Checking for Language Inclusion Using Simulation Preorders

David L. Dill, Alan J. Hu, and Howard Wong-Toi*
Department of Computer Science
Stanford University

1 Introduction

Systems involving interaction among state machines, such as protocols, concurrent algorithms, and certain kinds of hardware, often contain subtle design errors that defy detection by conventional means, such as inspection, simulation, and testing a prototype. As a result, formal verification methods for such systems are of increasing interest.

We are interested in automatic verification using finite-state models of systems, with the underlying assumption that system behavior can be represented as a set of sequences representing all the possible histories (or traces) of the system (we assume *linear-time*). In this model, verification consists of testing for *language inclusion*: the implementation describes a set of actual traces and the specification gives the set of allowed traces; the implementation meets the specification if every actual trace is allowed.

In this paper, we consider only the case where both the implementation and the specification are represented by finite-state automata. The automata used here can describe both safety properties (which intuitively say that nothing bad happens), and liveness properties (which intuitively assert that something good eventually happens). More specifically, we deal with safety automata and Büchi automata.

As specifications become more complicated, it becomes less natural to express them with deterministic automata. This occurs because a complicated specification is more likely to have invisible internal state that is not a function of the externally visible state. Although such specifications can be expressed using deterministic automata, this places an unnecessary burden on the user. Determinization algorithms may cause exponential blowups and are also difficult to program.

Deciding language inclusion for non-deterministic automata is PSPACE-complete. Therefore it is highly unlikely that a polynomial technique can be used to decide language inclusion. However, deciding language inclusion for deterministic automata is known to be polynomial. Our main goal is to provide polynomial methods that work not only for deterministic automata, but also work for non-deterministic automata in cases of practical interest.

The *simulation preorder* is one of many preorders and equivalences considered by people studying branching-time models of concurrency. Simulation preorder is decidable in polynomial time (proportional to the product of the sizes of the two automata) even when the specification automaton is nondeterministic. However, the *simulation preorder* is a stronger relation between automata than language inclusion. So from our perspective (linear time), the simulation preorder should be regarded as an approximation (sufficient condition) for language inclusion that is much easier to check.

*This work was supported by the NSF under grant MIP-8858807. The second author is also supported by an ONR Graduate Fellowship.

One automaton precedes another in the simulation preorder if there exists a certain kind of correspondence, called a simulation relation, between states of the two automata (the correspondence is defined precisely below). Hence, deciding simulation preorder involves finding a simulation relation or proving that none exists. We consider below some variants on the simulation preorder that are expensive to check directly. In such cases, it may still be useful to do "semi-automatic verification": a human defines a candidate relation, and uses a computer to check automatically whether the candidate is a simulation relation. Hence, the computational complexity of checking a given simulation is also of interest.

The verification methods presented here are all incomplete, in that they can be used to prove language inclusion (whenever a simulation relation exists), but cannot decide language inclusion (that is, an automaton may accept a subset of the language of another without any simulation relation between them). This deficiency is a necessary sacrifice in return for efficiency. Nevertheless, we provide evidence that the technique is useful in practice through some examples.

1.1 Background

State relations in one form or another have been studied for a long time, including the *weak homomorphisms* and *coverings* of Ginzburg [Gin68] and the *simulations* of Milner [Mil71]. Many verification methods consider (possibly) infinite state automata, and therefore develop proof methodologies where the human verifier supplies a relation together with a mathematical proof that it is a simulation relation (for example, Milner's simulations, Lam and Shankar's protocol *projections* [LS84], the *possibilities mappings* of Lynch and Tuttle [LT87], Klarlund and Schneider's *invariants* [KS89] and the *progress measures* of Klarlund [Kla90]).

The Concurrency Workbench [CPS89] is one of several programs that test for simulation preorder between automata. However, none of these can handle large state spaces or liveness properties.

For liveness and fairness properties, we are interested in defining simulation relations on Büchi automata (finite automata that accept infinite strings). Park proposed using simulation relations on Muller automata, which are somewhat similar to Büchi automata. Checking Park's relations can be done in polynomial time, but automatically finding the relation is NP-complete, which limits their usefulness in practice (his simulations are similar to the relation called BSR-dlc below).

