Compositional and Fully Local Reasoning Of Optimized Reactive Systems

No Author Given

No Institute Given

Abstract.

placeholder abstract

Reactive systems are non-terminating systems that maintain an ongoing interaction with their environment. Examples of such systems include safety-critical systems like automotive controllers and communication networks, and omnipresent systems like microprocessors and operating systems. Reactive systems differ from transformational systems; their behaviors cannot be formalized and reasoned about using relations between the input states and output states. Since reactive systems are not expected to terminate, we cannot describe or analyze its correctness based on input-output relation. Moreover, the need to analyze the ongoing interaction of reactive systems with their environment enforces a view of the behavior less abstract than the relational view of the behavior of transformational systems. As a result, a formal description of behaviors of reactive systems is based on infinite computations.

1 Introduction

Reasoning about reactive systems using refinement involves showing that any (infinite) observable behavior of the low-level, optimized implementation is a behavior of the simple high-level abstract specification. Several notions of refinement like (bi)simulation refinement, stuttering (bi)simulation refinement, and skipping refinement have been proposed in the literation to directly account for the difference in the abstraction levels between a specification and an implementation. The two key principles crucial for scalability of a refinement-based methodology in analyzing the correctness of complex reactive systems are (1) Compositionality: it enables us to decompose a monolithic proof establishing refinement between a simple high-level abstract specification and a low-level concrete implementation into a sequence of simpler refinement proofs. Each of the intermediate refinement proof can then be performed independently using verification tools best suited for it. (2) Effective proof methods: analyzing the correctness of a reactive system requires global reasoning about its infinite behaviors, a task that is often difficult for existing verification tools. Hence it is crucial that the refinement-based methodology also admits effective proof methods that are amenable for automated reasoning.

It is well known that the (bi)simulation refinement and (bi)stuttering simulation refinement are compositional and therefore support the stepwise refinement methodology [16, 12]. Moreover, the proof methods associated with them are local, *i.e.*, requires reasoning about states and their successors, hence are amenable for automated reasoning. In this paper, we focus on skipping refinement, a notion of refinement that can directly analyze the correctness of optimized implementations that can run "faster" than their simple high-level specification. To motivate the need for an effective proof methods, we consider the event processing system (EPS), discussed in [7].

1.1 Motivating Example

An abstract high-level specification, AEPS, of an event processing system is defined as follows. Let E be a set of *events* and V be a set of *state variables*. A *state* of AEPS is a three-tuple $\langle t, Sch, St \rangle$, where t is a natural number denoting the current time; Sch is a set of pairs (e, t_e), where $e \in E$ is an event scheduled to be executed at time $t_e \geq t$; St is an assignment to state variables in V. The transition relation for the AEPS system is defined as follows. If at time t there is no $(e, t) \in Sch$, *i.e.*, there is no event scheduled to be executed at time t, then t is incremented by 1. Otherwise, we (nondeterministically) choose and execute an event of the form $(e, t) \in Sch$. The execution of an event may result in modifying St and also removing and adding a finite number of new pairs (e', t') to Sch. We require that $t' > t$. Finally, execution involves removing the executed event (e, t) from Sch. This is a simple but a generic model of an event processing system. Notice that the ability to remove events can be used to specify systems with preemption [15]: an event scheduled to execute at some future time may be cancelled (and possibly rescheduled to be executed at a different time in future) as a result of execution of an event that preempts it. Now consider, tEPS, an optimized implementation of AEPS. As before, a state is a three-tuple $\langle t, Sch, St \rangle$. However, unlike the abstract system which just increments time by 1 when there are no events scheduled at the current time, the optimized system finds the earliest time in future an event is scheduled to execute. The transition relation of tEPS is defined as follows. An event (e, t_e) with the minimum time is selected, t is updated to t_e and the event e is executed, as in the AEPS.

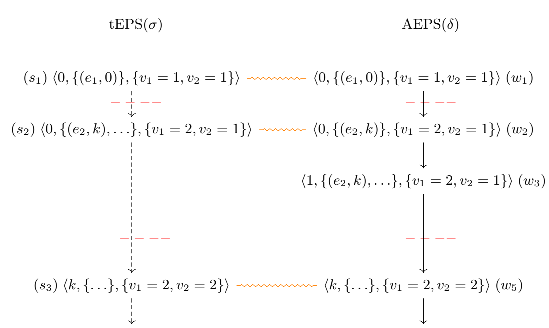

Fig. 1. Event simulation system

Consider an execution of AEPS and tEPS in Figure 1. (We only show the prefix of executions.) Suppose at $t = 0$, *Sch* be $\{(e_1, 0)\}$. The execution of event e_1 add a new pair (e_2, k) to *Sch*, where k is a positive integer. AEPS at $t = 0$, executes the event e_1, adds a new pair (e_2, k) to *Sch*, and updates t to 1. Since no events are scheduled to execute before $t = k$, the AEPS system repeatedly increments t by 1 until $t = k$. At $t = k$, it executes the event e_2. At time $t = 0$, tEPS executes e_1. The next event is scheduled to execute at time $t = k$; hence it updates in one step t to k. Next, in one step it executes the event e_2. Note that tEPS runs faster than AEPS by *skipping* over abstract states when no event is scheduled for execution at the current time. If $k > 1$, the step from s_2 to s_3 in tEPS neither corresponds to stuttering nor to a single step of the AEPS.

It is easy to see that notions of refinement based on stuttering simulation and bisimulation are not directly applicable to verify the correctness of tEPS. It was argued in [7] that skipping refinement is an appropriate notion to relate the behaviors of tEPS and its specification AEPS. Authors in [7] also provide two sound and complete proof methods, *reduced well-founded skipping simulation* and *well-founded skipping simulation*, to reason about skipping refinement and demonstrated their applicability for several systems with bounded skipping. However, notice that execution of an event in tEPS may add new events that are scheduled to execute at an arbitrary time in future, *i.e.*, in general k in the above example execution is unbounded. If we use the proof methods in [7] to verify that tEPS refines AEPS, would require unbounded reachability analysis which often is problematic for automated verification tools. Even in the particular case when one can a priori determine an upper bound on k and unroll the transition relation, the proposed proof methods are viable for mechanical reasoning only if the upper bound k is relatively small.

In this paper, we develop local proof methods to effectively analyze the correctness of an optimized reactive systems using skipping refinement. These proof methods reduce global reasoning about infinite computations to local reasoning about states and their successor. Moreover, we show that the proposed proof methods are complete, *i.e.*, if a system $\mathcal{M}_1$ is a skipping refinement of $\mathcal{M}_2$ under a suitable refinement map, then we can always locally reason about them. We also develop an algebraic theory of skipping refinement. In particular, we show that skipping simulation is closed under relational composition. Thus, skipping refinement aligns with the stepwise refinement methodology. Finally, we illustrate the benefits of the theory of skipping refinement and the associated proof methods by verifying the correctness of optimized event processing systems in ACl2s [2].

2 Preliminaries

A transition system model of a reactive system captures the concept of a state, atomic transitions that modify state during the course of a computation, and what is observable in a state. Any system with a well defined operational semantics can be mapped to a labeled transition system.

