Normed Simulations

David Griffioen[1,2]* Frits Vaandrager[2]

[1] CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

[2] Computing Science Institute, University of Nijmegen
P.O. Box 9010, 6500 GL Nijmegen, The Netherlands
{davidg,fvaan}@cs.kun.nl

Abstract. In existing simulation proof techniques, a single step in a low-level system may be simulated by an extended execution fragment in a high-level system. As a result, it is undecidable whether a given relation is a simulation, even if tautology checking is decidable for the underlying specification logic. This paper introduces various types of *normed simulations*. In a normed simulation, each step in a low-level system can be simulated by at most one step in the high level system, for any related pair of states. We show that it is decidable whether a given relation is a normed simulation relation, given that tautology checking is decidable. We also prove that, at the semantic level, normed simulations form a complete proof method for establishing behavior inclusion, provided that the high-level system has finite invisible nondeterminism. As an illustration of our method we discuss the verification in PVS of a leader election algorithm that is used within the IEEE 1394 protocol.

1 Introduction

Simulation relations and refinement functions are widely used to prove that a low-level specification of a reactive system correctly implements a higher-level one [1, 13]. Technically, a *simulation* (or *refinement*) is a relation (or function) R between the states of a low-level system A and a high-level system B, that satisfies conditions such as

$$(s,u) \in R \wedge s \xrightarrow{a}_A s' \Rightarrow \exists u' : u \xrightarrow{a}_B u' \wedge (s',u') \in R \qquad (1)$$

(If a low-level state s and a high-level state u are related, and A can make a transition from s to s', then there exists a matching transition in B from u to a state u' that is related to s'.) The existence of a simulation implies that any behavior that can be exhibited by A can also be exhibited by B.

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

* Supported by the Netherlands Organization for Scientific Research (NWO) under contract SION 612-316-125.

in the case of systems that perform internal (or stuttering) steps. The usual transfer condition for *forward simulations* [13], for instance, says

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

(Each low-level transition can be simulated by a sequence of transitions which, apart from the action that has to be matched, may also contain an arbitrary number of internal steps.) 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, the occurrence of executions in transfer condition (2) usually does not pose a real problem: often the matching execution fragments that have to be constructed are short since internal steps are rare in high-level specifications; 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 (see [5] for a comparative study). In fact, in several papers in which formalizations of simulation proofs are described, the authors only define a restricted type of simulation or refinement in which each transition of the low-level system is formalized by one or zero transitions of the high-level system [11,15,6]. In approaches such as [18], in which the full transfer condition (2) is formalized, the user has to supply the simulating execution fragment α to the prover explicitly in each case of the proof, which makes the verification process highly interactive.

In this paper, we introduce a simulation proof method which remedies the above problems. The key idea is to define a function n that assigns a norm $n(s \xrightarrow{a} s', u)$, in some well-founded domain, to each pair of a transition in A and a state of B. If u has to simulate step $s \xrightarrow{a} s'$ then it may either do nothing (if a is internal and s' is related to u), or it may do a corresponding a-step, or it may perform an internal action leading to a state u' such that the norm $n(s \xrightarrow{a} s', u')$ decreases. We establish that the *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 the history and prophecy variables of [1]) can be enriched with a norm function, to obtain another complete proof method in combination with a simple notion of refinement mapping.

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 [16]. 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 the responsibility of the user, even though some very powerful heuristics have been devised to automate this search [2]. Analogously, if systems A and B, and a conjectured simulation relation R

and norm function n can all be expressed within a decidable assertion logic, and if the transition relations of A and B can be specified using a finite number of deterministic transition predicates, then it is decidable whether the pair (R, n) is a normed simulation. This result, which does not hold for other methods such as [1,13], is a distinct advantage of normed simulations.

