ is a preorder.*

Proof. For reflexivity, observe that the identity function from *states*$(A)$ to itself is a branching forward simulation from $A$ to itself.

For transitivity, suppose $f$ and $g$ are branching forward simulations from $A$ to $B$ and from $B$ to $C$, respectively. We claim that $g \circ f$ is a branching forward simulation from $A$ to $C$. It is trivial to check that $g \circ f$ satisfies condition (1) in the definition of a branching forward simulation. For condition (2), suppose that $s \xrightarrow{a}_A t \wedge u \in (g \circ f)[s]$. Then there exists a state $w$ of $B$ such that $w \in f[s]$ and $u \in g[w]$. Hence there is an execution fragment $\alpha$ starting in $w$ such that $s \xrightarrow{a} t$ and $\alpha$ are $f$-related via some index relation $I$. By Lemma 4.7, we may assume that $I$ is reduced. Also, there is an execution fragment $\alpha'$ starting in $u$ such that $\alpha$ and $\alpha'$ are $g$-related via some index relation $J$. Again by Lemma 4.7, we may assume that $J$ is reduced. Using the fact that both $I$ and $J$ are reduced, it is routine to check that $s \xrightarrow{a} t$ and $\alpha'$ are $g \circ f$-related via index relation $J \circ I$. Thus $g \circ f$ satisfies condition (2) in the definition of a branching forward simulation. □

Variants of the partial completeness result below appear in several papers [Jonsson 1987; Lynch and Vaandrager 1995]. Since higher-level specifications are often deterministic, this result explains why in practice (normed) forward simulations can so often be used to prove behavior inclusion.

Theorem 4.9. *(Partial completeness of normed/branching forward simulations) If $B$ is deterministic and $A \leq_{*\mathrm{T}} B$ then $A \leq_{\mathrm{F}} B$.*

Proof. The relation $f \triangleq \mathit{after}(B) \circ \mathit{past}(A)$ is a branching forward simulation from $A$ to $B$. □

It is interesting to note that there is one earlier result [Lynch and Vaandrager 1995] concerning forward simulations that does not carry over to the normed (branching) simulations of this paper. This result, Proposition 3.12, states that if $A$ is a forest and $A \leq_{\mathrm{F}} B$ then $A \leq_{\mathrm{R}} B$. The automata $C$ and $D$ of Figure 6 constitute a counterexample. Actually, the same Proposition 3.12 also does not carry over to the setting of timed automata used earlier [Lynch and Vaandrager 1996].

## 5. NORMED BACKWARD SIMULATIONS

As we observed, there exists no normed forward simulation from automaton $B$ to automaton $A$ in Figure 3, even though both automata have the same traces. Also, there does not exist a normed forward simulation from automaton $C$ to the trace equivalent automaton $E$ in Figure 7. In both cases a forward simulation in Lynch and Vaandrager's sense [1995] exists. However, the example in Figure 8 below shows that also forward simulations do not yet provide us with a complete method for proving trace inclusion. It is well-known from the literature that completeness can be obtained by adding some form of *backward simulation*.

*Example* 5.1. There exists no (normed/branching) forward simulation from automaton $C$ to automaton $F$ in Figure 8. The relation indicated by the dashed lines fails since from state $u0$ the $b$-step from $s0$ can not be simulated, whereas from $u2$ the $a$-step from $s0$ can not be simulated.

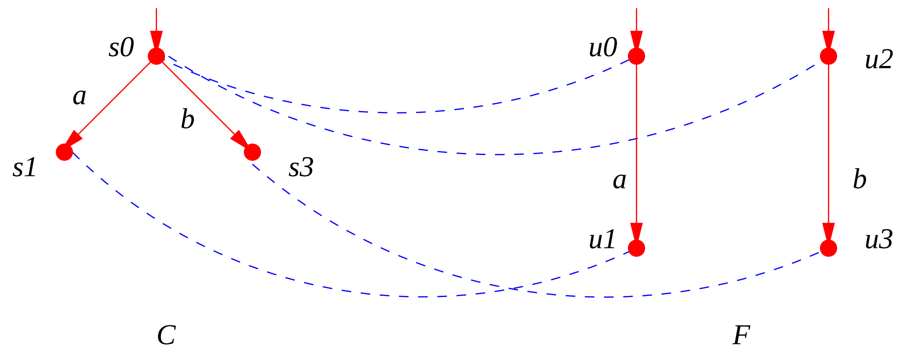


Fig. 8. The need for backward simulations.

In many respects, backward simulations are the dual of forward simulations. Whereas a forward simulation requires that *some* state in the image of each start state should be a start state, a backward simulation requires that *all* states in the image of a start state be start states. Also, a forward simulation requires that *forward* steps in the source automaton can be simulated from related states in the target automaton, whereas the corresponding condition for a backward simulations requires that *backward* steps can be simulated. However, the two notions are not completely dual: the definition of a backward simulation contains a nonemptiness condition, and also, in order to obtain soundness for general trace inclusion, backward simulations also require a finite image condition. The mismatch is due to the asymmetry in our automata between the future and the past: from any given state, all the possible histories are finite executions, whereas the possible futures can be infinite.

Formally, we define a *normed backward simulation* from $A$ to $B$ to be a pair of a total relation $b \subseteq \mathit{states}(A) \times \mathit{states}(B)$ and a function $n : (\mathit{steps}(A) \cup \mathit{start}(A)) \times \mathit{states}(B) \to S$, for some well-founded set $S$, satisfying

(1) If $s \in \mathit{start}(A) \ \wedge \ u \in b[s]$ then
  (a) $u \in \mathit{start}(B)$, or
  (b) $\exists v \in b[s] : v \xrightarrow{\tau}_B u \ \wedge \ n(s, v) < n(s, u)$.

(2) If $t \xrightarrow{a}_A s \ \wedge \ u \in b[s]$ then
  (a) $u \in b[t] \ \wedge \ a = \tau$, or
  (b) $\exists v \in b[t] : v \xrightarrow{a}_B u$, or
  (c) $\exists v \in b[s] : v \xrightarrow{\tau}_B u \ \wedge \ n(t \xrightarrow{a} s, v) < n(t \xrightarrow{a} s, u)$.

Write $A \leq_{\mathrm{B}} B$ if there is a normed backward simulation from $A$ to $B$, and $A \leq_{\mathrm{iB}} B$ if there is a normed backward simulation from $A$ to $B$ that is image-finite.

*Example* 5.2. In Figure 8, the relation indicated by the dashed lines is a normed backward simulation from $C$ to $E$, for arbitrary norm functions. It is not difficult to construct normed backward simulations from automaton $B$ to automaton $A$ in Figure 3, and from automaton $C$ to automaton $E$ in Figure 7.

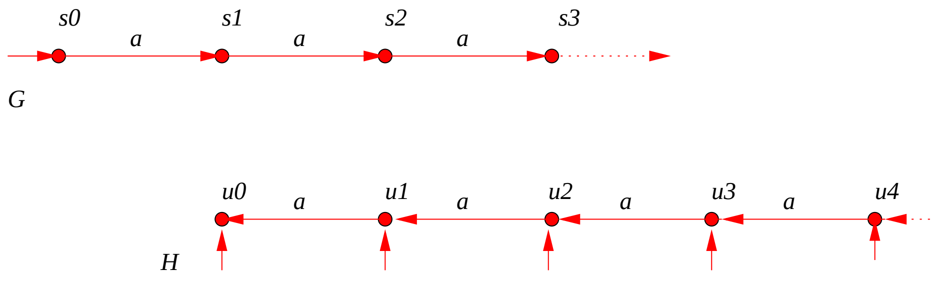


Fig. 9. No image-finite normed backward simulation.

