An Assume-Guarantee Rule For Checking Simulation

T.A. Henzinger S. Qadeer S.K. Rajamani S. Taşıran
EECS Department, University of California at Berkeley, CA 94720

Abstract

The simulation preorder on state transition systems is widely accepted as a useful notion of refinement, both in its own right and as an efficiently checkable sufficient condition for trace containment. For composite systems, due to the exponential explosion of the state space, there is a need for decomposing a simulation check of the form $P \preceq_s Q$ into simpler simulation checks on the components of P and Q. We present an assume-guarantee rule that enables such a decomposition. To the best of our knowledge, this is the first assume-guarantee rule that applies to a refinement relation different from trace containment. Our rule is circular, and its soundness proof requires induction on trace-trees. The proof is constructive: given simulation relations that witness the simulation preorder between components, we provide a procedure for constructing a witness relation for $P \preceq_s Q$. We also extend our assume-guarantee rule to account for fairness assumptions on transition systems.

1 Introduction

In hierarchical verification, we need to check proof obligations of the form $P \preceq Q$, where P and Q are system descriptions and $\preceq$ is a pre-order on system descriptions. The assertion $P \preceq Q$ holds if P describes the same system as Q, but possibly on a finer level of detail (or equivalently, Q describes the same system as P but possibly on a coarser level of abstraction). For example, P may be an RTL-level description of a pipelined processor, and Q may be an ISA description of the same processor. The assertion $P \preceq Q$ is therefore variously pronounced as "P implements Q", or "P refines Q", or "Q specifies P," or "Q abstracts P."

Mathematically, a popular choice for the pre-order $\preceq$ is *trace containment*. In this case, $P \preceq Q$ asserts that every sequence of inputs and outputs that is possible for P is also possible for Q (at the same time, the more abstract specification Q may allow some traces that are not realized by the more concrete implementation P). While simple and intuitive, trace containment has several shortcomings. First, it is a practical impossibility to check trace containment automatically for all but the smallest examples, because the check is exponential in the number of states of Q if Q is non-deterministic. Second, in top-down design, if system description P fleshes out detail that is left open in system description Q, there is a much tighter relation between P and Q than trace containment would indicate; namely, each implementation state of P corresponds to a specification state of Q. This tighter relation is captured mathematically by the notion of a *simulation relation*. Intuitively, Q simulates P iff, starting from the initial states and continuing ad infinitum, every input-output pair of P can be matched by the same input-output pair in Q [Mil71]. Clearly, if Q simulates P, then every trace of P is also a trace of Q. The converse is not true; that is, simulation is a stronger requirement than trace containment. However, it has been said that trace containment without simulation is more often than not due to coincidence rather than systematic design [Kur94].

While trace containment is defined *globally*, for input-output sequences of arbitrary length, simulation is defined *locally*, by considering individual input-output pairs for all states. It is this locality in the definition of simulation that leads to significant advantages. First, if Q is claimed to simulate P, then a witness to this claim can be produced in the form of a relation between states of P and states of Q, and the witness can be efficiently checked for correctness (the check is linear in the number of states of both P and Q). Such witness relations are widely used in verification tools, under various names like homomorphisms [Kur94] and refinement mappings [AL88, Lyn96]. Second, even if no witness is available, the existence of a simulation can be checked in polynomial time (the check is quadratic in the number of states of P and Q). The number of states, however, depends exponentially on the size of the system description (note that n boolean variables give rise to 2^n states—this is the *state-explosion problem*). Thus, even algorithms that are linear in the number of states are often infeasible in practice, and techniques have been studied for dividing a given verification task into simpler subtasks.

Compositional techniques for dividing the verification task $P \preceq Q$ into simpler subtasks are guided by the structures of P and Q. If the refinement relation $\preceq$ is interpreted as trace containment, a number of compositional techniques are known. Specifically, if $P = P_1 \| P_2$ and $Q = Q_1 \| Q_2$, then in order to check $P \preceq Q$, it suffices to check both $P_1 \preceq Q_1$ and $P_2 \preceq Q_2$. This *compositional principle* for trace containment is valid whenever parallel composition corresponds to trace intersection (replace $\|$ by conjunction, and $\preceq$ by implication). Unfortunately, the compositional principle is often not helpful, because P_1 typically refines Q_1 only when constrained by an environment that behaves like P_2, and similarly, P_2 may refine Q_2 only when constrained by an environment that behaves like P_1. Under certain modeling assumptions (namely, non-blocking and finite nondeterminism), the compositional principle can be strengthened to an *assume-guarantee principle* [Sta85, CLM89, GL94, AL95, AH96, McM97]: in order to check $P \preceq Q$, it