The preorders generated by normed forward simulations are strictly finer than the preorders induced by the simulations of [13]. In fact, it is easy to characterize normed forward simulations in terms of *branching simulations* [9]. We believe it will be possible to come up with a notion of normed simulation that induces the same preorder as forward simulations, but technically this will be much more involved. In [9] it is argued that branching bisimulations have much nicer mathematical properties than Milner's weak bisimulations. Similarly, the mathematical theory of normed simulations appears to be nicer and more tractable than the theory of simulations developed in [13].

The idea of using norm functions to prove simulation relations also occurs in [10], where it is used to prove branching bisimilarity in the context of the process algebra μCRL. However, in [10] the norm function is defined on the states of B only, and does not involve the transitions of A. Furthermore the method of [10] only applies to divergence free processes. Norm functions very similar to ours were also studied by Namjoshi [14]. He uses them to obtain a characterization of the stuttering bisimulation of [3], which is the equivalent of branching bisimulation in a setting where states rather than actions are labelled (see [4]). Both [10] and [14] do not 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 [10] or a state based setting such as [14].

As a substantial example of the use of normed simulations, we discuss the formalization in PVS of the verification of a leader election algorithm that plays a role in the tree identify phase of the physical layer of the IEEE 1394 protocol [12,6]. We establish a normed prophecy relation from a high-level specification of the protocol to an intermediate specification, and a normed history relation from the intermediate specification to a low-level specification.

2 A Theory of Normed Simulations

In this section we build on some (standard) definitions and notations presented in [13]. In fact, our aim is to derive the same results as in [13], only for different types of simulations.

2.1 Step Refinements

The simplest type of simulation we consider is a *step refinement*. A *step refinement* from automaton A to automaton 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 $s \xrightarrow{a}_A s' \wedge s \in domain(r)$ then $s' \in domain(r)$ and
   - $r(s) = r(s') \wedge a = \tau$, or
   - $r(s) \xrightarrow{a}_B r(s')$.

Write $A \leq_{\mathrm{R}} B$ if there exists a step refinement from A to B. It is easy to check that $\leq_{\mathrm{R}}$ is a preorder (i.e., is transitive and reflexive). If $A \leq_{\mathrm{R}} B$ then we can construct, for each execution α of A, a corresponding execution of B with the same trace. This idea is formalized below.

Suppose A and B are automata, $R \subseteq states(A) \times states(B)$, 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 executions of A and B, respectively. Let $index(\alpha)$ and $index(\alpha')$ denote the index sets of α and α'. We say that α and α' are R-related, written $(\alpha, \alpha') \in R$, if there exists an *index mapping*, i.e., a total, nondecreasing function $m : index(\alpha) \to index(\alpha')$ such that, for all $i \in index(\alpha)$ and $j \in index(\alpha')$,

1. $m(0) = 0$,
2. $(s_i, u_{m(i)}) \in R$,
3. $i > 0 \quad \Rightarrow \quad a_i = b_{m(i)} \vee (a_i = \tau \wedge m(i) = m(i-1))$,
4. $m(i) < j \wedge (i + 1 \in index(\alpha) \Rightarrow j < m(i+1)) \quad \Rightarrow \quad (s_i, u_j) \in R \wedge b_j = \tau$.

Write $(A, B) \in R$ if for every execution α of A there is an execution α' of B such that $(\alpha, \alpha') \in R$, and write $[A, B] \in R$ if for every finite execution α of A there is a finite execution α' of B such that $(\alpha, \alpha') \in R$.

An index mapping maps low-level states to corresponding high-level states such that the start states correspond (Condition 1), corresponding states are related by R (Condition 2), each non-τ action in the low-level execution corresponds to an action in the high-level execution (Condition 3), and each non-τ action in the high-level execution corresponds to an action in the low-level execution (Condition 4). Our notion of correspondence is similar to the one presented in [8, 19]. 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 the one of [8, 19], but has the advantage that it also preserves until properties.