Figure 9 illustrates the difference between $\leq_{\mathrm{B}}$ and $\leq_{\mathrm{iB}}$. Relation $states(G) \times states(H)$ together with an arbitrary norm function constitutes a normed backward simulation from $G$ to $H$. We claim that no image-finite normed backward simulation exist. Because suppose that $b$ is such a relation. Then, for all $i, j \in \mathsf{N}$ with $i > 0$,

$$(si, uj) \in b \;\Rightarrow\; (si-1, uj+1) \in b$$

This implies that

$$(si, uj) \in b \;\Rightarrow\; (s0, ui+j) \in b$$

Since each state $si$ is related to at least one state $sj$, it follows that state $s0$ is related to infinitely many states, which is a contradiction.

The following proposition states some trivial connections between the preorders induced by normed backward simulations and step refinements.

PROPOSITION 5.3.

(*1*) *If all states of $A$ are reachable and $A \leq_{\mathrm{R}} B$ then $A \leq_{\mathrm{iB}} B$.*
(*2*) *If $A \leq_{\mathrm{iB}} B$ then $A \leq_{\mathrm{B}} B$.*

PROOF. Trivial. □

The next lemma is required to prove soundness of normed backward simulations.

LEMMA 5.4. *Suppose $(b, n)$ is a normed backward simulation from $A$ to $B$, $A$ has a finite execution fragment $\alpha$ with last state $s$, and $u$ is a state of $B$ with $u \in b[s]$. Then $B$ has a finite execution fragment $\alpha'$ that ends in $u$ such that $(\alpha, \alpha') \in b$. Moreover, if $\alpha$ is an execution then $\alpha'$ can be chosen to be an execution as well.*

PROOF. Similar to the proof of Lemma 4.3. □

By Lemma 5.4 and Corollary 3.4, the existence of a normed backward simulation implies inclusion of finite traces. Normed backward simulations, however, are in general not a sound method for proving inclusion of infinite traces. As a counterexample, consider automata $G$ and $H$ from Figure 9. There exists a normed backward simulation from $G$ to $H$, but the infinite trace $a^\omega$ of $G$ is not a trace of $H$. As is well-known from the literature, a sound method for proving inclusion of infinite traces can be obtained by requiring image finiteness of the simulation relation.

Theorem 5.5. *(Soundness of normed backward simulations)*

(1) *If $b$ is a normed backward simulation from $A$ to $B$ then $[A,B] \in b$.*

(2) *If moreover $b$ is image-finite then $(A,B) \in b$.*

Proof. Statement (1) follows immediately by Lemma 5.4 and the totality of $b$. In order to prove (2), suppose that $b$ is image-finite. Let $\alpha$ be an execution of $A$. We have to establish the existence of an execution $\alpha'$ of $B$ with $(\alpha, \alpha') \in b$. If $\alpha$ is finite then this follows by Lemma 5.4 and the totality of $b$. So assume that $\alpha$ is infinite. We use a minor variation of Kőnig's Lemma [Knuth 1997] presented by Lynch and Vaandrager [1995]:

> *Let $G$ be an infinite digraph such that (1) $G$ has finitely many roots, i.e., nodes without incoming edges, (2) each node of $G$ has finite outdegree, and (3) each node of $G$ is reachable from some root. Then there is an infinite path in $G$ starting from some root.*

The nodes of the graph $G$ that we consider are pairs $(I, \gamma)$ where $\gamma$ is a finite execution of $B$ and $I$ is an index relation that relates $\gamma$ to some finite prefix of $\alpha$. There is an edge from a node $(I, \gamma)$ to a node $(I', \gamma')$ iff $\gamma$ is a prefix of $\gamma'$ and $I'$ extends $I$ with precisely one element. It is straightforward to check that $G$ satisfies the conditions of Kőnig's Lemma. Hence $G$ has an infinite path. Let $J$ be the union of all the index relations occurring on nodes in this path, and let $\alpha'$ be the limit of the finite executions of the nodes in this path. Observe that, by image-finiteness of $b$, each index of $\alpha$ occurs in the domain of $J$. Hence $(\alpha, \alpha') \in b$. $\square$

The following Proposition 5.6 is in a sense the converse of Proposition 5.3. The proof is similar to that of the corresponding result by Lynch and Vaandrager [1995].

Proposition 5.6.

(1) *If $B$ is deterministic and $A \leq_{\mathrm{B}} B$ then $A \leq_{\mathrm{R}} B$.*

(2) *If all states of $A$ are reachable, $B$ has fin and $A \leq_{\mathrm{B}} B$, then $A \leq_{\mathrm{iB}} B$.*

Proof. For (1), suppose that $B$ is deterministic and that $b$ is a normed backward simulation from $A$ to $B$. Suppose that $s$ is a reachable state of $A$. We will prove that $b[s]$ contains exactly one element. Since any normed backward simulation that is functional on the reachable states trivially induces a step refinement, this gives us $A \leq_{\mathrm{R}} B$.

Because $b$ is a normed backward simulation it is a total relation, so we know $b[s]$ contains at least one element. Suppose that both $u_1 \in b[s]$ and $u_2 \in b[s]$; we prove $u_1 = u_2$. Since $s$ is reachable, $A$ has an execution $\alpha$ that ends in $s$. By Lemma 5.4, $B$ has executions $\alpha_1$ and $\alpha_2$ which end in $u_1$ and $u_2$, respectively, such that $(\alpha, \alpha_1) \in b$ and $(\alpha, \alpha_2) \in b$. By Lemma 3.3, $trace(\alpha) = trace(\alpha_1) = trace(\alpha_2)$. Now $u_1 = u_2$ follows by Lemma 2.1(1), using the fact the $B$ is deterministic.

For (2), suppose that all states of $A$ are reachable, $B$ has fin, and $b$ is a normed backward simulation from $A$ to $B$. Suppose that $s$ is a state of $A$. Since $s$ is reachable, there is an execution $\alpha$ that ends in $s$. Let $\beta$ be trace of $\alpha$. By Lemma 5.4 there exists, for each $u \in b[s]$, an execution $\alpha_u$ of $B$ that ends in $u$ such that $(\alpha, \alpha_u) \in b$. By Lemma 3.3, $trace(\alpha_u) = \beta$. Hence $b[s] \subseteq after(B)[\beta]$. But since $B$ has fin, $after(B)[\beta]$ is finite by Lemma 2.1(2). Hence $b$ is image-finite. $\square$

*Example* 5.7. Consider the two automata in Figure 10. It is easy to see that


Fig. 10. Difference between backward simulations and normed backward simulations.

there does not exist a normed backward simulation from the first to the second automaton. However, there does exist a *backward simulation* in Lynch and Vaandrager's sense [1995]. In such a backward simulation, a step of one automaton may be matched by a sequence of steps in the other automaton with the same trace.

As in the forward case, we will now characterize normed backward simulations in terms of "branching backward simulations", and use this characterization to establish that $\leq_{\mathrm{B}}$ and $\leq_{\mathrm{iB}}$ are preorders.

A *branching backward simulation* from $A$ to $B$ is a total relation $b \subseteq states(A) \times states(B)$ such that

(1) If $s \in start(A)$ and $u \in b[s]$ then $B$ has an execution that ends in $u$ and is $b$-related to $s$.

(2) If $t \xrightarrow{a}_A s$ and $u \in f[s]$ then $B$ has an execution fragment that ends in $u$ and is $b$-related to $t \xrightarrow{a} s$.