Definition 1 Labeled Transition System. *A labeled transition system (TS) is a structure $\langle S, \rightarrow, L\rangle$, where S is a non-empty (possibly infinite) set of states, $\rightarrow \subseteq S \times S$, is a left-total transition relation (every state has a successor), and L is a labeling function whose domain is S.*

Notation: We first describe the notational conventions used in the paper. Function application is sometimes denoted by an infix dot "." and is left-associative. For a binary relation R, we often write xRy instead of $(x, y) \in R$. The composition of relation R with itself i times (for $0 < i \leq \omega$) is denoted R^i ($\omega = \mathbb{N}$ and is the first infinite ordinal). Given a relation R and $1 < k \leq \omega$, $R^{<k}$ denotes $\bigcup_{1 \leq i < k} R^i$ and $R^{\geq k}$ denotes $\bigcup_{\omega > i \geq k} R^i$. Instead of $R^{<\omega}$ we often write the more common R^+. $\uplus$ denotes the disjoint union operator. Quantified expressions are written as $\langle Qx \colon r \colon t\rangle$, where Q is the quantifier (*e.g.*, $\exists, \forall, min, \bigcup$), x is a bound variable, r is an expression that denotes the range of variable x (*true*, if omitted), and t is a term.

Let $\mathcal{M} = \langle S, \rightarrow, L\rangle$ be a transition system. An $\mathcal{M}$-path is a sequence of states such that for adjacent states, s and u, $s \rightarrow u$. The j^{th} state in an $\mathcal{M}$-path σ is denoted by $\sigma.j$. An $\mathcal{M}$-path σ starting at state s is a *fullpath*, denoted by *fp.σ.s*, if it is infinite. An $\mathcal{M}$-segment, $\langle v_1, \ldots, v_k\rangle$, where $k \geq 1$ is a finite $\mathcal{M}$-path and is also denoted by $\vec{v}$. The length of an $\mathcal{M}$-segment $\vec{v}$ is denoted by $|\vec{v}|$. Let *INC* be the set of strictly increasing sequences of natural numbers starting at 0. The i^{th} partition of a fullpath σ with respect to $\pi \in$ *INC*, denoted by ${}^{\pi}\sigma^i$, is given by an $\mathcal{M}$-segment $\langle \sigma(\pi.i), \ldots, \sigma(\pi(i + 1) - 1)\rangle$.

3 Theory of Skipping Refinement

In this section we first briefly recall the notion of skipping simulation as described in [7]. We then study the algebraic properties of skipping simulation and show that a theory of refinement based on it is compositional and therefore can be used in a stepwise refinement based verification methodology.

The definition of skipping simulation is based on the notion of *matching*. Informally, a fullpath σ matches a fullpath δ under the relation B iff the fullpaths can be partitioned in to non-empty, finite segments such that all elements in a segment of σ are related to the first element in the corresponding segment of δ.

Definition 2 smatch [7]. *Let $\mathcal{M} = \langle S, \rightarrow, L\rangle$ be a transition system, σ, δ be fullpaths in $\mathcal{M}$. For $\pi, \xi \in INC$ and binary relation $B \subseteq S \times S$, we define*

$$scorr(B, \sigma, \pi, \delta, \xi) \equiv \langle \forall i \in \omega :: \langle \forall s \in {}^{\pi}\sigma^i :: sB\delta(\xi.i)\rangle\rangle \text{ and}$$
$$smatch(B, \sigma, \delta) \equiv \langle \exists \pi, \xi \in INC :: scorr(B, \sigma, \pi, \delta, \xi)\rangle.$$

Figure 1 illustrates the notion of matching using our running example: σ is the fullpath of the concrete system and δ is a fullpath of the absract system. (The figure only shows the prefix of the fullpaths.) The other parameter for matching is the relation B, which is just the identity function. In order to show that $smatch(B, \sigma, \delta)$ holds, we have to find $\pi, \xi \in INC$ satisfying the definition.

In the figure, we separate the partitions induced by our choice for π, ξ using $--$ and connect elements related by B with ___. Since all elements of a σ partition are related to the first element of the corresponding δ partition, $scorr(B, \sigma, \pi, \delta, \xi)$ holds, therefore, $smatch(B, \sigma, \delta)$ holds.

Using the notion of matching, skipping simulation is defined as follows. Notice that skipping simulation is defined using a single transition system; it is easy to lift the notion defined on a single transition system to one that relates two transition systems by taking the disjoint union of the transition systems.

Definition 3 Skipping Simulation. *$B \subseteq S \times S$ is a skipping simulation on a TS $\mathcal{M} = \langle S, \to, L\rangle$ iff for all s, w such that sBw, both of the following hold.*

(SKS1) $L.s = L.w$
(SKS2) $\langle \forall \sigma\colon fp.\sigma.s\colon \langle \exists \delta\colon fp.\delta.w\colon smatch(B, \sigma, \delta)\rangle\rangle$

Theorem 1. *Let $\mathcal{M}$ be a transition system.*

1. *If B is a simulation on $\mathcal{M}$ then B is a stuttering simulation on $\mathcal{M}$.*
2. *If B is a stuttering simulation on $\mathcal{M}$ then B is an SKS on $\mathcal{M}$.*

3.1 Algebraic Properties

We now study the algebraic properties of SKS. We show that it is closed under arbitrary union. We also show that SKS is closed under relational composition. The later property is particularly useful since it enables to use a stepwise refinement based verification methodology and modularly analyze the correctness of a complex implementation.

Lemma 1. *Let $\mathcal{M}$ be a TS and $\mathcal{C}$ be a set of SKS's on $\mathcal{M}$. Then $G = \langle \cup B : B \in \mathcal{C} : B\rangle$ is an SKS on $\mathcal{M}$.*

Corollary 1. *For any TS $\mathcal{M}$, there is a greatest SKS on $\mathcal{M}$.*

Lemma 2. *SKS are not closed under negation and intersection.*

The following lemma shows that skipping simulation is closed under relational composition.

Lemma 3. *Let $\mathcal{M}$ be a TS. If P and Q are SKS's on $\mathcal{M}$, then $R = P;Q$ is an SKS on $\mathcal{M}$.*

Proof. To show that R is an SKS on $\mathcal{M} = \langle S, \to, L\rangle$, we show that for any $s, w \in S$ such that sRw, SKS1 and SKS2 hold. Let $s, w \in S$ and sRw. From the definition of R, there exists $x \in S$ such that sPx and xQw. Since P and Q are SKS's on $\mathcal{M}$, $L.s = L.x = L.w$, hence, SKS1 holds for R.

To prove that SKS2 holds for R, consider a fullpath σ starting at s. Since P and Q are SKSs on $\mathcal{M}$, there is a fullpath τ in $\mathcal{M}$ starting at x, a fullpath δ in $\mathcal{M}$ starting at w and $\alpha, \beta, \theta, \gamma \in INC$ such that $scorr(P, \sigma, \alpha, \tau, \beta)$ and

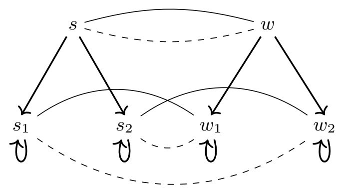

Fig. 2. An example showing that SKS is not closed under intersection. Consider a TS $\mathcal{M}$ with $S = \{s, w, s_1, s_2, w_1, w_2\}$. The transition relation is denoted by solid arrows and all states are labeled identically. The first SKS relation, denoted by solid lines, is $\{(s, w), (s_1, w_1), (s_2, w_2)\}$. The second SKS relation, denoted by dashed lines is $\{(s, w), (s_1, w_2), (s_2, w_1)\}$. The intersection of the two SKS relations is $\{(s, w)\}$ but does not include any related childrens.

$scorr(Q, \tau, \theta, \delta, \gamma)$ hold. We use the fullpath δ as a witness and define $\pi, \xi \in INC$ such that $scorr(R, \sigma, \pi, \delta, \xi)$ holds.