Theorem 1. *(Execution correspondence) (1) If $(\alpha, \alpha') \in R$ then $trace(\alpha) = trace(\alpha')$. (2) If $(A, B) \in R$ then $A \leq_{\mathrm{T}} B$. (3) If $[A, B] \in R$ then $A \leq_{*\mathrm{T}} B$.*

Theorem 2. *(Soundness of refinements) If r is a step refinement from A to B then $(A, B) \in r$.*

Combining Theorems 1 and 2 gives that $A \leq_{\mathrm{R}} B$ implies $A \leq_{\mathrm{T}} B$. In addition, Theorem 2 allows us to use refinement relations as a sound technique for proving implementation relations between live automata, as in [8, 19].

2.2 Normed Forward Simulations

A *normed forward simulation* from A to B consists of a relation f over $states(A) \times states(B)$ and a function $n : steps(A) \times states(B) \to S$, 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 $s \xrightarrow{a}_A s' \wedge u \in f[s]$ then
   - $u \in f[s'] \wedge a = \tau$, or
   - $\exists u' \in f[s'] : u \xrightarrow{a}_B u'$, or
   - $\exists u' \in f[s] : u \xrightarrow{\tau}_B u' \wedge n(s \xrightarrow{a}_A s', u') < n(s \xrightarrow{a}_A s', u)$.

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

The intuition behind this definition is that when $s \xrightarrow{a}_A s'$ and $(s, u) \in f$, either the transition in A is a stuttering step (first clause), or there is a matching step in B (second clause), or B can do a stuttering step which decreases the norm (third clause). Since the norm decreases at each application of the third clause, it 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 τ-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 [10], which only applies to convergent processes, the norm function is required to be of this restricted type. It is not hard to see that in the example of Figure 1, where B is divergent, the norm necessarily depends on the selected step in A.

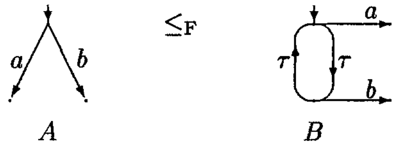


Fig. 1. Norm function must take steps of A into account.

As each step refinement is a normed forward simulation (for an arbitrary norm function) $A \leq_{\mathrm{R}} B$ implies $A \leq_{\mathrm{F}} B$. It is also not so difficult to prove that $\leq_{\mathrm{F}}$ is a preorder. The following theorem states that normed forward simulations induce the same preorder on automata as "branching forward simulations". Basically the same result has been obtained by Namjoshi [14] in the setting of stuttering bisimulations.

Theorem 3. *$A \leq_{\mathrm{F}} B$ iff there is a branching forward simulation from A to B, i.e., a relation f over $\mathit{states}(A) \times \mathit{states}(B)$ such that*

1. *If $s \in \mathit{start}(A)$ then $f[s] \cap \mathit{start}(B) \neq \emptyset$.*
2. *If $s \xrightarrow{a}_A s' \wedge u \in f[s]$ then*
   - *$u \in f[s'] \wedge a = \tau$, or*
   - *$\exists u_0, .., u_n \in f[s]\ \exists u' \in f[s'] : u_0 = u \wedge (\forall i < n : u_i \xrightarrow{\tau}_B u_{i+1}) \wedge u_n \xrightarrow{a}_B u'$.*

An interesting implication of our proof of Theorem 3 is that if there is a normed forward simulation from A to B, there is in fact a normed forward simulation with a norm function that has the set of natural numbers as its range.

The proofs of the following Theorems 4 and 5 are standard and similar to the proofs of the corresponding results in [13] and elsewhere.

Theorem 4. *(Soundness of forward simulations) If f is a normed forward simulation from A to B then $(A, B) \in f$.*

Theorem 5. *(Partial completeness of forward simulations) If B is deterministic and $A \leq_{*\mathrm{T}} B$, then $A \leq_{\mathrm{F}} B$.*

It is interesting to note that there is only one result from [13] that does not carry over to the setting of this paper. This result says that if A is a forest, i.e., each state can be reached via exactly one execution, and $A \leq_{\mathrm{F}} B$ then $A \leq_{\mathrm{R}} B$. The automata A and B of Figure 1 constitute a counterexample.

