Liveness-preserving Simulation Relations

Paul C. Attie *
School of Computer Science
Florida International University
attie@fiu.edu

Abstract

We present a simulation-based proof method for *liveness* properties. Our method is based on simulation relations [8] that relate the liveness properties of an implementation to those of the specification. Even though reasoning about liveness is usually associated with reasoning over entire executions, variant functions, fairness etc., our method requires reasoning over individual states/transitions only. It thus presents a significant methodological advance over current methods.

1 Introduction

One of the main proof techniques for concurrent systems to emerge recently is that of establishing a *simulation* [8] between a specification and its implementation. In this approach, the specification and implementation are uniformly expressed in an executable notation, e.g., as automata, and a relation between the states/transitions of the two automata is established[1]. The relation is called a simulation if a certain correspondence (depending on the precise type of simulation being established) between the states/transitions of the implementation and the states/transitions of the specification is established. An important consequence of establishing a simulation between implementation A and specification B is that it immediately follows that any externally observable behavior ("trace") of A is also an externally observable behavior of B. This notion, called *trace inclusion*, is taken as the technical embodiment of the idea that A is a "correct implementation" of B.

An important advantage of the simulation-based approach is that it only requires reasoning about individual states/transitions, rather than reasoning about entire executions. Unfortunately, the end-result, namely the establishment of trace inclusion, does not allow us to conclude that liveness properties are preserved.

In this paper, we present an approach for preserving liveness properties that requires reasoning only about individual states/transitions. Our approach uses a state-based technique to specify liveness properties: a liveness property is given by the acceptance condition of a *complemented-pairs (or Streett) automaton* [2, 4]. We then present the notion of *liveness-preserving forward simulation*, which appropriately relates the states mentioned in the implementation's liveness property to those mentioned in the specification's liveness property. Establishing a liveness-preserving forward simulation then allows us to conclude that every "live trace"[2] of the implementation is also a live trace of the specification.

The rest of the paper is organized as follows. Section 2 provides technical background from [6] and [8]. Section 3 presents our definitions for state-based liveness properties and liveness-preserving forward simulation, along with our main result—the soundness of liveness-preserving forward simulation with respect to live trace inclusion. Section 4 applies our results to the eventually-serializable data service of [5]. Section 5 discusses related work. Finally, Section 6 presents our conclusions and discusses avenues for further research.

2 Technical Background

The definitions and theorems below are taken from [6] and [8], to which the reader is referred for details.

Definition 1 (Automaton) *An* automaton A *consists of four components:*

1. *a set* $states(A)$ *of states,*
2. *a nonempty set* $start(A) \subseteq states(A)$ *of start states,*
3. *an action signature* $sig(A) = (ext(A), int(A))$ *where* $ext(A)$ *and* $int(A)$ *are disjoint sets of external and internal actions, respectively (denote by* $acts(A)$ *the set* $ext(A) \cup int(A)$*), and*

*Supported in part by NSF CAREER Grant CCR-9702616 and AFOSR Grant F49620-96-1-0221.

[1] We sometimes refer to the specification as the *abstract* automaton and the implementation as the *concrete* automaton.

PODC'99 Atlanta GA USA

[2] i.e., the externally observable behavior of an execution that satisfies the complemented-pairs liveness condition.

4. *a relation* $steps(A) \subseteq states(A) \times acts(A) \times states(A)$ *giving the transitions of* A.

Let $s, s', u, u', \ldots$ range over states and $a, b, \ldots$ range over actions. Write $s \xrightarrow{a}_A s'$ iff $(s, a, s') \in steps(A)$. We say that a is *enabled* in s. An execution fragment α of automaton A is an alternating sequence of states and actions $s_0 a_1 s_1 a_2 s_2 \ldots$ such that $(s_i, a_{i+1}, s_{i+1}) \in steps(A)$ for all $i \geq 0$, i.e., α conforms to the transition relation of A. Furthermore, if α is finite then it ends in a state. If α is an execution fragment, then $fstate(\alpha)$ is the first state along α, and if α is finite, then $lstate(\alpha)$ is the last state along α. If α_1 is a finite execution fragment, α_2 is an execution fragment, and $lstate(\alpha_1) = fstate(\alpha_2)$, then $\alpha_1 \frown \alpha_2$ is the concatenation of α_1 and α_2 (with $lstate(\alpha_1)$ repeated only once). Let $\alpha = s_0 a_1 s_1 a_2 s_2 \ldots$ be an execution fragment. Then the length of α, denoted $|\alpha|$, is the number of actions in α, and is infinite for infinite execution fragments. Also, $\alpha|_i \stackrel{\text{df}}{=} s_0 a_1 s_1 \ldots a_i s_i$. If α is a prefix of α', we write $\alpha \leq \alpha'$. We also write $\alpha < \alpha'$ for $\alpha \leq \alpha'$ and $\alpha \neq \alpha'$.