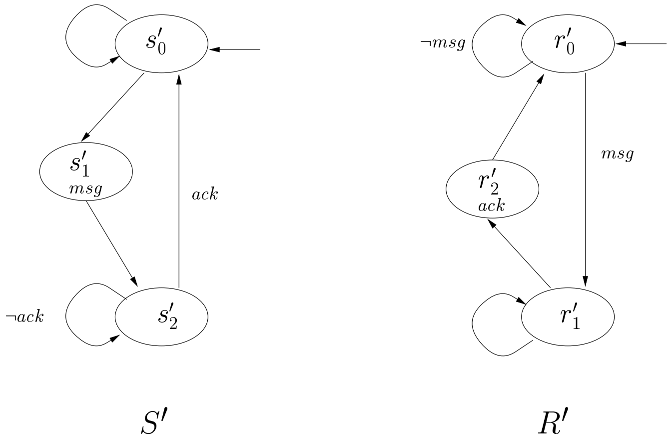

Figure 1: Specifications of sender and receiver

suffices to check both $P_1 \| Q_2 \preceq Q_1$ and $Q_1 \| P_2 \preceq Q_2$. Three observations about this proof rule are important. First, the rule addresses the issue that the environment of P_1 may have to be suitably constrained in order to implement Q_1, and similarly for P_2. Second, the rule avoids reasoning about the compound implementation $P_1 \| P_2$, which typically has the largest of the involved state spaces. Third, unlike the compositional principle, the assume-guarantee principle is circular and therefore not propositionally valid—its proof requires induction on the length of traces.

By contrast to the case of trace containment, if the refinement relation $\preceq$ is interpreted as "is simulated by," then little is known about compositional techniques other than the fact that the compositional principle remains valid whenever parallel composition corresponds to the intersection of trees whose branches are traces (this is because Q simulates P iff every trace tree of P is also a trace tree of Q). In particular, it would be useful to have an assume-guarantee principle for simulation, which, given witnesses for the two subtasks $P_1 \| Q_2 \preceq Q_1$ and $Q_1 \| P_2 \preceq Q_2$, lets us construct a witness for $P \preceq Q$. In this paper, we show that under the same modeling assumptions under which the assume-guarantee principle is sound for trace containment, it is also sound for simulation. Second, we show how the compound witness can be constructed from the witnesses for the subtasks. Third, we show that in analogy to the case of trace containment, the assume-guarantee principle for simulation can be extended to account for fairness assumptions in system descriptions.

We illustrate the assume-guarantee rule for simulation using an example. Figure 1 shows the specifications for a sender S' and a receiver R' in a communication protocol. Each state is labeled with outputs that are true in that state, and each arc

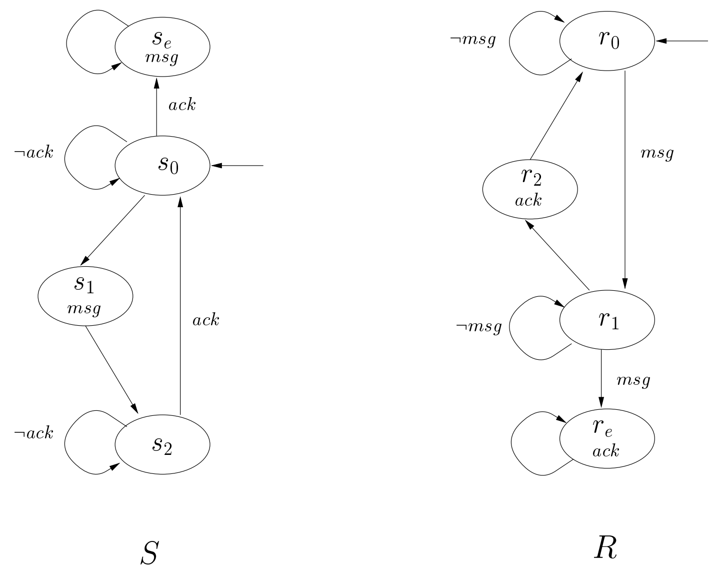

Figure 2: Implementations of sender and receiver

is labeled with conditions on inputs that need to be satisfied for the arc to be taken. A state with no label means that no output propositions are true in that state, and an arc with no label means that there is no condition on input propositions to take that transition. The initial states s_0' of the sender and r_0' receiver are marked using an arrow. The sender has one output proposition, namely msg, which is true whenever a message is produced, and one input proposition, namely ack, which is used to acknowledge the receipt of a message by the receiver. The receiver has ack as its only output proposition and msg as its only input proposition. The sender starts off at s_0' and stays there for an arbitrary number of steps. It nondeterministically produces a message by moving to s_1'. Once in s_1', it is forced to move to s_2' in one step. Then, the sender waits in s_2' until it receives an ack. On receiving the ack it goes back to s_0'. The receiver starts at r_0' and moves to r_1' on seeing a msg. It stays in r_1' for an arbitrary number of steps and nondeterministically moves to r_2'. An ack is output at r_2' after which it moves back to r_0' in the next step. Figure 2 shows the implementations of the sender S and receiver R. If S receives an ack while at s_0, it goes to an "error-state" s_e, from which it keeps sending messages in every step. Such a behavior is not allowed by the specification S'. Thus, S' does not simulate S. However, after composing with the specification of the receiver, it is seen that $S \parallel R' \preceq_s S'$, since the acknowledgments can never be received by S while at s_0. The relation $\Theta_S = \{(\langle s_0, r_0' \rangle, s_0'), (\langle s_1, r_0' \rangle, s_1'), (\langle s_2, r_1' \rangle, s_2'), (\langle s_2, r_2' \rangle, s_2')\}$