THEOREM 5.8.

*(1) Suppose $(b, n)$ is a normed backward simulation from $A$ to $B$. Then $b$ is a branching backward simulation from $A$ to $B$.*

*(2) Suppose $b$ is a branching backward simulation from $A$ to $B$. Let $n(s, u)$ be 0 if $s$ is not a start state or $u \notin b[s]$ and otherwise be equal to the length of the shortest execution that ends in $u$ and is $b$-related to $s$. Furthermore, let*

*$n(t \xrightarrow{a} s, u)$ be 0 if $u \notin f[s]$ and otherwise equal to the length of the shortest execution fragment ending in $u$ that is $b$-related to $t \xrightarrow{a}_A s$. Then $(b, n)$ is a normed forward simulation from $A$ to $B$.*

PROOF. Statement (1) follows by Lemma 5.4. The proof of statement (2) is routine. □

As in the forward case, we see that if there exists a normed backward simulation between two automata, there is in fact a normed backward simulation with a norm that has the natural numbers as its range.

PROPOSITION 5.9. *$\leq_{\mathrm{B}}$ and $\leq_{\mathrm{iB}}$ are preorders.*

PROOF. Similar to the proof of Proposition 4.8. □

The following partial completeness result is a variation of earlier results [Jonsson 1990; Lynch and Vaandrager 1995].

THEOREM 5.10. *(Partial completeness of normed backward simulations) If $A$ is a forest and $A \leq_{*\mathrm{T}} B$ then $A \leq_{\mathrm{B}} B$.*

PROOF. The relation $b \triangleq \mathit{after}(B) \circ \mathit{past}(A)$ is a branching backward simulation from $A$ to $B$. □

Note that by Proposition 5.6 we can strengthen the conclusion of Theorem 5.10 to $A \leq_{\mathrm{iB}} B$ in case $B$ has finite invisible nondeterminism.

*Example* 5.11. Consider the automata $A'$ and $B'$ in Figure 4. There exists no normed backward simulation from $B'$ to $A'$. The relation indicated by the dashed lines fails since the backward transition from state $u0$ cannot be simulated from the related state $s0$. Consequently, normed backward simulations do not provide a complete proof method for establishing trace inclusion. In the next section, we will see that completeness can be obtained by combining normed forward and backward simulations.

## 6. NORMED HISTORY RELATIONS

In this section we define *normed history relations*. These provide an abstract view of the *history variables* of Abadi and Lamport [1991], which in turn are abstractions of the *auxiliary variables* of Owicki and Gries [1976].

A pair $(r, n)$ is a *normed history relation* from $A$ to $B$ if $r$ is a step refinement from $B$ to $A$, and $(r^{-1}, n)$ is a normed forward simulation from $A$ to $B$. Write $A \leq_{\mathrm{H}} B$ if there exists a normed history relation from $A$ to $B$.

Clearly $A \leq_{\mathrm{H}} B$ implies $A \leq_{\mathrm{F}} B$ and $B \leq_{\mathrm{R}} A$. Through these implications, the preorder and soundness results for normed forward simulations and step refinements carry over to normed history relations. In fact, if $(r, n)$ is a normed history relation from $A$ to $B$ then $r$ is just a functional *branching bisimulation* from $B$ to $A$ in the sense of Van Glabbeek and Weijland [1996]. Hence, history relations preserve behavior of automata in a very strong sense. Intuitively, there is a history relation from $A$ to $B$ if $B$ can be obtained from $A$ by adding an extra state variable that records information about the history of an execution.

*Example* 6.1. Consider again the automata $A'$ and $B'$ in Figure 4. Together with an arbitrary norm function, the dashed lines constitute a normed history relation from $B'$ to $A'$. Because, as we observed, there is no step refinement from $B'$ to $A'$, there exists no normed history relation from $A'$ to $B'$.

An important example of a history relation is provided by the "unfolding" construction. The *unfolding* of an automaton $A$, notation $\mathit{unfold}(A)$, is the automaton obtained from $A$ by recording the complete history of an execution. Formally, $\mathit{unfold}(A)$ is the automaton $B$ defined by

—$\mathit{states}(B) = \mathit{execs}^*(A)$,
—$\mathit{start}(B) =$ the set of executions of $A$ that consist of a single start state,
—$\mathit{acts}(B) = \mathit{acts}(A)$, and
—for $\alpha', \alpha \in \mathit{states}(B)$ and $a \in \mathit{acts}(B)$, $\alpha' \xrightarrow{a}_B \alpha \quad \Leftrightarrow \quad \alpha = \alpha' \, a \, \mathit{last}(\alpha)$.

The next proposition relates an automaton to its unfolding.

Proposition 6.2. *$\mathit{unfold}(A)$ is a forest and $A \leq_{\mathrm{H}} \mathit{unfold}(A)$.*

Proof. Clearly, $\mathit{unfold}(A)$ is a forest. The function *last* which maps each finite execution of $A$ to its last state is a step refinement from $\mathit{unfold}(A)$ to $A$, and the relation $\mathit{last}^{-1}$, together with an arbitrary norm function, is a normed forward simulation from $A$ to $\mathit{unfold}(A)$. □

The following completeness theorem, a variation of a result due to Sistla [1991], asserts that normed history relations together with normed backward simulations constitute a complete proof method for establishing trace inclusion. Consequently, also normed forward simulations together with normed backward simulations constitute a complete proof method.

Theorem 6.3. *(Completeness of normed history relations and normed backward simulations)*
*If $A \leq_{*\mathrm{T}} B$ then there exists an automaton $C$ such that $A \leq_{\mathrm{H}} C \leq_{\mathrm{B}} B$.*

Proof. Take $C = \mathit{unfold}(A)$. By Proposition 6.2, $C$ is a forest and $A \leq_{\mathrm{H}} C$. Since $A \leq_{*\mathrm{T}} B$, also $C \leq_{*\mathrm{T}} B$ by soundness of history relations. Next apply the partial completeness result for backward simulations (Theorem 5.10) to conclude $C \leq_{\mathrm{B}} B$. □

Observe that if we can assume in addition that $B$ has fin, we may replace $\leq_{\mathrm{B}}$ by $\leq_{\mathrm{iB}}$ in the conclusion using Proposition 5.6.

Normed forward simulations are equivalent to normed history variables combined with step refinements: whenever there is a normed forward simulation from $A$ to $B$, we can find an intermediate automaton $C$ such that there is a normed history relation from $A$ to $C$ and a step refinement from $C$ to $B$. The converse implication trivially holds since normed history relations and step refinements are special cases of normed forward simulations. In order to prove the existence of automaton $C$, we need to define a notion of "superposition" of automata and to prove a technical lemma.

Let $R \subseteq \mathit{states}(A) \times \mathit{states}(B)$ be a relation with $R \cap (\mathit{start}(A) \times \mathit{start}(B)) \neq \emptyset$. The *superposition* $\mathit{sup}(A, B, R)$ of $A$ and $B$ via $R$ is the automaton $C$ defined by

—$states(C) = R$,
—$start(C) = R \cap (start(A) \times start(B))$,
—$acts(C) = acts(A) \cap acts(B)$, and
—for $(s, u), (t, v) \in states(C)$ and $a \in acts(C)$, $(s, u) \xrightarrow{a}_C (t, v) \quad \Leftrightarrow$

$$\begin{aligned} & a = \tau \wedge s = t \wedge u \xrightarrow{\tau}_B v \\ \vee \ & a = \tau \wedge u = v \wedge s \xrightarrow{\tau}_A t \\ \vee \ & s \xrightarrow{a}_A t \wedge u \xrightarrow{a}_B v. \end{aligned}$$

Essentially, the superposition $sup(A, B, R)$ is just the usual parallel composition of $A$ and $B$ with the set of states restricted to $R$.