Lynch and Tuttle give a manual verification technique similar in spirit to our BSR-aa's on their IO-automata, which can also express fairness properties. Since they do not consider finite-state automata, neither testing a given relation nor finding one is decidable.

1.2 Notation

Let Σ be a set. Then Σ^* is the set of all finite sequences over Σ, and Σ^ω is the set of all infinite sequences over Σ. We will use Σ^∞ for $\Sigma^* \cup \Sigma^\omega$. If σ is in Σ^∞, its i-th element, if it exists, will be denoted σ_{i-1}, and σ may be identified with the corresponding string with the same elements. We let $len(\sigma)$ be the length of any σ in Σ^∞.

Let σ be in Σ^ω. The set of prefixes of σ, $pr(\sigma)$ is defined as $\{\sigma' \in \Sigma^* \mid \text{for all } i < len(\sigma'), \sigma'_i = \sigma_i\}$. Given a set $A \subseteq \Sigma^*$, its closure $cl(A)$ is the set of strings in Σ^ω such that every prefix is in A, i.e. $cl(A) = \{\sigma \mid pr(\sigma) \subseteq A\}$. The set $B \subseteq \Sigma^\omega$ is *closed* iff $B = cl(pr(B))$.

2 Safety Automata

Intuitively, states of an automaton represent states of the system or process being modeled. A state is made up of an external visible component, and an internal invisible component. A trace of an automaton is an infinite sequence of external states, and models what an external agent could observe of the process. All automata used here are finite-state and define languages of infinite traces.

A *safety automaton* A is a tuple $\langle S, E, P, N \rangle$. S is a finite set of *internal state components* and E is a finite set of *external state components*. The set of *states* of the automaton is $S \times E$. $P \subseteq S \times E$ is a set of *initial states*, and $N \subseteq (S \times E) \times (S \times E)$ is the *next state relation*. A *run* of A on the infinite sequence $e = e_0, e_1, \ldots \in E^\omega$ is an infinite sequence of states $\langle s_0, e_0 \rangle, \langle s_1, e_1 \rangle \ldots$ such that $\langle s_0, e_0 \rangle \in P$, and for all $i \geq 0$, $(\langle s_i, e_i \rangle, \langle s_{i+1}, e_{i+1} \rangle)$ is in N. The infinite sequence e is *accepted* by A if there is a run of A on e. The language $L(A)$ is the set of all *infinite* sequences for which there are accepting runs.

A is said to be *deterministic* if P is a singleton set and if for every state $\langle s, e \rangle \in S$ and every external component $e' \in E$ there is at most one state $\langle s', e' \rangle \in S \times E$ such that $(\langle s, e \rangle, \langle s', e' \rangle) \in N$.

2.1 Simulation Relations for Safety Automata

We consider first simulation relations between safety automata, $A_1 = \langle S_1, E, I_1, N_1 \rangle$ and $A_2 = \langle S_2, E, I_2, N_2 \rangle$. Intuitively every state in the implementation must be related to a state in the specification with the same external component. It must be possible for every transition in the implementation to be simulated by a transition in the specification. The simulation relations relate each implementation state to several specification states, like those of Park, Lynch and Tuttle, and Loewenstein and Dill [Par81, LT87, LD90].

Definition 1 (SSR) A *safety simulation relation* between safety automata A_1 and A_2 is any relation $R \subseteq S_1 \times E \times S_2$ that satisfies the following properties:

(SR1) (simulation) $\forall s_1 \in S_1, e \in E, s_2 \in S_2, s'_1 \in S_1, e' \in E$,

$$[\, R(s_1, e, s_2) \wedge N_1(\langle s_1, e \rangle, \langle s'_1, e' \rangle) \,] \Rightarrow \exists s'_2 \in S_2 \; [\, R(s'_1, e', s'_2) \wedge N_2(\langle s_2, e \rangle, \langle s'_2, e' \rangle) \,].$$

(SR2) (initiality) $\forall s_1 \in S_1, e \in E$,

$$\langle s_1, e \rangle \in P_1 \Rightarrow \; \exists s_2 \in S_2 \; [\, \langle s_2, e \rangle \in P_2 \wedge R(s_1, e, s_2) \,].$$

Theorem 1 (SSR soundness) *If there is an SSR between A_1 and A_2, then $L(A_1) \subseteq L(A_2)$.*

As mentioned in the introduction, one way of verifying language inclusion is to check that a user-supplied relation is actually a simulation relation.