is the witness to this simulation. Similarly, R' does not simulate R, but $S' \parallel R \preceq_s R'$ with the relation $\Theta_R = \{(\langle s'_0, r_0\rangle, r'_0), (\langle s'_1, r_0\rangle, r'_0), (\langle s'_2, r_1\rangle, r'_1), (\langle s'_2, r_2\rangle, r'_2)\}$ as the witness. In Section 3, we prove that the existence of simulation relations from $S \parallel R'$ to S' and from $S' \parallel R$ to R' is sufficient to conclude the existence of a simulation relation Θ from $S \parallel R$ to $S' \parallel R'$, and give a procedure to construct Θ, given Θ_S and Θ_R. For our example, the witness simulation relation $\Theta = \{(\langle s_0, r_0\rangle, \langle s'_0, r'_0\rangle)$, $(\langle s_1, r_0\rangle, \langle s'_1, r'_0\rangle)$, $(\langle s_2, r_1\rangle, \langle s'_2, r'_1\rangle), (\langle s_2, r_2\rangle, \langle s'_2, r'_2\rangle)\}$.

For simplicity we use Moore machines to model systems. The results apply to other non-blocking, finitely non-deterministic models such as Reactive Modules [AH96]. Section 2 defines Moore machines and establishes the connection between trace-tree containment and simulation. In Section 3 we prove the validity of the assume-guarantee rule for the simulation preorder and describe how the compound witness simulation relation is constructed from the witnesses for the components. Section 4 defines simulation on fair Moore machines. The assume-guarantee rule for fair simulation is proven in Section 5.

2 Simulation Relations on Moore Machines

Moore machines. A Moore machine is a tuple $P = \langle S^P, s^P, I^P, O^P, L^P, R^P\rangle$ where

- S^P is the set of states,
- $s^P \in S^P$ is the initial state,
- I^P is the set of input propositions,
- O^P is the set of output propositions disjoint from I^P,
- $L^P : S^P \to \mathcal{P}(O^P)$ is a function that labels each state with the subset of output propositions true in that state, and
- $R^P \subseteq S^P \times \mathcal{P}(I^P) \times S^P$ is the transition relation. We write $R^P(s, i, s')$ as shorthand for $(s, i, s') \in R^P$.

We restrict our attention to Moore machines P satisfying the following two properties:

1. *Non-blocking*: For all $s \in S^P$ and $i \subseteq I^P$, there exists a state t such that $R^P(s, i, t)$.
2. *Finite non-determinism*: For all $s \in S^P$, $i \subseteq I^P$ and $o \subseteq O^P$, there are at most a finite number of states t such that $R^P(s, i, t)$ and $L^P(t) = o$.

Run-trees and trace-trees. A (finite or infinite) *tree* is a set $t \subseteq \mathbb{N}^*$ such that if $xn \in t$, for $x \in \mathbb{N}^*$ and $n \in \mathbb{N}$, then $x \in t$ and $xm \in t$ for all $0 \leq m < n$. The elements of t represent nodes: the empty word ϵ is the root of t, and for each node x, the nodes of the form xn, for $n \in \mathbb{N}$, are the children of x. The number of children of node x is denoted by $deg(x)$. A tree t is finite if t is a finite set. The *depth* of a node $t \in x$ is defined inductively as follows: (1) the depth of ϵ is 0, and (2) if the depth of $x \in t$ is d then the depth of xn is $d+1$. If t is finite, then the depth of t is defined as the maximum of depths over all nodes of t. The nodes of t with no children are called *leaves* of t. A *path* ρ of t is a finite or infinite set $\rho \subseteq t$ of nodes that is generated by the following three rules: (1) $\epsilon \in \rho$, (2) for each node $x \in \rho$, there exists at most one $n \in \mathbb{N}$ with $xn \in \rho$, and (3) if $xn \in \rho$, then $x \in \rho$. Given a pair of sets A and B, an $\langle A, B\rangle$*-labeled tree* is a triple $\langle t, \lambda, \delta\rangle$, where t is a tree, $\lambda : t \to A$ is a node labeling function that maps each node of t to an element in A, and $\delta : (t \times t) \to B$ is an edge labeling function that maps each edge $\langle x, xn\rangle$ of t to an element in B. Then, every path $\rho = \{\epsilon, n_0, n_0n_1, \ldots\}$ of t generates a sequence $\Gamma(\rho) = (\lambda(\epsilon)) \cdot (\delta(\epsilon, n_0), \lambda(n_0)) \cdot (\delta(n_0, n_0n_1), \lambda(n_0n_1)) \cdots$ in $A \times (B \times A)^* \cup A \times (B \times A)^\omega$.