We define a function, r, that given i, corresponding to the index of a partition of τ under β, returns the index of the partition of τ under θ in which the first element of τ's i^{th} partition under β resides. $r.i = j$ iff $\theta.j \leq \beta.i < \theta(j + 1)$ Note that r is indeed a function, as every element of τ resides in exactly one partition of θ. Also, since there is a correspondence between the partitions of α and β, (by $scorr(P, \sigma, \alpha, \tau, \beta)$), we can apply r to indices of partitions of σ under α to find where the first element of the corresponding β partition resides. Note that r is non-decreasing: $a < b \Rightarrow r.a \leq r.b$.

We define $\pi\alpha \in INC$, a strictly increasing sequence that will allow us to merge adjacent partitions in α as needed to define the strictly increasing sequence π on σ used to prove SKS2. Partitions in π will consist of one or more α partitions. Given i, corresponding to the index of a partition of σ under π, the function $\pi\alpha$ returns the index of the corresponding partition of σ under α.

$$\pi\alpha(0) = 0$$

$$\pi\alpha(i) = \ \min j \in \omega \text{ s.t. } |\{k : 0 < k \leq j \wedge r.k \neq r(k-1)\}| = i$$

Note that $\pi\alpha$ is an increasing function, *i.e.*, $a < b \Rightarrow \pi\alpha(a) < \pi\alpha(b)$. We now define π as follows.

$$\pi.i = \alpha(\pi\alpha.i)$$

There is an important relationship between r and $\pi\alpha$

$$r(\pi\alpha.i) = \cdots = r(\pi\alpha(i + 1) - 1)$$

That is, for all α partitions that are in the same π partition, the initial states of the corresponding β partitions are in the same θ partition.

We define ξ as follows: $\xi.i = \gamma(r(\pi\alpha.i))$.

We are now ready to prove SKS2. Let $s \in {}^{\pi}\sigma^{i}$. We show that $sR\delta(\xi.i)$. By the definition of π, we have

$$s \in {}^{\alpha}\sigma^{\pi\alpha.i} \vee \cdots \vee s \in {}^{\alpha}\sigma^{\pi\alpha(i+1)-1}$$

Hence,

$$sP\tau(\beta(\pi\alpha.i)) \vee \cdots \vee sP\tau(\beta(\pi\alpha(i+1)-1))$$

Note that by the definition of r (apply r to $\pi\alpha.i$):

$$\theta(r(\pi\alpha.i)) \leq \beta(\pi\alpha.i) < \theta(r(\pi\alpha.i)+1)$$

Hence,

$$\tau(\beta(\pi\alpha.i))Q\delta(\gamma(r(\pi\alpha.i))) \vee \cdots \vee \tau(\beta(\pi\alpha(i+1)-1))Q\delta(\gamma(r(\pi\alpha(i+1)-1)))$$

By the definition of ξ and the relationship between r and $\pi\alpha$ described above, we simplify the above formula as follows.

$$\tau(\beta(\pi\alpha.i))Q\delta(\xi.i) \vee \cdots \vee \tau(\beta(\pi\alpha(i+1)-1))Q\delta(\xi.i)$$

Therefore, by the definition of R, we have that $sR\delta(\xi.i)$ holds. $\square$

Theorem 2. *The reflexive transitive closure of an SKS is an SKS.*

Theorem 3. *Given a TS $\mathcal{M}$, the greatest SKS on $\mathcal{M}$ is a preorder.*

3.2 Skipping Refinement

We recall the notion of skipping refinement, introduced in [7]. It uses skipping simulation, a notion defined in terms of a single transition system, to define skipping refinement, a notion that relates *two* transition systems: an *abstract* transition system and a *concrete* transition system. Informally, if a concrete system is a skipping refinement of an abstract system, then its observable behaviors are also behaviors of the abstract system, modulo skipping (which includes stuttering). The notion is parameterized by a *refinement map*, a function that maps concrete states to their corresponding abstract states. A refinement map along with a labeling function determines what is observable at a concrete state.

Definition 4 Skipping Refinement. *Let $\mathcal{M}_A = \langle S_A, \xrightarrow{A}, L_A\rangle$ and $\mathcal{M}_C = \langle S_C, \xrightarrow{C}, L_C\rangle$ be transition systems and let $r : S_C \rightarrow S_A$ be a refinement map. We say $\mathcal{M}_C$ is a skipping refinement of $\mathcal{M}_A$ with respect to r, written $\mathcal{M}_C \lesssim_r \mathcal{M}_A$, if there exists a binary relation B such that all of the following hold.*

1. $\langle \forall s \in S_C :: sBr.s\rangle$ and
2. *B is an SKS on $\langle S_C \uplus S_A, \xrightarrow{C} \uplus \xrightarrow{A}, \mathcal{L}\rangle$ where $\mathcal{L}.s = L_A(s)$ for $s \in S_A$, and $\mathcal{L}.s = L_A(r.s)$ for $s \in S_C$.*

Next, we use the property that skipping simulation is closed under relational composition to show that skipping refinement supports modular reasoning using a stepwise refinement approach. In order to verify that a low-level complex implementation $\mathcal{M}_C$ refines a simple high-level abstract specification $\mathcal{M}_A$ one proceeds as follows: starting with $\mathcal{M}_A$ define a sequence of intermediate systems leading to the final complex implementation $\mathcal{M}_C$. At each step in the sequence, show that system at the current step is a refinement of the previous one. Since at each step, the verification effort is largely focused only on the difference between two systems under consideration, proof obligations are simpler than the monolothic proof. Note that this methodology is orthogonal to (horizontal) modular reasoning that infers correctness of a system from the correctness of its sub-components.

Theorem 4. *Let $\mathcal{M}_1 = \langle S_1, \xrightarrow{1}, L_1\rangle$, $\mathcal{M}_2 = \langle S_2, \xrightarrow{2}, L_2\rangle$, and $\mathcal{M}_3 = \langle S_3, \xrightarrow{3}, L_3\rangle$ be transition systems, $p : S_1 \to S_2$ and $r : S_2 \to S_3$. If $\mathcal{M}_1 \lesssim_p \mathcal{M}_2$ and $\mathcal{M}_2 \lesssim_r \mathcal{M}_3$, then $\mathcal{M}_1 \lesssim_{p;r} \mathcal{M}_3$.*

Proof. Since $\mathcal{M}_1 \lesssim_p \mathcal{M}_2$, we have an SKS, say A, such that $\langle \forall s \in S_1 :: sA(p.s)\rangle$. Furthermore, without loss of generality we can assume that $A \subseteq S_1 \times S_2$. Similarly, since $\mathcal{M}_2 \lesssim_r \mathcal{M}_3$, we have an SKS, say B, such that $\langle \forall s \in S_2 :: sB(r.s)\rangle$ and $B \subseteq S_2 \times S_3$. Define $C = A; B$. Then we have that $C \subseteq S_1 \times S_3$ and $\langle \forall s \in S_1 :: sCr(p.s)\rangle$. Also, from Theorem 2, C is an SKS on $\langle S_1 \uplus S_3, \xrightarrow{1} \uplus \xrightarrow{3}, \mathcal{L}\rangle$, where $\mathcal{L}.s = L_3(s)$ if $s \in S_3$ else $\mathcal{L}.s = L_3(r(p.s))$.

□

Formally, to establish that a complex low-level implementation $\mathcal{M}_C$ refines a simple high-level abstract specification $\mathcal{M}_A$, one defines intermediate systems $\mathcal{M}_1, \ldots \mathcal{M}_n$, where $n \geq 1$ and establishes the following: $\mathcal{M}_C = \mathcal{M}_0 \lesssim_{r_0} \mathcal{M}_1 \lesssim_{r_1} \ldots \lesssim_{r_{n-1}} \mathcal{M}_n = \mathcal{M}_A$. Then from Theorem 4, we have that $\mathcal{M}_C \lesssim_r \mathcal{M}_A$, where $r = r_0; r_1; \ldots; r_{n-1}$. We illustrate the utility of this approach in Section 5 by proving correctness of an optimized event processing systems.