LEMMA 6.4. *Suppose $(f, n)$ is a normed forward simulation from $A$ to $B$. Let $C = sup(A, B, f)$ and let $\pi_1$ and $\pi_2$ be the projection functions that map states of $C$ to their first and second components, respectively. Let $n'$ be the norm function given by $n'(\delta, u) = n(\delta, \pi_2(u))$. Then $(\pi_1, n')$ is a normed history relation from $A$ to $C$, and $\pi_2$ is a step refinement from $C$ to $B$.*

PROOF. Straightforward from the definitions. □

THEOREM 6.5. $A \leq_{\mathrm{F}} B \Leftrightarrow (\exists C : A \leq_{\mathrm{H}} C \leq_{\mathrm{R}} B)$.

PROOF. Forward implication follows by Lemma 6.4. For backward implication, suppose $A \leq_{\mathrm{H}} C \leq_{\mathrm{R}} B$. Then $A \leq_{\mathrm{F}} C$ by the definition of history relations, and $C \leq_{\mathrm{F}} B$ because any step refinement is a normed forward simulation. Now $A \leq_{\mathrm{F}} B$ follows by the fact that $\leq_{\mathrm{F}}$ is a preorder. □

Klop and Ariola [1996][Intermezzo 3.23] state a remarkable result: on a domain of of finitely branching process graphs (i.e., automata considered modulo isomorphism) the preorder induced by functional bisimulations (i.e., history relations) is in fact a partial order: $A \leq_{\mathrm{H}} B$ and $B \leq_{\mathrm{H}} A$ implies $A = B$. They also present a counterexample to show that the finite branching property is needed to prove this result. Below we present a slight generalization of their result [Ariola and Klop 1996] in the setting of our paper. It turns out to be sufficient to assume that automata have finite invisible nondeterminism (fin).

THEOREM 6.6. *Suppose $A$ and $B$ have fin, $A \leq_{\mathrm{H}} B$ and $B \leq_{\mathrm{H}} A$. Then the reachable subautomata of $A$ and $B$ are isomorphic.*

PROOF. Suppose that $(f, n)$ is a normed history relation from $A$ to $B$, and $(g, m)$ is a normed history relation from $B$ to $A$. Because $A$ and $B$ have fin, both $start(A)$ and $start(B)$ are finite. Since $f$ is a step refinement, it maps start states of $B$ to start states of $A$. Using the fact that $f^{-1}$ is a forward simulation, we infer that $f$ is surjective on start states. Hence $\mid start(B) \mid \ \leq \ \mid start(A) \mid$. By a similar argument, using the fact that $(g, m)$ is a normed history relation from $B$ to $A$, we obtain $\mid start(A) \mid \ \leq \ \mid start(B) \mid$. This means that $f$ is also injective on start states.

Let $\beta, \gamma$ be arbitrary traces of $A$ and $B$. Using a similar argument as above, we infer

$$\begin{aligned} f(after(A)[\beta] \cup after(A)[\gamma]) &= after(B)[\beta] \cup after(B)[\gamma] \\ g(after(B)[\beta] \cup after(B)[\gamma]) &= after(A)[\beta] \cup after(A)[\gamma] \end{aligned}$$

Since, by Lemma 2.1(2), all mentioned sets are finite, it follows that

$$| \, \mathit{after}(A)[\beta] \cup \mathit{after}(A)[\gamma] \, | \; = \; | \, \mathit{after}(B)[\beta] \cup \mathit{after}(B)[\gamma] \, |$$

This means that $f$ and $g$ are injective on the sets $\mathit{after}(B)[\beta] \cup \mathit{after}(B)[\gamma]$ and $\mathit{after}(A)[\beta] \cup \mathit{after}(A)[\gamma]$, respectively.

Since $f^{-1}$ is a forward simulation, $f$ is surjective on the reachable states. It remains to show that $f$ is injective on reachable states (once we have this, the required isomorphism property follows from the fact that $f$ is a step refinement). Suppose that $s$ and $t$ are reachable states of $B$ such that $f(s) = f(t)$. Then there are traces $\beta$ and $\gamma$ such that $s \in \mathit{after}(B)[\beta]$ and $t \in \mathit{after}(B)[\gamma]$. But since $f$ is injective on $\mathit{after}(B)[\beta] \cup \mathit{after}(B)[\gamma]$, this implies $s = t$. □

Intuitively, one may interpret the above result as follows: if $A \leq_{\mathrm{H}} B$ then $B$ contains as much *history information* as $A$. If $B$ contains as much history information as $A$, and $A$ contains as much history information as $B$, then they are equal.

## 7. NORMED PROPHECY RELATIONS

In this section, we will define normed prophecy relations and show that they correspond to normed backward simulations, very similarly to the way in which normed history relations correspond to normed forward simulations.

A pair $(r, n)$ is a *normed prophecy relation* from $A$ to $B$ if $r$ is a step refinement from $B$ to $A$ and $(r^{-1}, n)$ is a normed backward simulation from $A$ to $B$. We write $A \leq_{\mathrm{P}} B$ if there is a normed prophecy relation from $A$ to $B$, and $A \leq_{\mathrm{iP}} B$ if there is a normed prophecy relation $(r, n)$ with $r^{-1}$ image-finite. Thus $A \leq_{\mathrm{iP}} B$ implies $A \leq_{\mathrm{iB}} B$ and $A \leq_{\mathrm{P}} B$, and $A \leq_{\mathrm{P}} B$ implies $A \leq_{\mathrm{B}} B$ and $B \leq_{\mathrm{R}} A$. Moreover, if all states of $A$ are reachable, $B$ has finite invisible nondeterminism and $A \leq_{\mathrm{P}} B$, then $A \leq_{\mathrm{iP}} B$. It is easy to check that the preorder and soundness results for backward simulations and refinements carry over to prophecy relations.

The following lemma is the analogue of Lemma 6.4 in the backward setting. Using this lemma, we can prove that normed backward simulations are equivalent to normed prophecy variables combined with step refinements.

LEMMA 7.1. *Suppose $(b, n)$ is a normed backward simulation from $A$ to $B$. Let $C = \mathit{sup}(A, B, b)$ and let $\pi_1$ and $\pi_2$ be the projection functions that map states of $C$ to their first and second components, respectively. Let $n'$ be the norm function given by $n'(\delta, u) = n(\delta, \pi_2(u))$. Then $(\pi_1, n')$ is a normed prophecy relation from $A$ to $C$, and $\pi_2$ is a step refinement from $C$ to $B$. If $b$ is image-finite then so is $\pi_1^{-1}$.*

THEOREM 7.2.

*(1) $A \leq_{\mathrm{B}} B \Leftrightarrow (\exists C : A \leq_{\mathrm{P}} C \leq_{\mathrm{R}} B)$.*

*(2) $A \leq_{\mathrm{iB}} B \Leftrightarrow (\exists C : A \leq_{\mathrm{iP}} C \leq_{\mathrm{R}} B)$.*

PROOF. Analogous to that of Theorem 6.5, using Lemma 7.1. □

We can now prove variants of the well-known completeness result of Abadi and Lamport [1991].