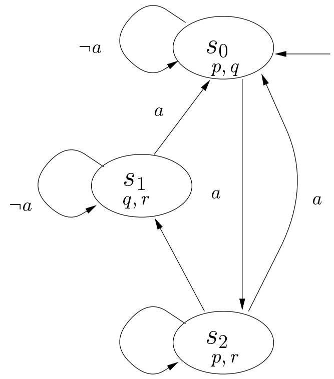

Figure 3: A simple Moore machine

An *s-run-tree* T of P is a $\langle S^P, \mathcal{P}(I^P)\rangle$-labeled tree $\langle t, \lambda, \delta\rangle$ such that $\lambda(\epsilon) = s$, and for all edges $\langle x, xn\rangle$ we have $R^P(\lambda(x), \delta(x, xn), \lambda(xn))$. A run-tree of P is an s^P-run-tree for initial state s^P. A *trace-tree* $T' = \langle t, \lambda', \delta\rangle$ of P is a $\langle \mathcal{P}(O^P), \mathcal{P}(I^P)\rangle$-labeled tree such that there is a run tree of $T = \langle t, \lambda, \delta\rangle$ of P, and for every $x \in t$ we have $L^P(\lambda(x)) = \lambda'(x)$. For brevity we say $T' = L^P(T)$, and call T as the *witness* to T'. See Figures 3 and 4 for an example.

Tree containment. Consider Moore machines $P = \langle S^P, s^P, I^P, O^P, L^P, R^P\rangle$ and $Q = \langle S^Q, s^Q, I^Q, O^Q, L^Q, R^Q\rangle$. We say that Q is refinable by P if (1) $O^Q \subseteq O^P$,

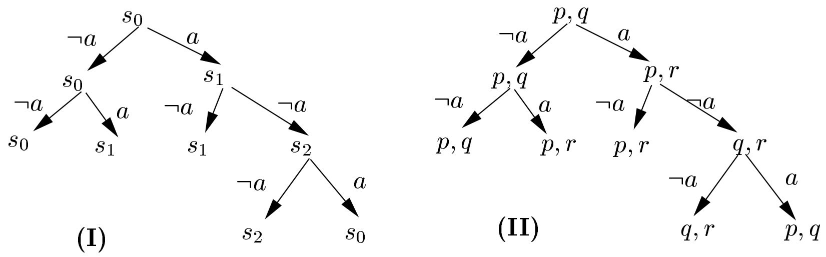

Figure 4: (I) is an s_0 run-tree for the Moore machine in Figure 3 and (II) is the corresponding trace-tree

and (2) $I^Q \subseteq I^P \cup O^P$.

Suppose Q is refinable by P. Let $T = \langle t, \lambda, \delta \rangle$ be a trace-tree of P. We say that the *projection* of T on Q is a trace-tree $[T]_Q = \langle t, \lambda', \delta' \rangle$ of Q, such that for all $x \in t$ we have $\lambda'(x) = \lambda(x) \cap O^Q$ and for all $x, xn \in t$, we have $\delta'(x, xn) = (\delta(x, xn) \cup \lambda(x)) \cap I^Q$. We say that Q *tree-contains* P if (1) Q is refinable by P, and (2) For every trace-tree T of P its projection $[T]_Q$ is a trace-tree of Q.

Composition. The composition of P and Q denoted by $P \parallel Q$ exists if $O^P \cap O^Q = \emptyset$ and is defined to be the Moore machine $K = \langle S^K, s^K, I^K, O^K, L^K, R^K \rangle$ where

1. $S^K = S^P \times S^Q$,
2. $s^K = \langle s^P, s^Q \rangle$,
3. $I^K = (I^P \cup I^Q) \setminus (O^P \cup O^Q)$,
4. $O^K = O^P \cup O^Q$,
5. $L^K(\langle p, q \rangle) = L^P(p) \cup L^Q(q)$ for all $\langle p, q \rangle \in S^K$.
6. $R^K(\langle p_1, q_1 \rangle, i, \langle p_2, q_2 \rangle)$ iff $R^P(p_1, (i \cup L^Q(q_1)) \cap I^P, p_2)$ and $R^Q(q_1, (i \cup L^P(p_1)) \cap I^Q, q_2)$ for all $\langle p_1, p_2 \rangle, \langle q_1, q_2 \rangle \in S^K$ and $i \in I^K$.

Simulation. Consider Moore machines P and Q such that Q is refinable by P. A relation $\Theta \subseteq S^P \times S^Q$ is said to be a *simulation relation* from P to Q if and only if the following hold:

1. $(s^P, s^Q) \in \Theta$.
2. For all $(p, q) \in \Theta$, we have $L^P(p) \cap O^Q = L^Q(q)$.
3. For all $(p, q) \in \Theta$ and for all i and $\tilde{p}$ such that $R^P(p, i, \tilde{p})$, there exists $\tilde{q}$ such that $R^Q(q, (i \cup L^P(p)) \cap I^Q, \tilde{q})$ and $(\tilde{p}, \tilde{q}) \in \Theta$.

If such Θ exists, Q is said to *simulate* P (written as $P \preceq_s Q$) with Θ as the *witnessing* simulation relation.