Algorithm 1 (Checking safety simulation relations) It is straightforward to verify that a relation satisfies SR1 and SR2 independently. For example, SR1 may be verified by a simple check that outgoing transitions from implementation states are mimicked in their simulating specification states. This simple procedure is polynomial in the number of states and edges of the two automata.

2.2 Finding Simulation Relations

Because simulation relations over safety automata are closed under union, there is a largest simulation relation that contains all others. The algorithm to find simulation relations finds the largest candidate relation and then verifies it is indeed a simulation relation. This candidate relation R is the largest relation satisfying the simulation property, SR1. Since SR2 (initiality) is a monotone property (that is, if it is true in a relation R', it is true in any larger relation), there is a simulation relation over $S_1 \times E \times S_2$ iff R satisfies SR2. Computing R from the maximum relation $S_1 \times E \times S_2$ involves repeatedly deleting triples which do not locally satisfy SR1. It is then straightforward to check whether SR2 holds. The algorithm is polynomial in the size of the automata.

While safety simulation relations are in general incomplete, there are special cases where language inclusion does imply the existence of a simulation relation. A safety automaton A is *non-deadlocking* whenever every finite string σ' having a run on A is a prefix of some infinite string in $L(A)$.

Theorem 2 *If $L(A_1) \subseteq L(A_2)$, A_2 is deterministic and A_1 is non-deadlocking, then there is an SSR between A_1 and A_2.*

2.3 Symbolic Implementation

One way to contain the state explosion problem is to represent automata and relations "symbolically," using some data structure that does not expand as quickly as an explicit list of states. One such data structure is the *binary decision diagram* (BDD), which gives a compact representation for a Boolean function [Bry86]. These data structures have proven especially efficient in many cases. Using the paradigm of symbolic model checking, [BCMDH90, BCMD90] we can efficiently perform the computations specified below.

An expression for the maximum relation satisfying the simulation condition (SR1 in Definition 1) on a safety simulation relation is:

$$\nu Z.\lambda s_1, e, s_2 \left[Z(s_1, e, s_2) \wedge \forall s_1', e' \left[N_1(\langle s_1, e\rangle, \langle s_1', e'\rangle) \Rightarrow \exists s_2' \left[Z(s_1', e', s_2') \wedge N_2(\langle s_2, e\rangle, \langle s_2', e'\rangle)\right]\right]\right]$$

where $\nu Z.F[Z]$ denotes the greatest fixed point of the predicate transformer F. Let $Q(s_1, e, s_2)$ be this fixed point.

Theorem 3 *A safety simulation relation exists iff Q satisfies the initiality condition (SR2 in Definition 1).*

3 Simulation relations for liveness properties

While safety automata can express many useful properties, they cannot express simple liveness properties, such as "process A will eventually read the variable x". To handle general liveness properties, we need automata that can handle general ω-regular languages. Many such automata have been proposed: Büchi automata, Muller automata, Rabin automata, Streett automata, $\forall$-automata, and L-automata. For simplicity, we choose to work with the conceptually simplest of these, Büchi automata. The ideas expressed here can be extended to the other types of automata, as well.

A *Büchi automaton* $A = \langle S, E, P, N, F\rangle$ is a safety automaton with an additional fifth component $F \subseteq S \times E$, a set of *accepting states.* An infinite run r over the safety automaton $A_S = \langle S, E, P, N\rangle$ is called a run of the Büchi automaton A. The run r is an *accepting run* iff an accepting state occurs infinitely often in r. The *language* accepted by A is the set of all infinite strings with an accepting run. In the following, it is assumed every state in a Büchi automaton is reachable. A safety automaton can be considered to be a Büchi automaton in which $F = S \times E$.

Our definition of Büchi automata is non-standard: while our automata have external visible state components, the usual Büchi automata have visible labeled transitions between internal states. This change makes it easy for us to model our examples. However, there is a simple correspondence between the two definitions, and the simulations we propose can easily be applied to the more conventional definition of Büchi automata, also.

We define various simulation relations between Büchi automata, as extensions of simulation relations between safety automata. A Büchi automaton accepts an infinite string iff it has a run r for that string, and r is an *accepting* run, i.e. it includes infinitely many accepting states. Thus Büchi simulation relations must guarantee the existence not only of simulating runs but of simulating accepting runs.