2.3 Normed Backward Simulations

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

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

Relation b is image-finite if, for all s, the set $b[s]$ is finite. 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 an image-finite normed backward simulation from A to B. It is routine to prove that $\leq_{\mathrm{B}}$ and $\leq_{\mathrm{iB}}$ are preorders, and to characterize these relations in terms of "branching backward simulations" as in Theorem 3.

The proofs of the following Proposition 1, Theorem 6 and Theorem 7 again closely follow the proofs of the corresponding results in [13].

Proposition 1. *(1) $A \leq_{\mathrm{R}} B \Rightarrow A \leq_{\mathrm{iB}} B$. (2) If all states of A are reachable, B is deterministic and $A \leq_{\mathrm{B}} B$, then $A \leq_{\mathrm{R}} B$. (3) $A \leq_{\mathrm{iB}} B \Rightarrow A \leq_{\mathrm{B}} B$. (4) If all states of A are reachable, B has finite invisible nondeterminism and $A \leq_{\mathrm{B}} B$, then $A \leq_{\mathrm{iB}} B$.*

Theorem 6. *(Soundness of backward simulations) If b is a normed backward simulation from A to B then $[A, B] \in b$. If, moreover, b is image-finite then $(A, B) \in b$.*

Theorem 7. *(Partial completeness of backward simulations) If A is a forest and $A \leq_{*\mathrm{T}} B$, then $A \leq_{\mathrm{B}} B$.*

2.4 History Relations

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

Thus $A \leq_{\mathrm{H}} B$ implies $A \leq_{\mathrm{F}} B$ and $B \leq_{\mathrm{R}} A$. Through these implications, the preorder and soundness results for forward simulations and refinements carry over to history relations. In fact, if (h, n) is a normed history relation from A to B then h^{-1} is just a functional *branching bisimulation* between A and B in the sense of Van Glabbeek and Weijland [9]. Hence, history relations preserve behavior of automata in a very strong sense.

The following theorem is a variant of a result proved by Sistla [17].

Theorem 8. *(Completeness of history relations and backward simulations) If $A \leq_{*\mathrm{T}} B$ then there exists a forest C such that $A \leq_{\mathrm{H}} C \leq_{\mathrm{B}} B$.*

Theorem 9. $A \leq_{\mathrm{F}} B \Leftrightarrow (\exists C : A \leq_{\mathrm{H}} C \leq_{\mathrm{R}} B)$.

2.5 Prophecy Relations

A pair (p, n) is a *normed prophecy relation* from A to B if (p, n) is a normed backward simulation from A to B and p^{-1} is a step refinement from B to A. Write $A \leq_{\mathrm{P}} B$ if there exists a prophecy relation from A to B, and $A \leq_{\mathrm{iP}} B$ if there is an image-finite prophecy relation from A to B. 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 $A \leq_{\mathrm{iP}} B$. Through these implications, the preorder and soundness results for backward simulations and refinements carry over to prophecy relations.

Theorem 10. *(1) $A \leq_{\mathrm{B}} B \Leftrightarrow (\exists C : A \leq_{\mathrm{P}} C \leq_{\mathrm{R}} B)$. (2) $A \leq_{\mathrm{iB}} B \Leftrightarrow (\exists C : A \leq_{\mathrm{iP}} C \leq_{\mathrm{R}} B)$.*

We can now state variants of the well-known completeness result of Abadi and Lamport [1].