Theorem 5. *Let $\mathcal{M} = \langle S, \to, L\rangle$ be a transition system. Let $\mathcal{M}' = \langle S', \to', L\rangle$ where $S' \subseteq S$, $\to' \subseteq S' \times S'$, $\to'$ is a left-total subset of $\to^+$, and $L' = L|_{S'}$. Then $\mathcal{M}' \lesssim_I \mathcal{M}$, where I is the identity function on S'.*

Corollary 2. *Let $\mathcal{M}_C = \langle S_C, \xrightarrow{C}, L_C\rangle$ and $\mathcal{M}_A = \langle S_A, \xrightarrow{A}, L_A\rangle$ be transition systems, $r : S_C \to S_A$ be a refinement map. Let $\mathcal{M}'_C = \langle S'_C, \xrightarrow{C}', L'_C\rangle$ where $S'_C \subseteq S_C$, $\xrightarrow{C}'$ is a left-total subset of $\xrightarrow{C}^+$, and $L'_C = L_C|_{S'_C}$. If $\mathcal{M}_C \lesssim_r \mathcal{M}_A$ then $\mathcal{M}'_C \lesssim_{r'} \mathcal{M}_A$, where r' is $r|_{S'_C}$.*

We now illustrate the usefulness of the theory of skipping refinement using our running example of event processing systems. Consider MPEPS, that uses a priority queue to find a non-empty set of events (say E_t) scheduled to execute at the current time and executes them. We allow the priority queue in MPEPS to be deterministic or nondeterministic. For example, the priority queue may deterministically select a single event in E_t to execute, or based on considerations such as resource utilization it may execute some subset of events in E_t in a single step. When reasoning about the correctness of MPEPS, one thing to notice is that there is a difference in the data structures used in the two systems: MPEPS uses a priority queue to effectively find the next set of events to execute in the scheduler, while AEPS uses a simple abstract set representation for the scheduler. Another thing to notice is that MPEPS can run "faster" than AEPS in two ways: it can increment time by more than 1 and it can execute more than one event in a single step. The theory of skipping refinement developed in this chapter enables us to separate out these concerns and apply a stepwise refinement approach to effectively analyse MPEPS.

First, we account for the difference in the data structures between MPEPS and AEPS. Towards, this we define an intermediate system MEPS that is identical to MPEPS except that the scheduler in MEPS is now represented as a set of event-time pairs. Under a refinement map, say p, that extracts the set of event-time pairs in the priority queue of MPEPS, a step in MPEPS can be matched by a step in MEPS. Hence, MPEPS $\lesssim_p$ MEPS. Next we account for the difference between MEPS and AEPS in the number of events the two systems may execute in a single step. Towards this, observe that the state space of MEPS and tEPS are equal and the transition relation of MEPS is a left-total subset of the transitive closure of the transition relation of tEPS. Hence, from Theorem 5, we infer that MPEPS is a skipping refinement of tEPS using the identity function, say I_1, as the refinement map, *i.e.*, MEPS $\lesssim_{I_1}$ tEPS. Next observe that the state space of tEPS and AEPS are equal and the transition relation of tEPS is left-total subset of the transition relation of AEPS. Hence, from Theorem 5 , tEPS is a skipping refinement of AEPS using the identity function, say I_2, as the refinement map, *i.e.*, tEPS $\lesssim_{I_2}$ AEPS. Finally, from the transitivity of skipping refinement (Theorem 4), we conclude that $MPEPS \lesssim_{p'} AEPS$, where $p' = p; I_1; I_2$.

4 Mechanised Reasoning

To prove that a transition system $\mathcal{M}_C$ is a skipping refinement of a transition system $\mathcal{M}_A$ using Definition 3, requires us to show that for any fullpath from $\mathcal{M}_C$ we can find a matching fullpath from $\mathcal{M}_A$. However, reasoning about existence of infinite sequences can be problematic using automated tools. In this section, we develop sound and complete local proof methods that are applicable even if a system exhibits unbounded skipping. We first briefly present the proof methods, reduced well-founded skipping and well-founded skipping simulation, developed in [7].

Definition 5 Reduced Well-founded Skipping [7]. *$B \subseteq S \times S$ is a reduced well-founded skipping relation on TS $\mathcal{M} = \langle S, \rightarrow, L\rangle$ iff :*

(RWFSK1) $\langle \forall s, w \in S : sBw : L.s = L.w\rangle$

(RWFSK2) There exists a function, rankt $: S \times S \rightarrow W$, *such that* $\langle W, \prec\rangle$ *is well-founded and*

$\langle \forall s,u, w \in S : s \rightarrow u \wedge sBw :$

(a) $(uBw \wedge rankt(u, w) \prec rankt(s, w)) \vee$

(b) $\langle \exists v : w \rightarrow^+ v : uBv\rangle\rangle$

Definition 6 Well-founded Skipping [7]. *$B \subseteq S \times S$ is a well-founded skipping relation on TS $\mathcal{M} = \langle S, \rightarrow, L \rangle$ iff :*

(WFSK1) $\langle \forall s, w \in S : sBw : L.s = L.w \rangle$
(WFSK2) There exist functions, $rankt : S \times S \rightarrow W$, $rankl : S \times S \times S \rightarrow \omega$, *such that $\langle W, \prec \rangle$ is well-founded and*

$\langle \forall s, u, w \in S : s \rightarrow u \wedge sBw :$

(a) $\langle \exists v : w \rightarrow v : uBv \rangle \vee$

(b) $(uBw \wedge rankt(u, w) \prec rankt(s, w)) \vee$

(c) $\langle \exists v : w \rightarrow v : sBv \wedge rankl(v, s, u) < rankl(w, s, u) \rangle \vee$

(d) $\langle \exists v : w \rightarrow^{\geq 2} v : uBv \rangle \rangle$

Theorem 6 [7]. *Let $\mathcal{M} = \langle S, \rightarrow, L \rangle$ be a TS and $B \subseteq S \times S$. The following statements are equivalent*

(i) B is a SKS on $\mathcal{M}$;
(ii) B is a WFSK on $\mathcal{M}$;
(iii) B is a RWFSK on $\mathcal{M}$.

Recall the event processing systems AEPS and tEPS described in Section 1.1. When no events are scheduled to execute at a given time, say t, tEPS increments time t to the earliest time in future, say $k > t$, at which an event is scheduled for execution. Execution of an event can add an event that is scheduled to be executed at an arbitrary time in future. Therefore, we cannot apriori determine an upper-bound on k. Using any of the above two proof-methods to reason about skipping refinement would require unbounded reachability analysis, often difficult for automated verification tools. To redress the situation, we develop two new proof methods of SKS; both require only local reasoning about steps and their successors.