Throughout the following we assume $A_1 = \langle S_1, E, P_1, N_1, F_1 \rangle$ and $A_2 = \langle S_2, E, P_2, N_2, F_2 \rangle$ are non-deadlocking Büchi automata.

3.1 Accepting-accepting Büchi simulation relations

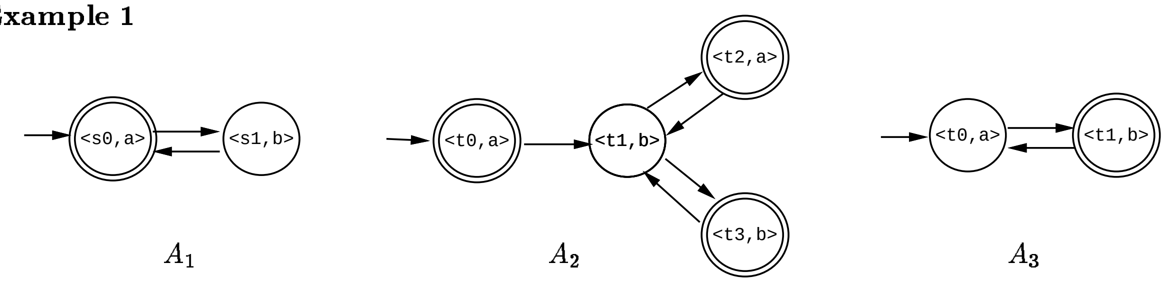
Here, safety simulation relations are augmented with a simple condition that guarantees that whenever the newly entered state of the implementation is accepting, then so must be its simulating A_2 state.

Definition 2 (BSR-aa) An *accepting-accepting Büchi simulation relation (BSR-aa)*, is any relation $R \subseteq S_1 \times E \times S_2$ that satisfies SR1 (simulation), SR2 (initiality) and the additional property:

$$\text{(SR-aa) } \forall s_1 \in S_1, e \in E, s_2 \in S_2, \quad R(s_1, e, s_2) \Rightarrow [F_1(\langle s_1, e \rangle) \Rightarrow F_2(\langle s_2, e \rangle)].$$

Theorem 4 (BSR-aa soundness) *If there is BSR-aa between A_1 and A_2, then $L(A_1) \subseteq L(A_2)$.*

Example 1



The relation $R = \{(s_0, a, t_0), (s_0, a, t_2), (s_1, b, t_1)\}$ is a BSR-aa between A_1 and A_2. However, there is no BSR-aa between A_1 and A_3, since the accepting states of one automaton are not "synchronized" with those of the other, even though clearly $L(A_1) \subseteq L(A_3)$.

3.2 Algorithms

As for safety simulation relations, we demonstrate algorithms for checking and finding each Büchi simulation relation. Büchi simulation relations are defined as safety simulation relations with an additional fairness property. When this additional property is of a certain form, the algorithms are trivial extensions of those for the safety case.

Suppose the fairness property determines *a priori* which pairings of automaton states are permitted in a simulation relation, independent of what other pairings appear in the simulation relation. Then a relation is a simulation relation whenever all pairings of states are permitted, or "good". Checking whether a relation is a Büchi simulation relation is then just checking it is a safety simulation relation and that all state pairings are good. Finding Büchi simulation relations is simply finding safety simulation relations among the good pairs.

Definition 3 A property P of relations in $S_1 \times E \times S_2$ is *locally-determined* if $P = 2^W$ for some $W \subseteq S_1 \times E \times S_2$.

Intuitively P is locally-determined iff R satisfies P whenever all triples in R are in some maximum relation W. We may interpret the triples in W as the good triples in $S_1 \times E \times S_2$.

Definition 4 Let BSR-P define the class of simulation relations that satisfy the properties SR1, SR2 and $SR - P$.