Theorem 11. *(Completeness of history+prophecy relations and 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 finite invisible nondeterminism then $\exists C, D : A \leq_{\mathrm{H}} C \leq_{\mathrm{iP}} D \leq_{\mathrm{R}} B$.*

2.6 Decidability

Fix an assertion language $\mathcal{L}$ that includes first-order predicate logic and interpreted symbols for expressing the standard operations and relations. If automata A and B, and a conjectured simulation relation R and norm function n can all be expressed within a fragment of $\mathcal{L}$ for which tautology checking is decidable and if the transition relations of A and B can be specified using a finite number of deterministic transition predicates (as defined, for instance in, [7]), then it is decidable whether the pair (R, n) is a normed forward or normed backward simulation. It is not hard to prove that this result does not hold for the refinements, forward and backward simulations presented in [13], nor for the prophecy variables of [1].

2.7 Reachability

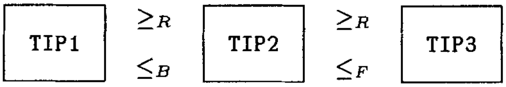
When proving simulations one often restricts the automata to the reachable subautomata, in order to be able to use invariants. In backward simulations this is not convenient, therefore a slightly adapted version of the backward simulation is presented below. The predicate Q on states of B can be used as induction hypothesis.

The adapted *normed backward simulation* from A to B consists of a total relation b over $states(A) \times states(B)$ and a function $n : (steps(A) \cup start(A)) \times states(B) \rightarrow S$, for some well-founded set S, such that

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

3 Example: IEEE 1394

In this section we illustrate the notions of step refinements and normed (forward and backward) simulations through the verification of a fragment of IEEE 1394 [12], a high performance serial multimedia bus protocol. The specific algorithm that we analyze is an abstract version of the tree identify phase (TIP) of the IEEE 1394. We present the TIP protocol at three levels of abstraction, and prove, via refinements and simulations, that these three specifications are trace equivalent. The three automata are described in the IOA language of [7], and the relations that will be established between them are depicted below.



IOA contains the basic type `Bool` with its standard operators, such as $\wedge$, $\vee$ and $\neg$. In addition type constructors `Array`, `Seq` (finite sequences) and `Set` (finite sets) are part of the language. The notation `_[_]` is used for array subscripting, an array with a value `e` in all cells is denoted by `const(e)`. The operation `_ ⊢_` appends an element at the end of a sequence.

The task of the TIP is to check whether the finite and connected network topology is cycle free, and (if this is indeed the case) to elect a leader amongst the nodes. In Figure 2, a simple example network is displayed, with *devices* `A`, `B` and `C`, and *ports* `p`, `q`, `r` and `s`. It is assumed that each port is connected to exactly one other port, which is called its *peer*. A network may contain a loop, and devices even can be connected to themselves. So, in the example port `q` also could have been connected to `r`, but then `q` and `r` could not have been connected to `p` and `s`, respectively.

```
automaton TIP1
  signature
    output root(d : Dev),
           loopdetect(d : Dev)
  states
    root, lpd: Array[Dev,Bool] := const(false)
  transitions
    output root(d)
      pre ¬∃ e: Dev (oncycle?(e) ∨ root[e])
      eff  root[d] := true
    output loopdetect(d)
      pre oncycle?(d) ∧ ¬ lpd[d]
      eff  lpd[d] := true
```

Fig. 3. Automaton TIP1.

In Figure 3, automaton TIP1 is presented. This simple automaton has two action schemas root(d:Dev) and loopdetect(d:Dev). Specification TIP1 says that if the network is cycle free exactly one node will perform a root action. Otherwise, no root action will occur, but instead each node that lies on a cycle will perform a loopdetect action.

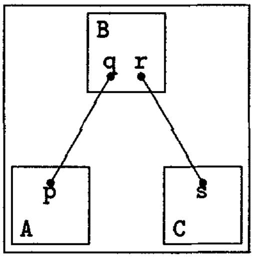


Fig. 2. A network

Automaton TIP2, presented in Figure 4, is an implementation of TIP1. The states contain an extra variable child: Set[Port]. If port p is in child then we say that its device dev(p) has a child, namely dev(peer(p)). When all but one neighbours of a device are its children it can become a child itself. Besides the loopdetect and root actions, TIP2 has an addchild action, which adds a port to the child set. If we consider the connections with a port in the child set to be the branches of a tree, then this tree grows with each addchild action from the leaves in the direction of the root. If all the ports of a device are in the child set then this device will become the root.