Definition 7 Reduced Local Well-founded Skipping. *$B \subseteq S \times S$ is a local well-founded skipping relation on TS $\mathcal{M} = \langle S, \rightarrow, L \rangle$ iff:*
(RLWFSK1) $\langle \forall s, w \in S : sBw : L.s = L.w \rangle$
(RLWFSK2) There exist functions, $rankt : S \times S \longrightarrow W$, $rankls : S \times S \longrightarrow \omega$ *such that $\langle W, \prec \rangle$ is well founded, and, a binary relation $\mathcal{O} \subseteq S \times S$ such that*

$\langle \forall s, u, w \in S : sBw \ \wedge \ s \rightarrow u :$

(a) $(uBw \ \wedge \ rankt(u, w) \prec rankt(s, w)) \vee$

(b) $\langle \exists v : w \rightarrow v : u\mathcal{O}v \rangle \rangle$

and

$\langle \forall x, y \in S : x\mathcal{O}y :$

(c) $xBy \vee$

(d) $\langle \exists z : y \rightarrow z : x\mathcal{O}z \ \wedge \ rankls(z, x) < rankls(y, x) \rangle \rangle$

Observe that to prove that a relation is an RLWFSK on a transition system, it is sufficient to reason about single steps of the transition system. Also, note that

RLWFSK does not differentiate between skipping and stuttering on the right. This is based on an earlier observation that skipping subsumes stuttering. We used this observation to simplify the definition. However, it can often be useful to differentiate between skipping and stuttering. Next we define local well-founded skipping simulation (LWFSK), a characterization of skipping simulation that separates reasoning about skipping and stuttering on the right.

Definition 8 Local Well-founded Skipping. *$B \subseteq S \times S$ is a local well-founded skipping relation on TS $\mathcal{M} = \langle S, \rightarrow, L \rangle$ iff:*

(LWFSK1) $\langle \forall s, w \in S : sBw : L.s = L.w \rangle$

(LWFSK2) There exist functions, $rankt : S \times S \longrightarrow W$, $rankl : S \times S \times S \longrightarrow \omega$, and $rankls : S \times S \longrightarrow \omega$ such that $\langle W, \prec \rangle$ is well founded, and, a binary relation $\mathcal{O} \subseteq S \times S$ such that

$\langle \forall s, u, w \in S : sBw \wedge s \rightarrow u :$

(a) $\langle \exists v : w \rightarrow v : uBv \rangle \vee$

(b) $(uBw \wedge rankt(u, w) \prec rankt(s, w)) \vee$

(c) $\langle \exists v : w \rightarrow v : sBv \wedge rankl(v, s, u) < rankl(w, s, u) \rangle \vee$

(d) $\langle \exists v : w \rightarrow v : u\mathcal{O}v \rangle \rangle$

and

$\langle \forall x, y \in S : x\mathcal{O}y :$

(e) $xBy \vee$

(f) $\langle \exists z : y \rightarrow z : x\mathcal{O}z \wedge rankls(z, x) < rankls(y, x) \rangle \rangle$

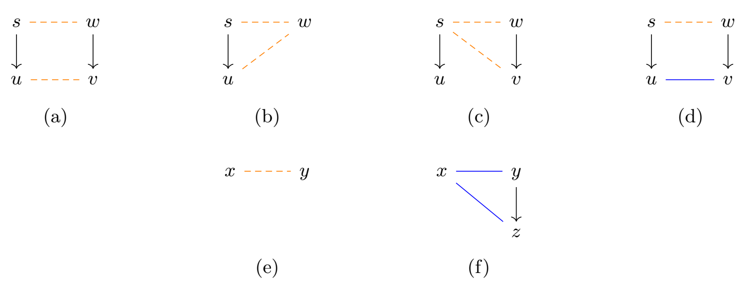

Fig. 3. Local well-founded skipping simulation (orange line indicates the states are related by B and blue line indicate the states are related by $\mathcal{O}$)

Like RLWFSK, to prove that a relation is a LWFSK, reasoning about single steps of the transition system suffices. However, LWFSK2b accounts for stuttering on the right, and LWFSK2d along with LWFSK2e and LWFSK2f acounts for skipping on the right. Also observe that states related by $\mathcal{O}$ are not required to be labeled identically and may have no observable relationship to the states related by B.

Soundness and Completeness We next show that RLWFSK and LWFSK in fact completely characterize skipping simulation, *i.e.*, RLWFSK and LWFSK are sound and complete proof rules. Thus if a conceret system $\mathcal{M}_C$ is a skipping refinement of $\mathcal{M}_A$, one can always effectively reason about it using RLWFSK and LWFSK.

Theorem 7. *Let $\mathcal{M} = \langle S, \rightarrow, L\rangle$ be a transition system and $B \subseteq S \times S$. The following statements are equivalent:*

(i) B is an SKS on $\mathcal{M}$;
(ii) B is a WFSK on $\mathcal{M}$;
(iii) B is an RWFSK on $\mathcal{M}$;
(iv) B is an RLWFSK on $\mathcal{M}$;
(v) B is a LWFSK on $\mathcal{M}$;

Proof. The equivalence of (i), (ii) and (iii) follows from Theorem 6. That (iv) implies (v) follows from the simple observation that RLWFSK2 implies LWFSK2. To complete the proof, we prove the following two implications. We prove below that (v) implies (ii) in Lemma 4 and that (iii) implies (iv) in Lemma 5.

□

Lemma 4. *If B is a LWFSK on $\mathcal{M}$, then B is a WFSK on $\mathcal{M}$.*

Proof. Let B be a LWFSK on $\mathcal{M}$. WFSK1 follows directly from LWFSK1.

Let *rankt*, *rankl*, and *rankls* be functions, and $\mathcal{O}$ be a binary relation such that LWFSK2 holds. To show that WFSK2 holds, we use the same *rankt* and *rankl* functions and let $s, u, w \in S$ and $s \rightarrow u$ and sBw. LWFSK2a, LWFSK2b and LWFSK2c are equivalent to WFSK2a, WFSK2b and WFSK2c, respectively, so we show that if only LWFSK2d holds, then WFSK2d holds. Since LWFSK2d holds, there is a successor v of w such that $u\mathcal{O}v$. Since $u\mathcal{O}v$ holds, either LWFSK2e or LWFSK2f must hold between u and v. However, since LWFSK2a does not hold, LWFSK2e cannot hold and LWFSK2f must hold, *i.e.*, there exists a successor v' of v such that $u\mathcal{O}v' \wedge rankls(v', u) < rankls(v, u)$. So, we need a path of at least 2 steps from w to satisfy the universally quantified constraint on $\mathcal{O}$. Let us consider an arbitrary path, δ, such that $\delta.0 = w$, $\delta.1 = v$, $\delta.2 = v'$, $u\mathcal{O}\delta.i$, LWFSK2e does not hold between u and $\delta.i$ for $i \geq 1$, and $rankls(\delta.(i + 1), u) < rankls(\delta.i, u)$. Notice that any such path must be finite because *rankls* is well founded. Hence, δ is a finite path and there exists a $k \geq 2$ such that LWFSK2e holds between u and $\delta.k$. Therefore, WFSK2d holds, *i.e.*, there is a state in δ reachable from w in two or more steps which is related to u by B.

□

Lemma 5. *If B is RWFSK on $\mathcal{M}$, then B is an RLWFSK on $\mathcal{M}$.*

Proof. Let B be an RWFSK on $\mathcal{M}$. RLWFSK1 follows directly from RWFSK1. To show that RLWFSK2 holds, we use any *rankt* function that can be used to show that RWFSK2 holds. We define $\mathcal{O}$ as follows.

$$\mathcal{O} = \{(u, v) : \langle \exists z : v \rightarrow^{+} z : uBz\rangle\}$$

We define $rankls(u, v)$ to be the minimal length of a $\mathcal{M}$-segment that starts at v and ends at a state, say z, such that uBz, if such a segment exists and 0 otherwise. Let $s, u, w \in S$, sBw and $s \to u$. If RWFSK2a holds between s, u, and w, then RLWFSK2a also holds. Next, suppose that RWFSK2a does not hold but RWFSK2b holds, *i.e.*, there is an $\mathcal{M}$-segment $\langle w, a, \ldots, v\rangle$ such that uBv; therefore, $u\mathcal{O}a$ and RLWFSK2b holds.