Lemma 1 *If $SR - P = 2^W$ is locally-determined, then a relation R is a BSR-P iff it is a safety simulation relation and contained in W.*

Theorem 5 *If $SR - P = 2^W$ is locally-determined and W is polynomially decidable, then checking whether a relation is a BSR-P is polynomial.*

Lemma 2 *If $SR - P$ is locally-determined, then BSR-P's are closed under union.*

Theorem 6 *If $SR - P = 2^W$ is locally-determined, and W is polynomially decidable, then deciding whether there is a BSR-P between two automata is polynomial.*

The property SR-aa is locally-determined, with SR-aa $= 2^W$, where $W = \{(s_1, e, s_2) \mid F_1(\langle s_1, e\rangle) \Rightarrow F_2(\langle s_2, e\rangle)\}$. Determining SR-aa is linear in the sizes of S_1 and S_2, so by the above results there are polynomial algorithms for checking and finding BSR-aa's.

3.3 Live-cycles Büchi simulation relations

We may relax the condition of having to simulate every F_1 state with an F_2 state. It is sufficient to simulate F_1 states by some state in S_2 from which it is guaranteed every A_2-run will later pass through an accepting state. Equivalently, it must be impossible to continue simulation of A_1 in a cycle from $\langle s_1, e\rangle \in F_1$ with a cycle of A_2 from $\langle s_2, e\rangle$ to $\langle s_2, e\rangle$ which does not pass through any states in F_2. In fact the converse is true, and this condition is also sufficient for language inclusion. Furthermore, it is locally-determined and polynomially decidable.

We first define the pseudo-product machine $A_{12} = \langle S_{12}, P_{12}, N_{12}\rangle$, where $S_{12} = S_1 \times E \times S_2$, $P_{12} = \{\langle s_1, e, s_2\rangle \mid P_1(\langle s_1, e\rangle) \text{ and } P_2(\langle s_2, e\rangle)\}$, and the next-state relation $N_{12} \subseteq S_{12} \times S_{12}$ is defined by $N_{12}(\langle s_1, e, s_2\rangle, \langle s_1', e, s_2'\rangle)$ iff $N_1(\langle s_1, e\rangle, \langle s_1', e'\rangle)$ and $N_2(\langle s_2, e\rangle, \langle s_2', e'\rangle)$. A product-state $\langle s_1, e, s_2\rangle$ is an F_1 state iff $\langle s_1, e\rangle \in F_1$, and likewise an F_2 state iff $\langle s_2, e\rangle \in F_2$.

Definition 5 (BSR-lc) A *live-cycles Büchi simulation relation (BSR-lc)*, between A_1 and A_2 is any relation $R \subseteq S_1 \times E \times S_2$ that satisfies SR1 (simulation), SR2 (initiality) and the additional property:

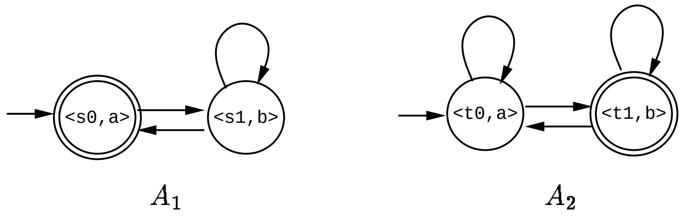
(SR-lc) $\forall s_1 \in S_1, e \in E, s_2 \in S_2,$

$$R(s_1, e, s_2) \Rightarrow [F_1(\langle s_1, e\rangle) \Rightarrow LC(\langle s_1, e, s_2\rangle)].$$

where $LC(\langle s_1, e, s_2\rangle)$ holds if every cycle through $\langle s_1, e, s_2\rangle$ in the pseudo-product machine A_{12} passes through an F_2 state (the cycle is "live").

Theorem 7 (BSR-lc soundness) *If there is a BSR-lc between A_1 and A_2, then $L(A_1) \subseteq L(A_2)$.*

Example 2 Here $R = \{(s_0, a, t_0), (s_1, b, t_1)\}$ is a BSR-lc, because the pseudo-product machine has the same structure as A_1. However, A_1 and A_2 have no BSR-ma.