Automaton TIP3, presented in Figure 6, is an implementation of TIP2. It extends TIP2 with a state variable mq, which gives a queue of outgoing messages per port. Furthermore, some status bits per device (init, rc) are added. For a detailed description of the protocol we refer to [6] and the full version of this paper. Next the relations between the automata will be discussed.

(TIP2 $\leq_R$ TIP1) The function b from states of TIP2 to states of TIP1 is defined as the projection on the state variables of TIP1, in IOA notation: b([child,root,lpd]) == [root, lpd]. It is quite simple to prove that b is a step refinement (see Section 2.1) from TIP2 to TIP1.

The simulation from TIP1 to TIP2 illustrates the usefullness of backward simulations. A (traditional) forward simulation exists, but no *normed* forward

```
automaton TIP2
  signature
    internal addchild(d: Dev,p : Port),
    output  root(d: Dev),
            loopdetect(d: Dev)
  states
    child: Set[Port] := {},
    root,lpd: Array[Dev,Bool] := const(false)
  transitions
    internal addchild(d,p) where d = dev(p)
      pre  p ∉ child ∧
           ports(dev(peer(p))) - child = {peer(p)}
      eff  child := insert(p,child)
    output root(d)
      pre  ¬root[d] ∧ ports(d) ⊆ child
      eff  root[d] := true
    output loopdetect(d)
      pre  oncycle?(d) ∧ ¬lpd[d]
      eff  lpd[d] := true
```

Fig. 4. Automaton TIP2.

simulation. The reason for this can be seen in Figure 5. This figure depicts the transition systems of TIP1 and TIP2, for a network with only two devices (d and e) and a single link connecting these. The solid arrows represent the transitions of the systems, r is a shorthand for root and a for addchild. In this case TIP1 only can do a root(d) or a root(e) action. Before a root action can be done in TIP2 an addchild action has to be done.

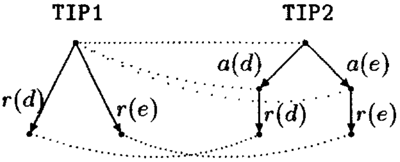


Fig. 5. Transition systems

A possible simulation relation is depicted by the dotted lines. This is not a normed *forward* simulation because the start state of TIP1 is related to a state where only a root(d) action can happen and not a root(e) action.

However the dotted lines depict a normed *backward* simulation from TIP1 to TIP2. The norm is defined on the states of TIP2 as the number of ports in the child set. Note that the only internal action of TIP2 is addchild, so the norm only needs to decrease when an addchild is simulated backwards (is 'undone').

In general, backward simulations can be useful when the implementation 'makes a decision' with internal steps. In TIP2 the decision who becomes root device is made by the internal action addchild.

(TIP2 $\leq_B$ TIP1) The inverse of function b is used as simulation relation. A predicate Q on the states of TIP2 is used to restrict the statespace.

$$\begin{aligned}\mathtt{Q(u)} = {} & \forall d.\mathtt{u.root[d]} \rightarrow \mathtt{ports(d)} \subseteq \mathtt{u.child} \wedge \\ & \forall d.\mathtt{u.lpd[d]} \rightarrow \mathtt{oncycle?(d)} \wedge \\ & \mathtt{GDT(u.child)}\end{aligned}$$

The first two conjucts are trivial consequences of the specification. GDT(S) is a predicate on ports stating that the net obtained by deleting all links without a port in the set S is a Growing Directed Tree. This means that it contains no cycles and each device has at most one parent and when a device has a parent all its other neighbours are its children. The norm function only depends on the state of TIP2, it is defined as the cardinalty of the set child.