To finish the proof, we show that $\mathcal{O}$ and *rankls* satisfy the constraints imposed by the second conjunct in RLWFSK2. Let $x, y \in S$, $x\mathcal{O}y$ and $x \not B\, y$. From the definition of $\mathcal{O}$, we have that there is an $\mathcal{M}$-segment from y to a state related to x by B; let $\overrightarrow{y}$ be such a segment of minimal length. From definition of *rankls*, we have $rankls(y, x) = |\overrightarrow{y}|$. Observe that y cannot be the last state of $\overrightarrow{y}$ and $|\overrightarrow{y}| \geq 2$. This is because the last state in $\overrightarrow{y}$ must be related to x by B, but from the assumption we know that $x \not B\, y$. Let y' be a successor of y in $\overrightarrow{y}$. Clearly, $x\mathcal{O}y'$; therefore, $rankls(y', x) < |\overrightarrow{y}| - 1$, since the length of a minimal $\mathcal{M}$-segment from y' to a state related to x by B, must be less or equal to $|\overrightarrow{y}| - 1$. $\square$

5 Case Study (Event Processing System)

In this section, we analyze the correctness of an optimized event processing system (PEPS) that uses a *priority queue* to find an event to execute scheduled to execute at any given time. We show that PEPS refines AEPS, a simple event processing system described in Section 1. Our goal is to illustrate the benefits of the theory of skipping refinement and the associated local proof methods developed in the paper. We use ACL2s [2], an interactive theorem prover, to define the operational semantics of the systems and mechanise the proof of correctness.

Operational Semantics of PEPS A state of PEPS is a three tuple $\langle \texttt{tm}, \texttt{otevs}, \texttt{mem}\rangle$, where `tm` is a natural number denoting current time, `otevs` is a set of timed-event pairs denoting the scheduler that is ordered with respect to a total order `te-<` on timed-event pairs, and `mem` is a collection of variable-integer pairs denoting the shared memory. The transition function of PEPS is defined as follows: if there are no events in `otevs`, then PEPS just increments the current time by 1. Else it picks the first timed-event pair, say $\langle e, t\rangle$ in `otevs`, executes it, and updates the time to t. The execution of an event may result in adding new timed-events to the scheduler, removing existing timed-events from the scheduler, and update to the memory. Finally, the executed timed-event is removed from the scheduler. Labeling function is just the identity function. Notice that PEPS is a deterministic system. This domain-specific information is useful in reducing the proof obligations to establish skipping refinement.

The execution of an event is modelled using three constraint functions that take as input an event `ev`, time `t` and memory `mem`: `step-events-add` returns the set of new timed-event pairs to add to the scheduler; `step-events-rm` returns the set of timed-event pairs to remove from the scheduler; and `step-memory` returns an updated assignments to variables in the memory. We place minimal constraints on these functions. For example, we only require that `step-events-add` returns a set of event-time pairs of the form (e, t_e) where t_e is greater than t and given the same event, same time and set-equivalent memories it returns set-equivalent set of timed-event pairs. The constraints functions are defined using the `encapsulate` construct in ACL2s and can be instantiated with any executable definitions that satsify these constraints without affecting the proof of correctness of PEPS. Moreover, note that the particular choice of the total order on timed-event pairs is irrelevant to the proof of correctness of PEPS.

Stepwise Refinement: We show that PEPS refines AEPS using a stepwise refinement approach: first we define an intermediate system HPEPS obtained by augmenting PEPS with history information and show that PEPS is a simulation refinement of HPEPS. Second, we show that HPEPS is a skipping refinement of AEPS. Finally, we appeal to Theorem 1 and Theorem 4 to infer that PEPS refines AEPS. Note that the compositionality of skipping refinement enables us to decompose the proof into a sequence of simpler refinement proofs, each of which is simpler. Moreover, the history information in HPEPS is helpful in defining the witnessing binary relation and the rank function required to prove skipping refinement.

An HPEPS state is a four-tuple $\langle tm, otevs, mem, h\rangle$, where `tm`, `otevs`, `mem` are respectively the current time, an ordered set of timed events and a collection of variable-integer pairs, and `h` is the history information. The history information `h` consists of a boolean variable `valid`, time `tm`, and an ordered set of timed-event pairs `otevs` and the memory `mem`. Informally, `h` records the state preceding the current state. Finally, labeling function is just the identity function. The transition function HPEPS is same as the transition function of PEPS except that HPEPS also records the history in `h`.

PEPS refines HPEPS : Observe that, modulo the history information, a step of PEPS directly corresponds to a step of HPEPS, *i.e.*, PEPS is a bisimulation refinement of HPEPS. But we only prove that it is a simulation refinement, because from Theorem 1, it suffices to establish that PEPS is a skipping refinement of HPEPS. The proofs primarily requires showing that two sets of ordered timed-events that are set equivalent are infact and that add/remove equivalent sets of timed-event on equal schedulers results in equal schedulers.

HPEPS refines AEPS: Next we show that HPEPS is a skipping refinement of AEPS under the refinement map R, a function that simply projects an HPEPS state to an AEPS state. To show that HPEPS is a skipping refinement of AEPS under the refinement map R, from Definition 4, we must show as witness a binary relation B that satisfies the two conditions. Let $B = \{(s, R.s) : s\ \textit{is an HPEPS state}\}$. To establish that B is an SKS on the disjoint union of HPEPS and AEPS, we have choice of four proof-methods (Section 4). Recall that execution of an event can add a new event scheduled to be executed at an arbitrary time in the future. As a result, if we were to use WFSK or RWFSK, proof methods, first introduced in [7], conditions WFSK2d (Definition 5) and RWFSK2b (Definition 6) would require unbounded reachability analysis, a task that is often difficult for verification tools. In contrast, proof obligations to establish RLWFSK are local and requires reasoning only about states and their successors. This significantly reduces the proof complexity.

RLWFSK1 holds trivially. To prove that RLWFSK2 holds we define a binary relation $\mathcal{O}$ and a rank function *rankls* and show that they satisfy the two universally quantified formulas in RLWFSK2. Moreover, since HPEPS does not stutter we ignore RLWFSK2a, and that is why we do not define *rankt*. Finally, our proof obligations are as follows: for all HPEPS s, u and AEPS state w such that $s \rightarrow u$ and sBw holds, there exists a AEPS state v such that $w \rightarrow v$ and $u\mathcal{O}v$ holds.

Verification Effort: We use `defdata` framework in ACL2s, to specify the data definitions for the three systems and the `definec` construct to introduce function definitions along with their input-contract (pre-condition) and output-contract (post-condition). In addition to admiting a data definition, `defdata` proves several theorems about the functions that are extremely helpful in automatically discharging type-like proof obligations. We also develop a framework to concisely describe functions using higher-order constructs like map and reduce. ACL2s supports specifying first-order quantifiers by way of the `defun-sk` construct. We use `defun-sk` to model the transition relation for AEPS (a non-deterministic system) and specify the proof obligations for proving that HPEPS refines AEPS. However, support for automated reasoning about quantifiers is limited in ACL2s. Therfore, we use the domain knowledge, when possible (*e.g.*, a system is deterministic), to eliminate quantifiers in the proof obligations.