THEOREM 7.3. *(Completeness of normed history+prophecy relations and step refinements)*
*Suppose $A \leq_{*\mathrm{T}} B$. Then*

(*1*) $\exists C, D : A \leq_{\mathrm{H}} C \leq_{\mathrm{P}} D \leq_{\mathrm{R}} B$.

(*2*) *If* $B$ *has fin then* $\exists C, D : A \leq_{\mathrm{H}} C \leq_{\mathrm{iP}} D \leq_{\mathrm{R}} B$.

Proof. By Theorem 6.3, there exists an automaton $C$ with $A \leq_{\mathrm{H}} C \leq_{\mathrm{B}} B$. Next, Theorem 7.2 yields the required automaton $D$ with $C \leq_{\mathrm{P}} D \leq_{\mathrm{R}} B$, which proves (1). The proof of (2) is similar, but uses Proposition 5.6. □

The following theorem states that $\leq_{\mathrm{P}}$ is a partial order on the class of automata with fin, considered modulo isomorphism of reachable subautomata. The proof is analogous to that of Theorem 6.6, the corresponding result for normed history relations.

Theorem 7.4. *Suppose* $A$ *and* $B$ *have fin,* $A \leq_{\mathrm{P}} B$ *and* $B \leq_{\mathrm{P}} A$*. Then the reachable subautomata of* $A$ *and* $B$ *are isomorphic.*

## 8. DECIDABILITY

Thus far, our exposition has been purely semantic. In the words of Abadi and Lamport [1991]: "We have considered specifications, but not the languages in which they are expressed. We proved the existence of refinement mappings, but said nothing about whether they are expressible in any language." In this section, we move to the syntactic world and discuss some decidability issues. To this end we have to fix a language for defining automata. The language below can be viewed as a simplified version of the IOA language of Garland et al. [1997].

We assume an underlying assertion language $\mathcal{L}$ which is a first-order language over interpreted symbols for expressing functions and predicates over some concrete domains such as integers, arrays, and lists of integers. If $X$ is a set of (typed) variables then we write $F(X)$ and $E(X)$ for the collection of formulas and expressions, respectively, in which variables from $X$ may occur free. An automaton can be described syntactically by first specifying a finite set $X$ of variables, referred to as the *state variables.* For each state variable $x$ we assume the presence of a copy $x'$, called the *primed version* of $x$. We write $X'$ for the set $\{x' \mid x \in X\}$ and, if $\phi$ is a formula then we write $\phi'$ for the formula obtained from $\phi$ by replacing each occurrence of a state variable by its primed version. The set of states of the automaton is defined as the set of all valuations of the state variables in $X$. The set of initial states is specified by a predicate in $F(X)$, called the *initial condition.* The actions are specified via a finite number of *action names* with, for each action name $a$, a finite list $\vec{v}$ of variables called the *parameters* of $a$. We assume $\{\vec{v}\} \cap X = \emptyset$. The set of actions of the automaton is defined as the union, for each action name $a$, of all tuples $a(\vec{d})$, where $\vec{d}$ is a valuation of the parameters $\vec{v}$ in their respective domains. The transition relation is specified by providing, for each action name $a$ with parameters $\vec{v}$, a *transition predicate* in $F(X \cup \{\vec{v}\} \cup X')$, i.e., a predicate that may contain action parameters as well as primed and unprimed state variables.

*Example* 8.1. Below we specify a FIFO channel in IOA syntax [Garland et al. 1997].

```
automaton Channel
  states
    buffer: Seq[Nat]
```

```
initial condition
  buffer = {}
actions
  send(v: Nat),
  receive(v: Nat),
  tau
transitions
  action send(v)
    predicate buffer' = buffer |- v
  action receive(v)
    predicate buffer ~= {} /\ v = head(buffer)
                /\ buffer' = tail(buffer)
  action tau
    predicate false
```

In IOA datatypes are specified using the Larch specification language [Guttag and Horning 1993]. In the example we use the standard finite list datatype, with {} denoting the empty list, `|-` denotes the opereration that appends an element to the end of a list, etc. Transitions are specified in a standard predicative style. The example automaton has no $\tau$ transitions, which is specified by the transition predicate `false`.

This piece of syntax defines an automaton $A$ with

—$states(A) = \mathsf{N}^*$,

—$start(A) = \{\lambda\}$,

—$acts(A) = \{\mathtt{send}(d),\ \mathtt{receive}(d) \mid d \in \mathsf{N}\} \cup \{\tau\}$,

—$steps(A)$ is the least set that contains the following elements, for all $\sigma \in \mathsf{N}^*$ and $d \in \mathsf{N}$,

$$\begin{array}{ccc} \sigma & \xrightarrow{\mathtt{send}(d)} & \sigma\, d \\ d\,\sigma & \xrightarrow{\mathtt{receive}(d)} & \sigma. \end{array}$$

Now assume that we have specified two automata $A$ and $B$, using state variables $\vec{x}$ and $\vec{y}$, respectively. Let $X = \{\vec{x}\}$ and $Y = \{\vec{y}\}$. Assume $X \cap Y = \emptyset$.

A step refinement from $A$ to $B$ can be specified by a formula of the form $\theta \wedge \vec{y} = \vec{e}$, with $\theta \in E(X)$ and $\vec{e}$ a list of expressions in $E(X)$ that matches $\vec{y}$ in terms of length and types. In this formula, the first conjunct defines the domain of the step refinement whereas the second conjunct defines a map from states of $A$ to states of $B$ by specifying, for each state variable of $B$, its value in terms of the values of the state variables of $A$.

A normed forward simulation can be described by a predicate in $F(X \cup Y)$ together with, for each action type $a$ with parameters $\vec{v}$, an expression in $E(X \cup \{\vec{v}\} \cup X' \cup Y)$ that specifies the norm function. In practice, norm functions often only depend on the states of $B$, which means that they can be specified by means of a single expression in $E(Y)$.

*Example* 8.2. Consider the following specification, essentially just the chaining of two FIFO channels.

```
automaton TwoChannels
  states
    buffer1: Seq[Nat],
    buffer2: Seq[Nat]
  initial condition
    buffer1 = {} /\ buffer2 = {}
  actions
    send(v: Nat),
    receive(v: Nat),
    tau
  transitions
    action send(v)
      predicate buffer1' = buffer1 |- v /\ buffer2' = buffer2
    action receive(v)
      predicate buffer2 ~= {} /\ v = head(buffer2)
                /\ buffer2' = tail(buffer2) /\ buffer1' = buffer1
    action tau
      predicate buffer1 ~= {} /\ buffer1' = tail(buffer1) /\
               /\ buffer2' = buffer2 |- head(buffer1)
```

Let $B$ be the automaton denoted by this specification. It is easy to prove that the formula below (where `||` denotes concatenation of lists) defines a step refinement from $B$ to the automaton $A$ of Example 8.1.

```
buffer = buffer2 || buffer1
```

It is also routine to check that this formula together with the norm on states of $B$ defined by

```
if buffer1 ~= {} /\ buffer2 = {} then 1 else 0
```

defines a normed forward simulation from $A$ to $B$.

We will now show that, under some reasonable (sufficient but certainly not necessary) assumptions, it is in fact decidable whether a given predicate/expression indeed corresponds to a step refinement or normed forward simulation. Assume that automaton $A$ is described using state variables $\vec{x}$, initial condition $\varphi_0$ and, for each action name $a$, a transition predicate $\varphi_a$. Likewise, assume that automaton $B$ is described using state variables $\vec{y}$, initial condition $\psi_0$ and, for each action name $a$, a transition predicate $\psi_a$. Assume further that each action name $a$ of $A$ is also an action name of $B$, and that $a$ has the same parameters in both $A$ and $B$. Write $P_a$ for the list of parameters of $a$. We require that $P_\tau = \emptyset$.