Theorem 8 (BSR-lc completeness for deterministic specifications) *If $L(A_1) \subseteq L(A_2)$ and A_2 is deterministic, then there is a BSR-lc between A_1 and A_2.*

3.4 Dynamic-live-cycles Büchi simulation relations

The fairness properties of all the Büchi simulation relations defined so far have been locally-determined, and static in the sense that they are given as predetermined safety conditions over the state-pairings allowable. They do not take into consideration exactly which state pairings appear in the relation. However, in order to guarantee simulating runs are accepting, we need only consider runs permitted by R. Consider the pseudo-machine $A'_{12} = \langle S'_{12}, P'_{12}, N'_{12}\rangle$, with state set $S'_{12} = (S_1 \times E \times S_2) \cap R$, initial states $P'_{12} = P_{12} \cap R$, and the next-state relation given by $N'_{12}(\langle s_1, e, s_2\rangle, \langle s'_1, e', s'_2\rangle)$ iff $N_1(\langle s_1, e\rangle, \langle s'_1, e'\rangle)$, $N_2(\langle s_2, e\rangle, \langle s'_2, e'\rangle)$, and $\langle s_1, e, s_2\rangle, \langle s'_1, e, s'_2\rangle \in R$. The machine A'_{12} is simply A_{12} restricted to R, so there will be fewer non-live cycles and thus more simulation relations between the automata. The SR-dlc condition is merely SR-lc with cycles taken with respect to A'_{12} instead of A_{12}.

Definition 6 (BSR-dlc) A *dynamic-live-cycles Büchi simulation relation (BSR-dlc)*, is any relation $R \subseteq S_1 \times E \times S_2$ that satisfies SR1 (simulation), SR2 (initiality) and the additional property:

$$\text{(SR-dlc) } \forall s_1 \in S_1, e \in E, s_2 \in S_2,$$
$$R(s_1, e, s_2) \Rightarrow [F_1(\langle s_1, e\rangle) \Rightarrow LC'(\langle s_1, e, s_2\rangle)].$$

where $LC'(\langle s_1, e, s_2\rangle)$ iff every cycle through $\langle s_1, e, s_2\rangle$ in the pseudo-product machine A'_{12} is live, i.e. it passes through an F_2 state.

Theorem 9 (BSR-dlc soundness) *If there is a BSR-dlc between A_1 and A_2, then $L(A_1) \subseteq L(A_2)$.*

Theorem 10 (BSR-dlc deterministic completeness) *If A_2 is deterministic and $L(A_1) \subseteq L(A_2)$, then there is a BSR-dlc between A_1 and A_2.*

While SR-dlc is still polynomially decidable, it is not locally-determined. Thus checking BSR-dlc's is polynomial, and in fact finding BSR-dlc's is NP-complete.

	Deterministic completeness	Checking	Finding
BSR-aa	no	poly	poly
BSR-lc	yes	poly	poly
BSR-dlc	yes	poly	exponential

Figure 1: Completeness and Complexity of Büchi simulation relations.

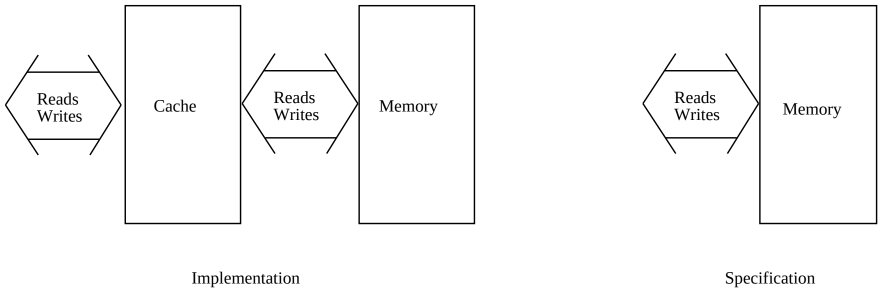


Figure 2: In the first verification example, a cached memory (nondeterministic) implements a memory (deterministic).

3.5 Summary/Comparative Expressiveness

Figure 1 summarizes the results above. Of the alternatives for Büchi simulations here, BSR-lc is the only one whose preorder is complete for deterministic specifications and decidable in polynomial time. Hence, we believe it is the one most likely to be of practical use in verification.

4 Verification Examples

The first example is adapted from an earlier paper using simulation relations [LD90]. We have a very general model of a cache (that allows prefetch, concurrent operations, etc.) and would like to show that a cached memory implements a memory. The cache model is nondeterministic and the memory model is deterministic. (See Figure 2.) Running on a DecStation 3100, finding a safety simulation relation took less than 5 seconds.