The proof makes essential use of several libraries available in ACL2 for reasoning about lists and sets. In addition, we prove a collection of additional lemmas that can be roughly categorized into four categories. First, the collection of lemmas to prove the input-output contracts of the functions. Second, a collection of lemmas to show that operations on the schedulers in the three systems preserve the invariant that any timed-events in the scheduler is scheduled to execute at a time greater or equal to the current time. Third, a collection of lemmas to show that inserting and removing two equivalent sets of timed-events from a scheduler results in an *equivalent* scheduler. And fourth, a collection of lemmas were proven to show that two schedulers are equivalent *iff* they are set equal. The above lemmas play an important part in establishing a relationship between priority queue, a data structures in the implementation system and a set, the data structure the specification system. The behavioral difference between the two systems is accounted for by the notion of skipping refinement. This separation significantly eases understanding as well as mechanical reasoning about correctness of reactive systems. In total we introduced 9 data definitions and 28 functions to model the three systems. We have 8 top-level proof obligations and

updated the stats

144 supporting lemmas. The entire proof takes about 120 seconds on a machine with 2.2 GHz Intel Core i7 with 16GB main memory.

6 Related Work

This paper develops a theory of skipping refinement to effectively prove the correctness of optimized reactive systems using automated verification tools. These results establish skipping refinement on par with notions of refinement based on (bi)simulation [14] and stuttering (bi)simulation [16, 12], in the sense that skipping refinement is compositional and admits local proofs methods. Together the two properties have been instrumental in reducing the proof complexity in verification of large and complex systems. The proof of correctness (implementation refines the specification) is decomposed into simpler refinement proof obligations. Each simpler proof obligations can then be discharged using an associated local-proof method.

Refinement-based methodology has been successfully used to verify the correctness of several realistic hardware and software systems. In [9], several complex concurrent programs we verified using a stepwise refinement methodology. In addition, they also develop a compact representation to facilitate the description of programs at different levels of abstraction and associated refinement proofs. Several back-end compiler transformations are proved correct in Compcert [10] using simulation refinement. In [17], several compiler transformations were verified using stuttering refinement and associated local proof methods. Recently, refinement-based methodology has also been applied to verify the correctness of practical distributed systems [5] and a general-purpose operating system microkernel [8]. The full verification of CertiKOS [3, 4], an OS kernel, is based on the notion of simulation refinement. Refinement based approach has been extensively used to verification of microprocessors [19, 6, 1, 18, 13]. Skipping refinement was used to verify the correctness of optimized memory controllers and a JVM-inspired stack machine [7].

7 Conclusions

In this paper, we studied a refinement-based methodology to effectively analyze the correctness of optimized reactive systems, a class of systems where a single transition in the concrete low-level implementation corresponds to a sequence of observable steps in the abstract high-level specification. We developed sound and complete proof methods that reduce global reasoning about infinite computations of a reactive system to local reasoning about states and their successors. We also show that the skipping simulation is closed under composition and therefore is amenable for modular reasoning using a stepwise refinement approach. We experimentally validated the usefulness of skipping refinement and the associated local proof methods to analyze the correctness of an optimized event-processing system in ACL2s. For future work, we plan to precisely classify temporal logic properties that are preserved by skipping refinement. This would enable us to establish properties of the low-level concrete implementations by analysing the properties of the simpler high-level abstract system.

References

1. Aagaard, M., Cook, B., Day, N., Jones, R.: A framework for microprocessor correctness statements. Correct Hardware Design and Verification Methods (2001)
2. Chamarthi, H.R., Dillinger, P.C., Manolios, P., Vroon, D.: The ACL2 sedan theorem proving system. In: TACAS (2011)
3. Gu, L., Vaynberg, A., Ford, B., Shao, Z., Costanzo, D.: Certikos: a certified kernel for secure cloud computing. In: APSys (2011)
4. Gu, R., Koenig, J., Ramananandro, T., Shao, Z., Wu, X., Weng, S.C., Zhang, H., Guo, Y.: Deep specifications and certified abstraction layers. In: POPL (2015)
5. Hawblitzel, C., Howell, J., Kapritsos, M., Lorch, J.R., Parno, B., Roberts, M.L., Setty, S., Zill, B.: Ironfleet: Proving practical distributed systems correct. In: SOSP (2015)
6. Hosabettu, R., Gopalakrishnan, G., Srivas, M.: Formal verification of a complex pipelined processor. In: FMSD (2003)
7. Jain, M., Manolios, P.: Skipping refinement. In: CAV (2015)
8. Klein, G., Sewell, T., Winwood, S.: Refinement in the formal verification of the sel4 microkernel. In: Design and Verification of Microprocessor Systems for High-Assurance Applications (2010)
9. Kragl, B., Qadeer, S.: Layered concurrent programs. In: CAV 2018 (2018)
10. Leroy, X., Blazy, S.: Formal verification of a c-like memory model and its uses for verifying program transformations. Journal of Automated Reasoning (2008)
11. Manolios, P.: Mechanical verification of reactive systems. Ph.D. thesis, University of Texas (2001)
12. Manolios, P.: A compositional theory of refinement for branching time. In: CHARME (2003)
13. Manolios, P., Srinivasan, S.K.: A complete compositional reasoning framework for the efficient verification of pipelined machines. In: ICCAD (2005)
14. Milner, R.: An algebraic definition of simulation between programs. Proceedings of the 2nd International Joint Conference on Artificial Intelligence (1971)
15. Misra, J.: Distributed discrete-event simulation. ACM Computing Survey (1986)
16. Namjoshi, K.S.: A simple characterization of stuttering bisimulation. In: Foundations of Software Technology and Theoretical Computer Science. Springer (1997)
17. Namjoshi, K.S., Zuck, L.D.: Witnessing program transformations. Static Analysis Symposium (2013)
18. Ray, S., Jr., W.A.H.: Deductive verification of pipelined machines using first-order quantification. In: CAV 2004 (2004)
19. Velev, M., Bryant, R.: Formal verification of superscale microprocessors with multicycle functional units, exception, and branch prediction. In: Proceedings of the 37th Annual Design Automation Conference (2000)

A Proofs for Section 3 (Theory of Skipping Refinement)

Definition 9 Simulation [14]. *R is a simulation relation of a transition system $\mathcal{M} = \langle S, \to, L\rangle$ iff $R \subseteq S \times S$ and for all $s, w \in S$ such that sRw, both of the following conditions hold.*

(SIM1) $L.s = L.w$
(SIM2) $\langle \forall u : s \to u : \langle \exists v : w \to v : uRv\rangle\rangle$

Definition 10 match [11]. *Let $\mathcal{M} = \langle S, \to, L\rangle$ be a transition system, σ, δ be fullpaths in $\mathcal{M}$. For $\pi, \xi \in INC$ and binary relation $B \subseteq S \times S$, we define*

$scorr(B, \sigma, \pi, \delta, \xi) \equiv \langle \forall i \in \omega :: \langle \forall s \in {}^{\pi}\sigma^{i} \wedge w \in {}^{\xi}\delta^{i} :: sBw\rangle\rangle$ *and*
$match(B, \sigma, \delta) \equiv \langle \exists \pi, \xi \in INC :: scorr(B, \sigma, \pi, \delta, \xi)\rangle.$

Definition 11 Stuttering Simulation [11]. *B is a stuttering simulation (STS) relation of a transition system $\mathcal{M} = \langle S, \to, L\rangle$ iff $B \subseteq S \times S$ and for all $s, w \in S$ such that sBw, both of the following conditions hold.*

(STS1) $L.s = L.w$
(STS2) $\langle \forall \sigma : fp.\sigma.s : \langle \exists \delta : fp.\delta.w : match(B, \sigma, \delta)\rangle\rangle$

Theorem 1. *Let $\mathcal{M}$ be a transition system.*

1. *If B is a simulation on $\mathcal{M}$ then B is a stuttering simulation on $\mathcal{M}$.*
2. *If B is a stuttering simulation on $\mathcal{M}$ then B is an SKS on $\mathcal{M}$.*

Proof. Follows directly from the definitions of SKS (Definition 3), *smatch* (Definition 2), STS(Definition 11), and *match*(Definition 10). □