Proposition 1 *Consider any two Moore machines P and Q. Then $P \preceq_s Q$ iff Q tree-contains P.*

3 Assume-Guarantee Rule for Simulation

Theorem 2 *Let P, Q, P', Q' be Moore machines such that $P \parallel Q$ and $P' \parallel Q'$ exist. Suppose $P \parallel Q' \preceq_s P'$ and $P' \parallel Q \preceq_s Q'$ with witnessing simulation relations Θ_P and Θ_Q respectively, and every input of $P' \parallel Q'$ is either an input or output of $P \parallel Q$. Then, we can construct a simulation Ω from $P \parallel Q$ to $P' \parallel Q'$.*

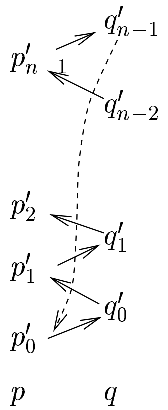

Figure 5: Figure demonstrating the definition of Ω.

Proof: Let $\Omega \subseteq (S^P \times S^Q) \times (S^{P'} \times S^{Q'})$ be defined as as follows. $(\langle p, q\rangle, \langle p', q'\rangle) \in \Omega$ iff there exist $p'_0, p'_1, ..., p'_n \in S^{P'}$ and $q'_0, q'_1, ..., q'_{n-1} \in S^{Q'}$ such that (See Figure 5)

- $p'_0 = p'_n = p'$ and $q'_0 = q'$, and
- for all $i < n$, we have that $(\langle p'_i, q\rangle, q'_i) \in \Theta_Q$, and $(\langle p, q'_i\rangle, p'_{i+1}) \in \Theta_P$.

We need to show that the conditions 1-3 of the definition of a simulation relation are satisfied by Θ. In the following (for 2 and 3), assume that $(\langle p, q\rangle, \langle p', q'\rangle) \in \Omega$, i.e., there exist $p'_0, p'_1, ..., p'_n \in S^{P'}$ and $q'_0, q'_1, ..., q'_{n-1} \in S^{Q'}$ satisfying the conditions mentioned above.

1. Since Θ_P and Θ_Q are simulation relations, we have that $(\langle s^P, s^{Q'}\rangle, s^{P'}) \in \Theta_P$ and $(\langle s^{P'}, s^Q\rangle, s^{Q'}) \in \Theta_Q$. Let $n = 1$, $p'_0 = p'_1 = s^{P'}$ and $q'_0 = s^{Q'}$. Then, we have that $(\langle s^P, s^Q\rangle, \langle s^{P'}, s^{Q'}\rangle) \in \Omega$.

2. Since $(\langle p'_0, q\rangle, q'_0) \in \Theta_Q$ and Θ_Q is a simulation relation, $L^Q(q) \cap O^{Q'} = L^{Q'}(q'_0)$. Since $q'_0 = q'$, we have $L^Q(q) \cap O^{Q'} = L^{Q'}(q')$. Also, $(\langle p, q'_i\rangle, p'_{i+1}) \in \Theta_P$ for all i from 0 to n. Therefore, $L^P(p) \cap O^{P'} = L^{P'}(p'_{i+1})$ for all i from 0 to n. Since, $p'_{n+1} = p'_0 = p'$, we have $L^P(p) \cap O^{P'} = L^{P'}(p')$. Hence, we get $L^{P \parallel Q}(\langle p, q\rangle) \cap O^{P' \parallel Q'} = L^{P' \parallel Q'}(\langle p', q'\rangle)$.

3. Let $\langle p, q\rangle \xrightarrow{i} \langle \tilde{p}, \tilde{q}\rangle$ for some $\langle \tilde{p}, \tilde{q}\rangle$ and $i \in \mathcal{P}(I^{P \parallel Q})$. Define $i' = i \cup L^P(p) \cup L^Q(q)$. Clearly, i' includes all inputs of the machines P, Q, P' and Q'. In the following, whenever we use i' as the input for some Moore machine K, the intention is to take the projection $i' \cap I^K$ onto the set of inputs of K. We want to show that there exists $(\langle \tilde{p}, \tilde{q}\rangle, \langle \tilde{p}', \tilde{q}'\rangle) \in \Omega$ such that $R^{P' \parallel Q'}(\langle p', q'\rangle, i' \cap I^{P' \parallel Q'}, \langle \tilde{p}', \tilde{q}'\rangle)$.

   - Since all machines are non-blocking, we have $R^{P'}(p'_0, i', \tilde{p}'_0)$ for some $\tilde{p}'_0$. Hence, $R^{P' \parallel Q}(\langle p'_0, q\rangle, i', \langle \tilde{p}'_0, \tilde{q}\rangle)$.
   - The fact that $(\langle p'_0, q\rangle, q'_0) \in \Theta_Q$ implies that there exists some $\tilde{q}'_0$ such that $R^{Q'}(q'_0, i', \tilde{q}'_0)$ and $(\langle \tilde{p}'_0, \tilde{q}\rangle, \tilde{q}'_0) \in \Theta_Q$.
   - $R^{Q'}(q'_0, i', \tilde{q}'_0)$ implies that $R^{P \parallel Q'}(\langle p, q'_0\rangle, i', \langle \tilde{p}, \tilde{q}'_0\rangle)$.
   - $(\langle p, q'_0\rangle, p'_1) \in \Theta_P$, therefore, there exists $\tilde{p}'_1$ such that $R^{P'}(p'_1, i', \tilde{p}'_1)$ and $(\langle \tilde{p}, \tilde{q}'_0\rangle, \tilde{p}'_1) \in \Theta_P$