Suppose that we want to check whether a formula $\rho \triangleq \theta \wedge \vec{y} = \vec{e}$ denotes a step refinement. This is equivalent to proving validity of the following formula:

$$
\begin{array}{lrcl}
 & \varphi_0 & \Rightarrow & \theta \\
\bigwedge & \varphi_0 \wedge \rho & \Rightarrow & \psi_0 \\
\bigwedge_a & \varphi_a \wedge \theta & \Rightarrow & \theta' \\
\bigwedge_{a \neq \tau} & \varphi_a \wedge \rho \wedge \rho' & \Rightarrow & \psi_a \\
\bigwedge & \varphi_\tau \wedge \rho \wedge \rho' & \Rightarrow & \psi_\tau \vee \vec{y} = \vec{y'}
\end{array}
$$

In this formula, the first conjunct asserts that the function is defined for start states of $A$; the second conjunct that start states of $A$ are mapped onto start states of $B$; the third conjunct that if the function is defined for the source of a transition then it is also defined for the target state of a transition; and the two final conjuncts encode the transfer condition. Thus checking whether a partial function is a step refinement from $A$ to $B$ is decidable if the partial function as well as $A$ and $B$ can all be expressed within a fragment of $\mathcal{L}$ for which tautology checking is decidable.

Next suppose that we want to check whether a formula $\rho$ together with norm expressions $n_a$, for each action name $a$, denotes a normed forward simulation from $A$ to $B$. In order to turn this into a decidable question, we have to make some additional assumptions about the specification of $B$. We assume that $B$ has finitely many start states[2], which are listed explicitly, i.e., we require that the initial condition $\psi_0$ is of the form

$$\psi_0 \;=\; \bigvee_{i \in I_0} \vec{y} = \vec{e_0^i} \tag{3}$$

where $I_0$ is a finite index set and, for each $i$, $\vec{e_0^i}$ is a list of closed terms. In addition we assume that in any state and for any given value of the action parameters, only finitely many transitions are possible in $B$, which are listed explicitly. Formally we require that, for each action type $a$, transition predicate $\psi_a$ is of the form

$$\psi_a \;=\; \bigvee_{i \in I_a} (\chi_a^i \wedge \vec{y'} = \vec{e_a^i}) \tag{4}$$

where $I_a$ is a finite index set and, for each $i$, $\chi_a^i$ is a formula in $F(Y \cup \{P_a\})$ and $\vec{e_0^i}$ is a list of expressions in $E(Y \cup \{P_a\})$. Basically, $\chi_a^i$ gives the precondition of the $i$-th instance of transition $a$ and $\vec{y'} = \vec{e_a^i}$ specifies the effect of taking it. Both assumption (3) and (4) are satisfied by most automaton specifications that one encounters in practice. In particular, the assumptions hold for the channels specified in Examples 8.1 and 8.1. Only specifications that involve a nondeterministic choice that is not a priori bounded fall outside of our format. An example of this, described by Sogaard-Andersen et al. [1993], is a FIFO channel in which a crash action may result in the loss of an arbitrary subset of the messages contained in a buffer. Under assumptions (3) and (4), we can eliminate the existential quantifiers that occur in the definition of a normed forward simulation, and checking the conditions in this definition becomes equivalent to proving validity of the following formula:

$$\begin{aligned}
\varphi_0 \;&\Rightarrow\; \bigvee_{i \in I_0} \rho[\vec{e_0^i}/\vec{y}] \\
\bigwedge_{a \neq \tau} \varphi_a \wedge \rho \;&\Rightarrow\; \bigvee_{i \in I_a} (\chi_a^i \wedge \rho'[\vec{e_a^i}/\vec{y'}]) \vee \bigvee_{i \in I_\tau} (\chi_\tau^i \wedge \rho[\vec{e_\tau^i}/\vec{y}] \wedge n_a[\vec{e_\tau^i}/\vec{y}] < n_a) \\
\bigwedge \varphi_\tau \wedge \rho \;&\Rightarrow\; \rho'[\vec{y}/\vec{y'}] \vee \bigvee_{i \in I_\tau} (\chi_\tau^i \wedge \rho'[\vec{e_\tau^i}/\vec{y'}]) \vee \bigvee_{i \in I_\tau} (\chi_\tau^i \wedge \rho[\vec{e_\tau^i}/\vec{y}] \wedge n_\tau[\vec{e_\tau^i}/\vec{y}] < n_\tau)
\end{aligned}$$

[2]This assumption can be relaxed if we assume that the value of certain state variables of $B$ is fully determined by $\rho$ and the state of $A$: for those state variables the initial value can be left unspecified.

If this formula can be expressed within a fragment of $\mathcal{L}$ for which tautology checking is decidable then it is decidable whether $\rho$ together with expressions $n_a$ constitutes a normed forward simulation. It is easy to see that a similar result can also be obtained for normed history variables. Thus far, however, we have not been able to come up with plausible syntactic restrictions, applicable in practical cases, that ensure decidability of normed backward simulations and/or normed prophecy relations. It is for instance not clear how one can eliminate the existential quantifier in the formula that asserts that in a normed backward simulation for each state of $A$ there exists a related state of $B$. We think this constitutes an interesting area for future research.

Our decidability results for step refinements and normed forward simulations do not carry over to the refinements and forward simulations as described, for instance, by Lynch and Vaandrager [1995]. In order to see this, let $A$ be a system with two states, an initial and a final one, and a single transition labeled *halt* from the initial to the final state. Let $B$ be a system that simulates the $n$-th Turing machine such that each computation step of the Turing machine corresponds with a $\tau$-move, and that moves via a *halt*-action to a designated final state if and only if the computation of the Turing machine terminates. The function that maps the initial state of $A$ to the initial state of $B$ and the final state of $A$ to the final state of $B$ is a weak refinement iff the $n$-th Turing machine halts. It is straightforward to specify $A$, $B$ and the function from states of $A$ to states of $B$ in a decidable logic. Hence it is undecidable whether a given function is a weak refinement, even in a setting where the underlying logic is decidable.

## 9. REACHABILITY

For the sake of simplicity, all definitions of simulations and refinements so far have been presented without any mention of reachability or invariants. However, in practical verifications it is almost always the case that first some invariants (properties that hold for all reachable states) are established for the lower-level and/or higher-level specification. These invariants are then used in proving the step correspondence. In this section we show how to integrate reachability concerns into the simulation definitions. More specifically, we present adapted versions of step refinements, normed forward simulations and normed backward simulations which include reachability concerns, and discuss their relationship with the original definitions. For examples of the use of these adapted definitions and their formalization in PVS, we refer to our earlier work [Griffioen 2000].

An adapted *step refinement* from $A$ to $B$ consists of a partial function $r : \mathit{states}(A) \to \mathit{states}(B)$ satisfying the following two conditions:

(1) If $s \in \mathit{start}(A)$ then $s \in \mathit{domain}(r)$ and $r(s) \in \mathit{start}(B)$.
(2) If $s \xrightarrow{a}_A t \wedge s \in \mathit{domain}(r) \wedge \mathit{reachable}(A, s) \wedge \mathit{reachable}(B, r(s))$ then $t \in \mathit{domain}(r)$ and
   (a) $r(s) = r(t) \wedge a = \tau$, or
   (b) $r(s) \xrightarrow{a}_B r(t)$.