As CPU speeds have increased, the cache-memory port has become a bottleneck. Many architectures now incorporate a write buffer between the cache and the memory to reduce this problem. The second example considers a weaker memory model that allows the memory to buffer writes (delayed arbitrarily, but preserving the order of the writes) while allowing reads to bypass the buffer and read directly from memory. While this is not entirely realistic (real machines do not do it, at least intentionally), but it is similar to some consistency models used in *multi-processor* caching. We assume a finite-length write buffer.

We would like to show that the same cache from the first example can be attached to a write-buffered memory, with the result implementing a write-buffered memory. (See Figure 3.) Note that both the implementation and the specification are nondeterministic, demonstrating this important feature of simulation relations.

In under a minute, the verifier reported that no simulation relation exists. Additional queries to the system suggested the following scenario that demonstrates that in this case, the implementation is not correct with respect to the specification:

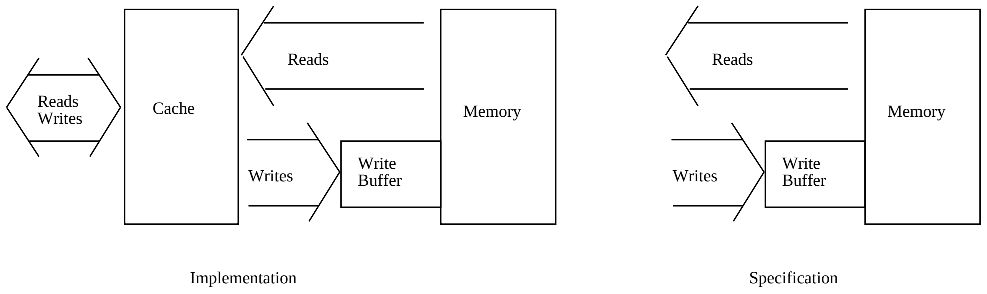


Figure 3: In the second example, a cached memory with write buffer (nondeterministic) fails to implement a memory with write buffer (nondeterministic).

1. Processor writes the value A to location X and receives an acknowledge from the memory system.
2. Processor performs operations not related to location X.
3. Processor writes a sequence of B's to location X (and receives acknowledgements for each). The number of B's written must be greater than the length of the write buffer.
4. Processor performs operations not related to location X.
5. Processor reads location X.

For the specification (a write-buffered memory), step 3 must result in a B being stored at location X because the write buffer must write B's to location X in order to keep from overflowing. Therefore, at step 5, the read must return the value B. For the cached implementation, however, consider the following possible scenario:

1. During step 1, the cache has a dirty copy of location X equal to the value A.
2. During step 2, the cache writes back its dirty copy. At some point, the write makes its way through the write buffer, so location X now equals A.
3. During step 3, the cache misses, reads a clean copy of $X = A$, and modifies its copy to a dirty copy $X = B$. Memory location X still holds value A.
4. During step 4, the cache writes back its dirty copy of $X = B$. This write gets buffered.
5. During step 5, the processor attempts to read location X. The cache misses and gets a clean copy of $X = A$ from the memory. The cache returns $X = A$.

This trace is possible in the implementation, but not in the specification.

To correct that bug, the third example again verifies that a cached, write-buffered memory implements a write-buffered memory, but the write buffer is modeled differently. We add an interlock to the memory to block a read to any location that has a write pending in the write buffer. (See Figure 4.) With this modification, the cached memory operates correctly. The verifier found a simulation relation in just over 20 seconds.

The following table summarizes the results. All runs used a DecStation 3100 with 16MB of memory. The implementation uses Brace, Rudell, and Bryant's package for boolean decision diagram manipulation. [BRB90]

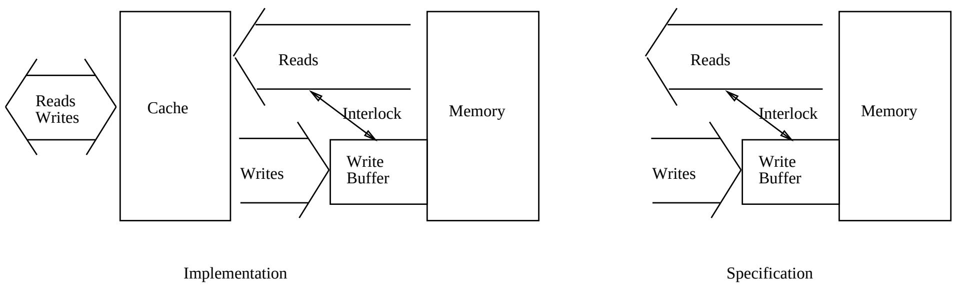


Figure 4: In the third example, a modified write buffer now blocks reads to locations that have a write pending in the buffer. Both the specification and the implementation are still nondeterministic.