   Repeating this process, we obtain $\tilde{p}'_0, \tilde{p}'_1, \tilde{p}'_2, \ldots$, and $\tilde{q}'_0, \tilde{q}'_1, \tilde{q}'_2, \ldots$ such that

   - for all $k \in \mathbb{N}$, we have $R^{P'}(p'_0, i', \tilde{p}'_{kn})$ and $R^{Q'}(q'_0, i', \tilde{q}'_{kn})$, and
   - for all i, we have that $(\langle \tilde{p}'_i, \tilde{q}\rangle, \tilde{q}'_i) \in \Theta_Q$ and $(\langle \tilde{p}, \tilde{q}'_i\rangle, \tilde{p}'_{i+1}) \in \Theta_P$.

   Since P' has the finite nondeterminism property, there exist $a, b \in \mathbb{N}$ with $b > a$ such that $\tilde{p}'_{an} = \tilde{p}'_{bn}$. Consider the states $\tilde{p}'_{an}, \tilde{p}'_{an+1}, \ldots, \tilde{p}'_{bn} \in S^{P'}$ and $\tilde{q}'_{an}, \tilde{q}'_{an+1}, \ldots, \tilde{q}'_{bn-1} \in S^{Q'}$. We know that for all $an \leq i < bn$, $(\langle \tilde{p}'_i, \tilde{q}\rangle, \tilde{q}'_i) \in \Theta_Q$ and $(\langle \tilde{p}, \tilde{q}'_i\rangle, \tilde{p}'_{i+1}) \in \Theta_P$. It follows from the definition of Ω that $(\langle \tilde{p}, \tilde{q}\rangle, \langle \tilde{p}'_{an}, \tilde{q}'_{an}\rangle) \in \Omega$. We also have that $R^{P' \parallel Q'}(\langle p', q'\rangle, i' \cap I^{P' \parallel Q'}, \langle \tilde{p}'_{an}, \tilde{q}'_{an}\rangle)$.

■

4 Simulation Relations on Fair Moore Machines

Fair Moore machines. Consider a Moore machine $P = \langle S^P, s^P, I^P, O^P, L^P, R^P\rangle$. An *s-run* of P is a finite or infinite sequence $\overline{s} = s_0 \xrightarrow{i_1} s_1 \xrightarrow{i_2} s_2 \xrightarrow{i_3} s_3 \ldots$ such that $s_0 = s$ and $R^P(s_n, i_{n+1}, s_{n+1})$ for all $n \geq 0$. The set of all finite s-runs of P is denoted by Σ^P. Note that Σ^P contains the empty sequence also. Let $\overline{s} = s_0 \xrightarrow{i_1} s_1 \xrightarrow{i_2} s_2 \xrightarrow{i_3} \ldots \xrightarrow{i_{n-1}} s_{n-1}$. Then the length of $\overline{s}$, denoted by $|\overline{s}|$, is n, and the kth prefix for $0 \leq k \leq n$, denoted by $\overline{s}_k$, is $s_0 \xrightarrow{i_1} s_1 \xrightarrow{i_2} s_2 \xrightarrow{i_3} \ldots \xrightarrow{i_{k-1}} s_{k-1}$. If $\overline{s}$ is infinite, then $|\overline{s}|$ is defined to be ω. If $\overline{s}$ is a run of P, we write $\inf(\overline{s})$ for the (finite) set of states that are repeated infinitely often in $\overline{s}$. A *fairness constraint* for P is a function that maps every run of P to the binary set $\{\mathit{fair}, \mathit{unfair}\}$. Specifically, we use *Streett* constraints to express fairness. A Streett constraint F is specified by a set $F_S \subseteq 2^{S^P} \times 2^{S^P}$ of pairs of state sets. Then, for a run $\overline{s}$ of P we have $F(\overline{s}) = \mathit{fair}$ iff for every pair $\langle l, r\rangle \in F_S$, if $\inf(\overline{s}) \cap l \neq \emptyset$ then $\inf(\overline{s}) \cap r \neq \emptyset$.

A *fair Moore machine* $\mathcal{P} = \langle P, F^P\rangle$ consists of a Moore machine P and a fairness constraint F^P for P. For a state $s \in S^P$, a *fair* s-run of $\mathcal{P}$ is either a finite s-run of P or an infinite s-run $\overline{s}$ of P such that $F^P(\overline{s}) = \mathit{fair}$. A fair run of $\mathcal{P}$ is a fair s^P-run, for the initial state s^P. A *fair run-tree* of $\mathcal{P}$ is a run-tree $\langle t, \lambda, \delta\rangle$ of P such that every path of t generates a fair run of $\mathcal{P}$. A *fair trace-tree* of $\mathcal{P}$ is a trace-tree of P that is witnessed by a fair run-tree of $\mathcal{P}$. In the following, we consider fair Moore machines, $\mathcal{P} = \langle P, F^P\rangle$ and $\mathcal{Q} = \langle Q, F^Q\rangle$.