Clause $\mathit{reachable}(A, s)$ in condition (2) allows reuse of invariants previously established for lower-level specification $A$, whereas clause $\mathit{reachable}(B, r(s))$ in condition

(2) makes it possible to reuse known invariants of higher-level specification $B$. The adapted definition can easily be seen as a special case of the original definition in Section 3.1: if $r$ is an adapted step refinement then the restriction $r'$ of $r$ defined by

$$s \in \mathit{domain}(r') \triangleq s \in \mathit{domain}(r) \wedge \mathit{reachable}(A, s) \wedge \mathit{reachable}(B, r(s)),$$

is a regular step refinement. Conversely, any regular step refinement trivially satisfies the conditions of the adapted version.

An adapted *normed forward simulation* from $A$ to $B$ consists of a relation $f \subseteq \mathit{states}(A) \times \mathit{states}(B)$ and a function $n : \mathit{steps}(A) \times \mathit{states}(B) \to S$, for some well-founded set $S$, such that:

(1) If $s \in \mathit{start}(A)$ then $f[s] \cap \mathit{start}(B) \neq \emptyset$.
(2) If $s \xrightarrow{a}_A t \wedge u \in f[s] \wedge \mathit{reachable}(A, s) \wedge \mathit{reachable}(B, u)$ then
  (a) $u \in f[t] \wedge a = \tau$, or
  (b) $\exists v \in f[t] : u \xrightarrow{a}_B v$, or
  (c) $\exists v \in f[s] : u \xrightarrow{\tau}_B v \wedge n(s \xrightarrow{a} t, v) < n(s \xrightarrow{a} t, u)$.

Again, the clause $\mathit{reachable}(A, s)$ in condition (2) allows us to reuse invariants that have previously been established for $A$, whereas clause $\mathit{reachable}(B, u)$ in condition (2) permits reuse of invariants of $B$. And again the adapted definition can easily been seen as a special case of the original definition (in Section 4): if $(f, n)$ is an adapted normed forward simulation then the pair $(g, n)$, where $g = f \cap (\mathit{reachable}(A) \times \mathit{reachable}(B))$, is a regular normed forward simulation. Conversely, any regular normed forward simulation trivially is an adapted normed forward simulation.

An adapted *normed backward simulation* from $A$ to $B$ consists of a relation $b \subseteq \mathit{states}(A) \times \mathit{states}(B)$, a predicate $Q \subseteq \mathit{states}(B)$, and a function $n : (\mathit{steps}(A) \cup \mathit{start}(A)) \times \mathit{states}(B) \to S$, for some well-founded set $S$, such that:

(1) If $s \in \mathit{start}(A) \wedge u \in b[s] \wedge Q(u)$ then
  (a) $u \in \mathit{start}(B)$, or
  (b) $\exists v \in b[s] : v \xrightarrow{\tau}_B u \wedge n(s, v) < n(s, u) \wedge Q(v)$.
(2) If $t \xrightarrow{a}_A s \wedge u \in b[s] \wedge \mathit{reachable}(A, t) \wedge Q(u)$ then
  (a) $u \in b[t] \wedge a = \tau$, or
  (b) $\exists v \in b[t] : v \xrightarrow{a}_B u \wedge Q(v)$, or
  (c) $\exists v \in b[s] : v \xrightarrow{\tau}_B u \wedge n(t \xrightarrow{a} s, v) < n(t \xrightarrow{a} s, u) \wedge Q(v)$.
(3) If $\mathit{reachable}(A, s)$ then $\exists u \in b[s] : Q(u)$.

Clause $\mathit{reachable}(A, t)$ in condition (2) allows us to reuse invariants that have previously been established for $A$, and clause $Q(u)$ in condition (2) permits reuse of invariants of $B$. Note that by a trivial inductive argument a backward simulation can never relate a reachable state of $A$ to a non-reachable state of $B$. Thus we can safely restrict the range of any backward simulation by all invariants proven for $B$. To this end predicate $Q$ has been included in the definition of the adapted normed backward simulation, even though strictly speaking (1) $Q$ need not be an invariant, and (2) $Q$ can always be eliminated by restricting the range of $b$. Once more the adapted definition is a special case of the original definition (in Section 5): if $(b, n)$ is an adapted normed backward simulation then $(b, n)$ is also a regular normed backward simulation from the automaton $A'$, that restricts $A$ to its reachable states, to the automaton $B'$, that restricts $B$ to the states in $Q$. Conversely, any regular normed backward simulation trivially is an adapted normed backward simulation with $Q = states(B)$.

We leave it up to the reader to work out adapted versions of the normed history and prophecy relations.

## Acknowledgement

We thank Mariëlle Stoelinga and Ling Cheung for spotting mistakes (and proposing fixes) in an earlier version of this paper, and Jan Willem Klop for discussions that led us to Theorem 6.6.

## REFERENCES

Abadi, M. and Lamport, L. 1991. The existence of refinement mappings. *Theor. Comput. Sci. 82,* 2, 253–284.

Ariola, Z. and Klop, J. 1996. Equational term graph rewriting. *Fundamenta Informaticae 26,* 3/4, 207–240. Extended version as University of Oregon Technical Report CIS-TR-95-16.

Baier, C. and Stoelinga, M. 2000. Norm functions for probabilistic bisimulations with delays. In *Proceedings of 3rd International Conference on Foundations of Science and Computation Structures (FOSSACS),* Berlin, Germany, March 2000, J. Tiuryn, Ed. Lecture Notes in Computer Science, vol. 1784. Springer-Verlag, 1–16.

Basten, T. 1996. Branching bisimilarity is an equivalence indeed! *Information Processing Letters 58,* 3, 141–147.

Bensalem, S., Ganesh, V., Lakhnech, Y., noz, C. M., Owre, S., Ruess, H., Rushby, J., Rusu, V., Saïdi, H., Shankar, N., Singerman, E., and Tiwari, A. 2000. An overview of SAL. In *LFM 2000: Fifth NASA Langley Formal Methods Workshop*, C. M. Holloway, Ed. NASA Langley Research Center, Hampton, VA, 187–196.

Bensalem, S., Lakhnech, Y., and Saidi, H. 1996. Powerful techniques for the automatic generation of invariants. In *Proceedings of the 8th International Conference on Computer Aided Verification,* New Brunswick, NJ, USA, R. Alur and T. Henzinger, Eds. Lecture Notes in Computer Science, vol. 1102. Springer-Verlag, 323–335.

Browne, M., Clarke, E., and Grümberg, O. 1988. Characterizing finite Kripke structures in propositional temporal logic. *Theor. Comput. Sci. 59,* 1,2, 115–131.

De Nicola, R. and Vaandrager, F. 1995. Three logics for branching bisimulation. *Journal of the ACM 42,* 2 (Mar.), 458–487.

Devillers, M., Griffioen, W., and Müller, O. 1997. Possibly infinite sequences: A comparative case study. In *10th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'97)*, E. Gunter and A. Felty, Eds. Lecture Notes in Computer Science, vol. 1275. Springer-Verlag, 89–104.

Devillers, M., Griffioen, W., Romijn, J., and Vaandrager, F. 2000. Verification of a leader election protocol: Formal methods applied to IEEE 1394. *Formal Methods in System Design 16,* 3 (June), 307–320.

Garland, S., Lynch, N., and Vaziri, M. 1997. IOA: A language for specifiying, programming, and validating distributed systems. Available through URL `http://larch.lcs.mit.edu:8001/~garland/ioaLanguage.html`.