Lemma 1. *Let $\mathcal{M}$ be a TS and $\mathcal{C}$ be a set of SKS's on $\mathcal{M}$. Then $G = \langle \cup B : B \in \mathcal{C} : B\rangle$ is an SKS on $\mathcal{M}$.*

Proof. Let $s, w \in S$ and sGw. We show that SKS1 and SKS2 holds for G. Since $G = \langle \cup B : B \in \mathcal{C} : B\rangle$, there is an SKS $B \in \mathcal{C}$ such that sBw. Hence $L.s = L.w$. SKS1 holds for G. Next since B is an SKS on $\mathcal{M}$, SKS2 holds for B, *i.e.*, for any fullpath σ starting at s, there is a fullpath δ starting at w such that $smatch(B, \sigma, \delta)$ holds. From definition of *smatch*, there exists $\pi, \xi \in INC$ such that for all $i \in \omega$ and for all $s \in {}^{\pi}\sigma^{i}$ we have that $sB\delta(\xi.i)$. Since $B \subseteq G$, $sG\delta(\xi.i)$ holds. Hence $smatch(G, \sigma, \delta)$ holds, *i.e.*, SKS2 holds for G. □

Corollary 1. *For any TS $\mathcal{M}$, there is a greatest SKS on $\mathcal{M}$.*

Proof. Let $\mathcal{C}$ be a set of SKS's on $\mathcal{M}$ and $G = \langle \cup B : B \in \mathcal{C} : B\rangle$. G is the greatest and from Lemma 1, G is an SKS. □

Lemma 2. *SKS are not closed under negation and intersection.*

Proof. Consider a TS $\mathcal{M} = \langle S = \{a, b\}, \rightarrow = \{(a, a), (b, b)\}, L = \{(a, 1), (b, 2)\}\rangle$. The indentity relation is an SKS, but its negation is not.

A counter example to show that SKS is not closed under intersection appears in Figure 2.

□

Theorem 2. *The reflexive transitive closure of an SKS is an SKS.*

Proof. Let $\mathcal{M} = \langle S, \rightarrow, L\rangle$ be a TS and B be an SKS on $\mathcal{M}$. The reflexive, transitive closure of B, written B^*, is $\langle \cup i \in \omega :: B^i\rangle$. First we show that for all $i \in \omega$, B^i is an SKS using induction on natural numbers. In the base case, B^0, the identity relation, is clearly an SKS. For $i \geq 0$, we have that $B^{i+1} = B; B^i$; from Lemma 3 and the induction hypothesis, we have that B^{i+1} is an SKS on $\mathcal{M}$. Finally, from Lemma 1, we have that $\langle \cup i \in \omega :: B^i\rangle$, *i.e.*, B^* is an SKS on $\mathcal{M}$. □

Theorem 3. *Given a TS $\mathcal{M}$, the greatest SKS on $\mathcal{M}$ is a preorder.*

Proof. Let G be the greatest SKS on $\mathcal{M}$. From Theorem 2, G^* is an SKS. Hence $G^* \subseteq G$. Furthermore, since $G \subseteq G^*$, we have that $G = G^*$, *i.e.*, G is reflexive and transitive.

□

The following lemma is useful for lifting the notion of skipping simulation to the notion of skipping refinement, a notion relating two transition systems at different levels of abstraction.

Lemma 6. *Let S, S_1, S_2 be a set of states such that $S_1 \cap S_2 = \varnothing$ and $S_1 \cup S_2 \subseteq S$. Let B be an SKS on $\mathcal{M} = \langle S, \rightarrow, L\rangle$ such that any state in S_1 can only reach states in S_1, and any state in S_2 can only reach states in S_2, then $B' = \{(s_1, s_2) \mid s_1 \in S_1 \wedge s_2 \in S_2 \wedge s_1 B s_2\}$ is an SKS on $\mathcal{M}$.*

Proof. Let $s_1 B' s_2$. We show that SKS1 and SKS2 holds for B'. From definition of B', we have that $s_1 \in S_1$, $s_2 \in S_2$, and $s_1 B s_2$. Since B is an SKS on $\mathcal{M}$, we have that $L.s_1 = L.s_2$; hence SKS1 holds for B'. Next let σ and δ be fullpaths in $\mathcal{M}$ starting at s_1 and s_2 respectively and $\pi, \xi \in INC$ such that $\langle \forall i \in \omega :: \langle \forall s \in {}^{\pi}\sigma^i :: sB\delta(\xi.i)\rangle\rangle$ holds. Next from the assumptions that any state in S_1 can only reach states in S_1, and σ is a fullpath in $\mathcal{M}$ starting at $s_1 \in S_1$, all states in ${}^{\pi}\sigma^i$ are in S_1. Also, since any state in S_2 can only reach states in S_2, state $\delta(\xi.i) \in S_2$. Hence we have that $\langle \forall i \in \omega :: \langle \forall s \in {}^{\pi}\sigma^i :: sB'\delta(\xi.i)\rangle\rangle$, *i.e.*, SKS2 holds for B'.

□

Theorem 5. *Let $\mathcal{M} = \langle S, \rightarrow, L\rangle$ be a transition system. Let $\mathcal{M}' = \langle S', \rightarrow', L\rangle$ where $S' \subseteq S$, $\rightarrow' \subseteq S' \times S'$, $\rightarrow'$ is a left-total subset of $\rightarrow^+$, and $L' = L|_{S'}$. Then $\mathcal{M}' \lesssim_I \mathcal{M}$, where I is the identity function on S'.*

Proof. Let B be I. Let σ be a fullpath starting at an $\mathcal{M}'$ state. To show that B is an SKS relation, the key observation is that since $\rightarrow' \subseteq \rightarrow^+$, there is a fullpath starting from the correspodning $\mathcal{M}$ state, say δ, such that a step in σ corresponds to a finite, positive number of steps in δ. We choose such a fullpath δ as a witness and show σ and δ smatch under B. We consider the partitioning of σ such that a partition has only one state. Next we define the partitioning of δ. The i^{th} partition of δ includes (1) the state, say s, in $\mathcal{M}$ corresponding to the state in the i^{th} partition of σ, and (2) intermediate states in $\mathcal{M}$ required to reach from s to the state in $\mathcal{M}$ corresponding to the state in the $(i+1)^{th}$ partition of σ. It is easy to see that σ, δ and their partitions defined above satisfy *scorr* in Definition 2.

$\square$

Corollary 2. *Let $\mathcal{M}_C = \langle S_C, \xrightarrow{C}, L_C \rangle$ and $\mathcal{M}_A = \langle S_A, \xrightarrow{A}, L_A \rangle$ be transition systems, $r : S_C \rightarrow S_A$ be a refinement map. Let $\mathcal{M}'_C = \langle S'_C, \xrightarrow{C}', L'_C \rangle$ where $S'_C \subseteq S_C$, $\xrightarrow{C}'$ is a left-total subset of $\xrightarrow{C}^+$, and $L'_C = L_C|_{S'_C}$. If $\mathcal{M}_C \lesssim_r \mathcal{M}_A$ then $\mathcal{M}'_C \lesssim_{r'} \mathcal{M}_A$, where r' is $r|_{S'_C}$.*

Proof. From Theorem 5, $\mathcal{M}'_C \lesssim_I \mathcal{M}_C$, where I is the identity function on $S_{C'}$. Since $\mathcal{M}_C \lesssim_r \mathcal{M}_A$, from Theorem 4, we have that $\mathcal{M}'_C \lesssim_{I;r} \mathcal{M}_A$, *i.e.*, $\mathcal{M}'_C \lesssim_{r'} \mathcal{M}_A$. $\square$