Composition. The composition of $\mathcal{P}$ and $\mathcal{Q}$ denoted by $\mathcal{P} \parallel \mathcal{Q}$ exists if $P \parallel Q$ exists, and is defined to be the fair Moore machine $\mathcal{K} = \langle K, F^K\rangle$, where

- $K = P \parallel Q$.
- $F^K = \{\langle l \times S^Q, r \times S^Q\rangle | \langle l, r\rangle \in F^P\} \cup \{\langle S^P \times l, S^P \times r\rangle | \langle l, r\rangle \in F^Q\}$

Intuitively, a run $\overline{s}$ of $\mathcal{K}$ is *fair* if the "P component" of $\overline{s}$ is *fair* with respect to F^P and the "Q component" is *fair* with respect to F^Q.

Proposition 3 *Consider fair Moore machines $\mathcal{P}$ and $\mathcal{Q}$ such that the composition $\mathcal{P} \parallel \mathcal{Q}$ exists. Then T is a fair trace-tree of $\mathcal{P} \parallel \mathcal{Q}$ iff (1) $[T]_P$ is a fair trace-tree of $\mathcal{P}$, and (2) $[T]_Q$ is a fair trace-tree of $\mathcal{Q}$.*

Fair tree containment. We say that $\mathcal{Q}$ *fair-tree-contains* $\mathcal{P}$ if (1) $\mathcal{Q}$ is refinable by $\mathcal{P}$, and (2) for every fair trace-tree T of $\mathcal{P}$ its projection $[T]_Q$ is a fair-trace-tree of $\mathcal{Q}$.

Fair Simulation. We will use the generalization of simulation to account for fairness, as given in [HKR97]. Consider Moore machines $\mathcal{P}$ and $\mathcal{Q}$ such that Q is refinable by P. A strategy τ of $\mathcal{Q}$ with respect to $\mathcal{P}$ is a partial function from $\Sigma^P \times \Sigma^Q$ to S^Q. Given a finite or infinite s-run $\overline{s} = s_0 \xrightarrow{i_1} s_1 \xrightarrow{i_2} s_2 \xrightarrow{i_3} \ldots$, the *outcome* $\tau[\overline{s}]$ of the strategy is the finite or infinite sequence $\overline{s}' = s_0' \xrightarrow{i_1'} s_1' \xrightarrow{i_2'} s_2' \xrightarrow{i_3'} \ldots$ such that (1) $|\tau[\overline{s}]| = |\overline{s}|$, (2) for all $n \geq 0$, we have $s_n' = \tau(\overline{s}_{n+1}, \tau[\overline{s}]_n)$, and (3) for all $n > 0$, we have $i_n' = (i_n \cup L^P(s_{n-1})) \cap I^Q$. A binary relation $\Theta \subseteq S^P \times S^Q$ is a *fair simulation relation* of $\mathcal{P}$ by $\mathcal{Q}$ if the following two conditions hold:

1. $(s^P, s^Q) \in \Theta$.

2. For all $(p, q) \in \Theta$, we have $L^P(p) \cap O^Q = L^Q(q)$.

3. There exists a strategy τ of $\mathcal{Q}$ with respect to $\mathcal{P}$ such that, if $\Theta(p, q)$ and $\overline{s} = s_0 \xrightarrow{i_1} s_1 \xrightarrow{i_2} s_2 \xrightarrow{i_3} \ldots$, is a fair p-run of $\mathcal{P}$, the outcome $\tau[\overline{s}] = s'_0 \xrightarrow{i'_1} s'_1 \xrightarrow{i'_2} s'_2 \xrightarrow{i'_3} \ldots$ is a fair q-run of $\mathcal{Q}$ and $\tau[\overline{s}]$ Θ-matches $\overline{s}$ (that is, $(s_n, s'_n) \in \Theta$ for all $0 \leq n < |\overline{s}|$). We say that τ is a *witness* to the fair simulation Θ.

If such Θ exists, $\mathcal{Q}$ is said to *fairly simulate* $\mathcal{P}$, written as $\mathcal{P} \preceq_s^F \mathcal{Q}$. We will state some properties of fair simulation. First, since all finite runs are fair by definition, we have the following proposition.

Proposition 4 *Consider any two fair Moore machines $\mathcal{P} = \langle P, F^P \rangle$ and $\mathcal{Q} = \langle Q, F^Q \rangle$. If $\mathcal{P} \preceq_s^F \mathcal{Q}$ then $P \preceq_s Q$.*

Analogous to the equivalence between simulation and tree-containment, fair simulation can be proved to be equivalent to fair tree-containment [HKR97].