Gawlick, R., Segala, R., Søgaard-Andersen, J., and Lynch, N. 1993. Liveness in timed and untimed systems. Tech. Rep. MIT/LCS/TR-587, Laboratory for Computer Science, MIT, Cambridge, MA. Dec.

Ginzburg, A. 1968. *Algebraic Theory of Automata.* Academic Press, New York – London.

GLABBEEK, R. V. AND WEIJLAND, W. 1996. Branching time and abstraction in bisimulation semantics. *Journal of the ACM 43,* 3, 555–600.

GRIFFIOEN, W. 2000. Studies in computer aided verification of protocols. Ph.D. thesis, University of Nijmegen. Postscript and PVS sources available via `http://www.cs.kun.nl/ita/former_members/davidg/`.

GRIFFIOEN, W. AND VAANDRAGER, F. 1998. Normed simulations. In *Proceedings of the 10th International Conference on Computer Aided Verification,* Vancouver, BC, Canada, A. Hu and M. Vardi, Eds. Lecture Notes in Computer Science, vol. 1427. Springer-Verlag, 332–344.

GROOTE, J. AND SPRINGINTVELD, J. 1995. Focus points and convergent process operators — a proof strategy for protocol verification. Report CS-R9566, Department of Software Technology, CWI, Amsterdam. Nov.

GUTTAG, J. AND HORNING, J. 1993. *Larch: Languages and Tools for Formal Specification.* Springer-Verlag.

HELMINK, L., SELLINK, M., AND VAANDRAGER, F. 1994. Proof-checking a data link protocol. In *Proceedings International Workshop TYPES'93,* Nijmegen, The Netherlands, May 1993, H. Barendregt and T. Nipkow, Eds. Lecture Notes in Computer Science, vol. 806. Springer-Verlag, 127–165.

JONSSON, B. 1985. A model and proof system for asynchronous networks. In *Proceedings of the $4^{th}$ Annual ACM Symposium on Principles of Distributed Computing,* Minaki, Ontario, Canada. 49–58.

JONSSON, B. 1987. Compositional verification of distributed systems. Ph.D. thesis, Department of Computer Systems, Uppsala University. DoCS 87/09.

JONSSON, B. 1990. On decomposing and refining specifications of distributed systems. In *Proceedings REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness,* Mook, The Netherlands, May/June 1989, J. de Bakker, W. d. Roever, and G. Rozenberg, Eds. Lecture Notes in Computer Science, vol. 430. Springer-Verlag, 361–387.

JONSSON, B. 1991. Simulations between specifications of distributed systems. In *Proceedings CONCUR 91,* Amsterdam, J. Baeten and J. Groote, Eds. Lecture Notes in Computer Science, vol. 527. Springer-Verlag, 346–360.

JONSSON, B. 1994. Compositional specification and verification of distributed systems. *ACM Trans. Program. Lang. Syst. 16,* 2 (Mar.), 259–303.

KLARLUND, N. AND SCHNEIDER, F. 1989. Verifying safety properties using infinite-state automata. Tech. Rep. 89-1039, Department of Computer Science, Cornell University, Ithaca, New York.

KLARLUND, N. AND SCHNEIDER, F. 1993. Proving nondeterministically specified safety properties using progress measures. *Information and Computation 107,* 1 (Nov.), 151–170.

KNUTH, D. 1997. *Fundamental Algorithms.* The Art of Computer Programming, vol. 1. Addison-Wesley, Reading, Massachusetts. Third edition.

LAKHNECH, Y., BENSALEM, S., BEREZIN, S., AND OWRE, S. 2001. Incremental verification by abstraction. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems,* Genova, Italy, T. Margaria and W. Yi, Eds. Lecture Notes in Computer Science, vol. 2031. Springer-Verlag.

LAMPORT, L. 1983. What good is temporal logic? In *Information Processing 83*, R. Mason, Ed. North-Holland, 657–668.

LYNCH, N. 1996. *Distributed Algorithms.* Morgan Kaufmann Publishers, Inc., San Fransisco, California.

LYNCH, N. AND TUTTLE, M. 1987. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the $6^{th}$ Annual ACM Symposium on Principles of Distributed Computing.* 137–151. A full version is available as MIT Technical Report MIT/LCS/TR-387.

LYNCH, N. AND VAANDRAGER, F. 1995. Forward and backward simulations, I: Untimed systems. *Information and Computation 121,* 2 (Sept.), 214–233.

LYNCH, N. AND VAANDRAGER, F. 1996. Forward and backward simulations, II: Timing-based systems. *Information and Computation 128,* 1 (July), 1–25.

MANNA, Z., BROWNE, A., SIPMA, H., AND URIBE, T. 1998. Visual abstraction for temporal verification. In *Proceedings AMAST'98*, A. Haeberer, Ed. Lecture Notes in Computer Science, vol. 1548. Springer-Verlag, 28–41.

MILNER, R. 1971. An algebraic definition of simulation between programs. In *Proceedings* $2^{nd}$ *Joint Conference on Artificial Intelligence*. BCS, 481–489. Also available as Report No. CS-205, Computer Science Department, Stanford University, February 1971.

MILNER, R. 1989. *Communication and Concurrency*. Prentice-Hall International, Englewood Cliffs.

MUELLER, O. 1998. A verification environment for i/o automata based on formalized meta-theory. Ph.D. thesis, Technical University of Munich.

NAMJOSHI, K. 1997. A simple characterization of stuttering bisimulation. In *Proceedings 17th Conference on Foundations of Software Technology and Theoretical Computer Science,* Kharagpur, India, S. Ramesh and G. Sivakumar, Eds. Lecture Notes in Computer Science, vol. 1346. Springer-Verlag, 284–296.

NIPKOW, T. AND SLIND, K. 1995. I/O automata in Isabelle/HOL. In *Types for Proofs and Programs*, P. Dybjer, B. Nordström, and J. Smith, Eds. Lecture Notes in Computer Science, vol. 996. Springer-Verlag, 101–119.

OWICKI, S. AND GRIES, D. 1976. An axiomatic proof technique for parallel programs. *Acta Inf. 6,* 4, 319–340.

OWRE, S., RUSHBY, J., SHANKAR, N., AND HENKE, F. V. 1995. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering 21,* 2 (Feb.), 107–125.

ROEVER, W. D. AND ENGELHARDT, K. 1998. *Data Refinement: Model-Oriented Proof Methods and their Comparison.* Cambridge Tracts in Theoretical Computer Science 47. Cambridge University Press.

SISTLA, A. 1991. Proving correctness with respect to nondeterministic safety specifications. *Inf. Process. Lett. 39,* 1 (July), 45–49.

SØGAARD-ANDERSEN, J., GARLAND, S., GUTTAG, J., LYNCH, N., AND POGOSYANTS, A. 1993. Computer-assisted simulation proofs. In *Proceedings of the 5th International Conference on Computer Aided Verification,* Elounda, Greece, C. Courcoubetis, Ed. Lecture Notes in Computer Science, vol. 697. Springer-Verlag, 305–319.

SØGAARD-ANDERSEN, J., LYNCH, N., AND LAMPSON, B. 1993. Correctness of communication protocols – a case study. Tech. Rep. MIT/LCS/TR-589, Laboratory for Computer Science, MIT, Cambridge, MA. Nov.

STARK, E. 1988. Proving entailment between conceptual state specifications. *Theor. Comput. Sci. 56*, 135–154.

WOLPER, P. 1997. The meaning of formal: from weak to strong formal methods. *Springer International Journal on Software Tools for Technology Transfer 1,* 1-2, 6–8.

Received July 2000; revised September 2002 and April 2003; accepted April 2003