Memory	Implementation (w/cache)		Specification		Simulation	Time
Model	Det	States	Det	States	Relation	(in sec)
Plain	No	64K	Yes	64	Yes	5
w/write buf	No	500K	No	500	No	41
w/interlock	No	500K	No	500	Yes	22

5 Conclusion

We have implemented an efficient verifier for language inclusion, using simulation relations as a heuristic. The examples above demonstrate the promise of this approach. Since the method is incomplete, more examples need to be verified to determine its practical usefulness. Future work along these lines includes development of improved diagnostics during verification, especially to suggest counterexamples when no simulation relation exists.

We plan to extend the implementation to find Büchi simulation relations. We are also investigating simulation relations defined over other forms of ω-automata.

Our framework deals only with the logical sequencing of events in trace traces. We are currently working on including timing properties in our specifications (*cf.* [LA89, Bes90]).

6 Acknowledgements

We would like to thank Andreas Drexler for his help in implementing the verifier.

References

[BCMD90] J.R. Burch, E.M. Clark, K.L. McMillan, and David L. Dill, "Sequential Circuit Verification Using Symbolic Model Checking," *27th ACM/IEEE Design Automation Conference,* 1990, pp. 46-51.

[BCMDH90] J.R. Burch, E.M. Clark, K.L. McMillan, D.L. Dill, and L.J. Hwang, "Symbolic Model Checking: 10^{20} States and Beyond," *Proceedings of the Conference on Logic in Computer Science*, 1990, pp. 428–439.

[Bes90] A.A. Bestavros, "The input-output timed automaton: a model for real-time parallel computation", Presentation at Workshop on Timing Issues in the Specification and Synthesis of Digital Systems, 1990.

[BRB90] Karl S. Brace, Richard L. Rudell, and Randal E. Bryant, "Efficient Implementation of a BDD Package," *27th ACM/IEEE Design Automation Conference,* 1990, pp. 40-45.

[Bry86] Randal E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, Vol. C-35, No. 8 (August 1986), pp. 677-691.

[CPS89] R. Cleaveland, J. Parrow, B. Steffen, "The Concurrency Workbench", Proceedings of the International Workshop on Automatic Verification of Finite State Systems, June 1989, LNCS 407, J. Sifakis (ed.), Springer-Verlag 1989, pp. 24–37.

[Gin68] A. Ginzburg, "Algebraic Theory of Automata", ACM Monograph Series, Academic Press, 1968.

[Kla90] N. Klarlund, "Progress Measures and Finite Arguments for Infinite Computations", Ph.D Thesis, Cornell University, TR 90-1153, September 1990.

[KS89] N. Klarlund and F.B. Schneider, "Verifying safety properties using infinite-state automata", Technical report TR-1036, Cornell University, 1989.

[Kur90] R. Kurshan, "Analysis of discrete event coordination", in *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, LNCS 430, J.W. deBakker, W.-P. de Roever, G. Rozenberg (eds.), Springer-Verlag 1990, pp. 414–453.

[LA89] N.A. Lynch, H. Attiya, "Using mappings to prove timing properties", MIT-LCS-TM-412.b, 1989.

[LD90] Paul Loewenstein and David Dill, "Formal Verification of Cache Systems using Refinement Relations," *IEEE International Conference on Computer Design,* 1990, pp. 228-233.

[LS84] S.S. Lam, A.U. Shankar, "Protocol verification via projections", *IEEE Transactions on Software Engineering*, SE-10(4):325–342, July 1984

[LT87] N.A. Lynch, M.R. Tuttle, "Hierarchical correctness proofs for distributed algorithms", in *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, 1987, pp. 137–151.

[Mil71] R. Milner, "An algebraic definition of simulation between programs", *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, British Computer Society, 1971, pp. 481–489.

[Par81] D.M.R. Park, "Concurrency and automata on infinite sequences", in *Proc. 5th GI conference (P. Deussen. ed.)*, LNCS 104, 1981, pp. 167–183.