Proposition 5 *Consider any two fair Moore machines $\mathcal{P}$ and $\mathcal{Q}$. Then $\mathcal{P} \preceq_s^F \mathcal{Q}$ iff $\mathcal{Q}$ fair-tree-contains $\mathcal{P}$.*

5 Assume-Guarantee Rule for Fair Simulation

Let *Safe*$(\mathcal{P})$ be the fair Moore machine obtained by replacing the fairness constraint of $\mathcal{P}$ with the trivial fairness constraint that maps every run to *fair*. We now present the assume-guarantee proof rule for fair simulation. The same rule was proved for fair trace-trace containment in [AH96].

Theorem 6 *Let $\mathcal{P} = \langle P, F^P \rangle, \mathcal{Q} = \langle Q, F^Q \rangle, \mathcal{P}' = \langle P', F^{P'} \rangle$, and $\mathcal{Q}' = \langle Q', F^{Q'} \rangle$ be Moore machines such that $P \parallel Q$ and $P' \parallel Q'$ exist. Suppose $\mathcal{P} \parallel Safe(\mathcal{Q}') \preceq_s^F \mathcal{P}'$, $\mathcal{P}' \parallel \mathcal{Q} \preceq_s^F \mathcal{Q}'$, and every input of $P' \parallel Q'$ is either an input or output of $P \parallel Q$. Then, we have that $\mathcal{P} \parallel \mathcal{Q} \preceq_s^F \mathcal{P}' \parallel \mathcal{Q}'$.*

Proof: From Proposition 4, we know that $P \parallel Q' \preceq_s P'$ and $P' \parallel Q \preceq_s Q'$. Consequently, by Theorem 2 we know that $P \parallel Q \preceq_s P' \parallel Q'$. Therefore, from Proposition 1, we get that $P' \parallel Q'$ tree-contains $P \parallel Q$. Hence, if T is a trace-tree of $\mathcal{P} \parallel \mathcal{Q}$, then $[T]_{P' \parallel Q'}$ is a trace-tree of $\mathcal{P}' \parallel \mathcal{Q}'$. It remains to be proved that if T is a fair trace-tree of $\mathcal{P} \parallel \mathcal{Q}$, then $[T]_{P' \parallel Q'}$ is a fair trace-tree of $\mathcal{P}' \parallel \mathcal{Q}'$.

For notational simplicity, we omit explicit projections in the following. Suppose T is a fair trace-tree of $\mathcal{P} \parallel \mathcal{Q}$. From Proposition 3, T is a fair-trace-tree of both $\mathcal{P}$

and $\mathcal{Q}$. Further we know that T is a trace-tree of $P' \parallel Q'$. Thus, T is a fair trace-tree of $Safe(\mathcal{Q}')$. Again, by Proposition 3, T is a fair trace-tree of $\mathcal{P} \parallel Safe(\mathcal{Q}')$. Since $\mathcal{P} \parallel Safe(\mathcal{Q}') \preceq_s^F \mathcal{P}'$, we conclude (from Proposition 5) that T is a fair trace-tree of $\mathcal{P}'$. Therefore, from Proposition 3, T is a fair trace-tree of $\mathcal{P}' \parallel \mathcal{Q}$. Since $\mathcal{P}' \parallel \mathcal{Q} \preceq_s^F \mathcal{Q}'$, T is a fair trace-tree of $\mathcal{Q}'$. Finally from Proposition 3, since T is a fair trace-tree of both $\mathcal{P}'$ and $\mathcal{Q}'$, it is a fair trace-tree of $\mathcal{P}' \parallel \mathcal{Q}'$. ∎

References

[AH96] R. Alur and T.A. Henzinger. Reactive modules. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 207–218. IEEE Computer Society Press, 1996.

[AL88] M. Abadi and L. Lamport. The existence of refinement mappings. In *Proceedings of the Third Annual Symposium on Logic in Computer Science*, pages 165–175. IEEE Computer Society Press, 1988.

[AL95] M. Abadi and L. Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534, 1995.

[CLM89] E.M. Clarke, D.E. Long, and K.L. McMillan. Compositional model checking. In *Proceedings of the 4th Annual Symposium on Logic in Computer Science*, pages 353–362. IEEE Computer Society Press, 1989.

[GL94] O. Grumberg and D.E. Long. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems*, 16(3):843–871, 1994.

[HKR97] T.A. Henzinger, O. Kupferman, and S. K. Rajamani. Fair simulation. In *CONCUR 97: Theories of Concurrency*, Lecture Notes in Computer Science 1243, pages 273–287. Springer-Verlag, July 1997.

[Kur94] R.P. Kurshan. *Computer-aided Verification of Coordinating Processes.* Princeton University Press, 1994.

[Lyn96] N. A. Lynch. *Distributed Algorithms.* Morgan-Kaufman, 1996.

[McM97] K.L. McMillan. A compositional rule for hardware design refinement. In O. Grumberg, editor, *CAV 97: Computer Aided Verification*, Lecture Notes in Computer Science 1254, pages 24–35. Springer-Verlag, 1997.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *Second International Joint Conference on Artificial Intelligence*, pages 481–489. The British Computer Society, 1971.

[Sta85] E. W. Stark. A proof technique for rely/guarantee properties. In *Proceedings of the 5th Conference on Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science 206, pages 369–391. Springer-Verlag, 1985.