(TIP3 $\leq_R$ TIP2) The function f from states of TIP3 to states of TIP2 is the projection on the state variables of TIP2. In IOA notation :
f([child,mq,init,rc,root,lpd]) == [child,root,lpd]

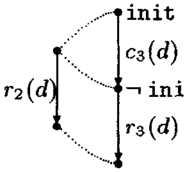
In the routine proof of TIP3 $\leq_R$ TIP2 an invariant I is proved at the same time, where I == oncycle?(d) --> init(d).

(TIP2 $\leq_F$ TIP3) The proof that $\mathtt{f}^{-1}$ is a normed forward simulation from TIP2 to TIP3 will be discussed in more detail. The condition for start states holds trivially. Next the three actions of TIP2 must be simulated, they will be discussed each. The norm function is defined per action schema and the result type is the natural numbers with the usual ordering. For convenience actions of TIPx are subscripted with x. The states s and t are states of TIP2 before and after a transition respectively, similarly u and v are states of TIP3.

The $\mathtt{loopdetect}_2$ action of TIP2 has the same precondition as the action $\mathtt{loopdetect}_3$ of TIP3, and mentions only state variables that the automata have in common. Thus if the precondition of $\mathtt{loopdetect}_2$ holds on a state s then the precondition of $\mathtt{loopdetect}_3$ also holds on states in $\mathtt{f}^{-1}\mathtt{(s)}$. Because the $\mathtt{loopdetect}_2$ action can be simulated directly, the norm function for $\mathtt{loopdetect}_2$ is irrelevant.

The precondition of the the $\mathtt{root}_2$ action is similar to the precondition of $\mathtt{root}_3$, the latter has only a single extra conjunct: ¬ init [d]. The norm function for $\mathtt{root(d)}_2$ is defined to be 1 when init[d] holds and 0 otherwise.

If $\mathtt{root(d)}_2$ is enabled and f(u) = s then $\mathtt{root(d)}_3$ or $\mathtt{childrenknown(d)}_3$ are enabled in u. A case distinction on u.init[d] is made. Suppose u.init[d] holds then the action $\mathtt{childrenknown(d)}_3$ is enabled. This action reduces the norm, and the state after this action, is also related to s. Suppose ¬ u.init[d] then $\mathtt{root(d)}_3$ is enabled in u and $\mathtt{root(d)}_3$ can be simulated directly.



The proof that $\mathtt{addchild}_2$ is simulated is similar to the proof for $\mathtt{root}_2$ but longer. Where the $\mathtt{root}_2$ simulation had two cases, we have seven cases for $\mathtt{addchild}_2$. The case distinction is on whether init holds or not and whether the message queues of the port that is added and its peer are empty or contain a parent request.

Notice that the relations used in the simulations are the inverses of the functions (b and f) used in the step refinements, so in fact we have proved a stronger result, namely: TIP1 $\leq_{iP}$ TIP2 and TIP2 $\leq_{H}$ TIP3.

A drawback of the use of normed simulations instead of 'traditional' simulations could be that one has to find a suitable norm function. In our experience the norm functions were obvious. We expect this to be the case in general, because the norm function is 'local', only for a specific transition in one automaton the internal step in the other should decrease this measure.

References

1. M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.
2. S. Bensalem, Y. Lakhnech, and H. Saidi. Powerful techniques for the automatic generation of invariants. In *Proc. CAV'96*, LNCS 1102, pp 323–335. Springer, 1996.
3. M. Browne, E. Clarke, and O. Grümberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Comp. Sci.*, 59(1,2):115–131, 1988.
4. R. De Nicola and F. Vaandrager. Three logics for branching bisimulation. *Journal of the ACM*, 42(2):458–487, 1995.
5. M. Devillers, W. Griffioen, and O. Müller. Possibly infinite sequences: A comparative case study. In *Proc. TPHOLs'97*, LNCS 1275, pp 89–104. Springer, 1997.
6. M. Devillers, W. Griffioen, J. Romijn, and F. Vaandrager. Verification of a leader election protocol — formal methods applied to IEEE 1394. Technical Report CSI-R9728, University of Nijmegen, 1997.
7. S. Garland, N. Lynch, and M. Vaziri. IOA: A language for specifiying, programming, and validating distributed systems, September 1997. Available through http://larch.lcs.mit.edu:8001/~garland/ioaLanguage.html.
8. R. Gawlick, R. Segala, J. Søgaard-Andersen, and N. Lynch. Liveness in timed and untimed systems. In *Proc.* 21^{th} *ICALP*, LNCS 820. Springer, 1994. A full version appears as MIT Technical Report MIT/LCS/TR-587.
9. R. van Glabbeek and W. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996.
10. J. Groote and J. Springintveld. Focus points and convergent process operators — a proof strategy for protocol verification. Report CS-R9566, CWI, 1995.
11. L. Helmink, M. Sellink, and F. Vaandrager. Proof-checking a data link protocol. In *Proc. TYPES'93*, LNCS 806, pp 127–165. Springer, 1994.
12. IEEE Computer Society. IEEE Standard for a High Performance Serial Bus. Std 1394-1995, August 1996.
13. N. Lynch and F. Vaandrager. Forward and backward simulations, I: Untimed systems. *Information and Computation*, 121(2):214–233, 1995.
14. K. Namjoshi. A simple characterization of stuttering bisimulation. In *Proc. FST & TCS'97*, LNCS 1346, pp 284–296. Springer, 1997.
15. T. Nipkow and K. Slind. I/O automata in Isabelle/HOL. In *Types for Proofs and Programs*, LNCS 996, pp 101–119. Springer, 1995.
16. S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, 1995.
17. A. Sistla. Proving correctness with respect to nondeterministic safety specifications. *Information Processing Letters*, 39(1):45–49, 1991.

18. J. Søgaard-Andersen, S. Garland, J. Guttag, N. Lynch, and A. Pogosyants. Computer-assisted simulation proofs. In *Proc. CAV'93*, LNCS 697, pp 305–319. Springer, 1993.
19. J. Søgaard-Andersen, N. Lynch, and B. Lampson. Correctness of communication protocols – a case study. Report MIT/LCS/TR-589, MIT, Cambridge, MA, 1993.

```
automaton TIP3
  signature

  states
    child: Set[Port] := {}
    mq: Array[Port,Seq[Mes]] := const({})
    init: Array[Dev,Bool] := const(true)
    rc, root, lpd: Array[Dev,Bool] := const(false)
  transitions
    internal childrenknown(d)
      pre init[d] ∧ size(ports(d) - child) ≤ 1
      eff init[d] := false;
          for p in ports[d] do if p ∈ child
                                 then mq[p] := mq[p] ⊢ ack
                                 else mq[p] := mq[p] ⊢ parent fi od
    internal addchild(d,p) where d = dev(p)
      pre init[d] ∧ head(mq[peer(p)]) = parent
      eff child := insert(p, child); mq[peer(p)] := tail(mq[peer(p)])
    internal receivemes(d,p,mes) where d = dev(p)
      pre ¬init[d] ∧ ports(d) - child = {p} ∧ head(mq[peer(p)]) = mes
      eff if mes = parent then rc[d] := true fi;
          mq[peer(p)] := tail(mq[peer(p)])
    internal solverootcontent(d,p) where d = dev(p)
      pre rc(d) ∧ rc(dev(peer(p)))
      eff child := insert(p,child);
          rc(d) := false; rc(dev(peer(p))) := false
    output root(d)
      pre ¬init[d] ∧ ¬root[d] ∧ ports(d) ⊆ child
      eff root[d] := true
    output loopdetect(d)
      pre oncycle?(d) ∧ ¬lpd[d]
      eff lpd[d] := true
```

Fig. 6. Automaton TIP3.