An execution of A is an execution fragment that begins with a state in $start(A)$. The trace $trace(\alpha)$ of execution fragment α is obtained by taking the sequence of all the actions of α and removing the internal actions. $traces(A)$ is the set of traces β such that β is the trace of some execution of A. If L is a set of executions, then $traces(L)$ is the set of traces β such that β is the trace of some execution in L. $exec(A)$ is the set of all executions of A.

If R is a relation over $S_1 \times S_2$ (i.e., $R \subseteq S_1 \times S_2$) and $s_1 \in S_1$, then $R[s_1] \stackrel{\text{df}}{=} \{s_2 \mid s_2 \in S_2 \text{ and } (s_1, s_2) \in R\}$.

2.1 Simulation Relations

Definition 2 (Forward Simulation) *Let A and B be automata with the same external actions. A* forward simulation *from A to B is a relation f over $states(A) \times states(B)$ that satisfies:*

1. *If $s \in start(A)$, then $f[s] \cap start(B) \neq \emptyset$.*
2. *If $s \xrightarrow{a}_A s'$ and $u \in f[s]$, then there exists a finite execution fragment α of B such that $fstate(\alpha) = u$, $lstate(\alpha) \in f[s']$, and $trace(\alpha) = trace(a)$.*

Simulation based proof methods typically use *invariants* to restrict the steps that have to be considered. An invariant of an automaton is a predicate that holds in all of its reachable states, or alternatively, is a superset of the reachable states.

Definition 3 (Forward Simulation w.r.t. Invariants) *Let A and B be automata with the same external actions and with invariants I_A, I_B, respectively. A* forward simulation *from A to B with respect to I_A and I_B is a relation f over $states(A) \times states(B)$ that satisfies:*

1. *If $s \in start(A)$, then $f[s] \cap start(B) \neq \emptyset$.*
2. *If $s \xrightarrow{a}_A s'$, $s \in I_A$, and $u \in f[s] \cap I_B$, then there exists a finite execution fragment α of B such that $fstate(\alpha) = u$, $lstate(\alpha) \in f[s']$, and $trace(\alpha) = trace(a)$.*

We write $A \leq_F B$ if there exists a forward simulation from A to B, and $A \leq_F B$ via f if f is a forward simulation from A to B. A forward simulation requires that (1) each execution of an external action a of A is matched by a finite execution fragment of B containing a, and all of whose other actions are internal to B, and (2) each execution of an internal action of A is matched by a finite (possibly empty) execution fragment of B all of whose actions are internal to B (if the fragment is empty, then we have $u \in f[s']$, i.e., u and s' must be related by the simulation). It follows that forward simulation implies trace inclusion (also referred to as the *safe preorder* below).

2.2 Execution Correspondence

Simulation relations induce a correspondence between the executions of the concrete and the abstract automata. This correspondence is captured by the notion of R-relation.

Definition 4 (R-relation, Index Mapping) *Let A and B be automata with the same external actions and let R be a relation over $states(A) \times states(B)$. Furthermore, let α and α' be executions of A and B, respectively:*

$$\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$$
$$\alpha' = u_0 b_1 u_1 b_2 u_2 \cdots$$

Say that α and α' are R-related, *written $(\alpha, \alpha') \in R$, if there exists a total, nondecreasing mapping $m : \{0, 1, \ldots, |\alpha|\} \mapsto \{0, 1, \ldots, |\alpha'|\}$ such that:*

1. *$m(0) = 0$,*
2. *$(s_i, u_{m(i)}) \in R$ for all $0 \leq i \leq |\alpha|$,*
3. *$trace(b_{m(i-1)+1} \cdots b_{m(i)}) = trace(a_i)$ for all $0 < i \leq |\alpha|$, and*
4. *for all $j, 0 \leq j \leq |\alpha'|$, there exists an i, $0 \leq i \leq |\alpha|$, such that $m(i) \geq j$.*

The mapping m is referred to as an index mapping *from α to α' with respect to R. Write $(A, B) \in R$ if for every execution α of A, there exists an execution α' of B such that $(\alpha, \alpha') \in R$.*

Lemma 1 *Let A and B be automata with the same external actions, and such that $A \leq_F B$ via f for some forward simulation f. Let α be an arbitrary execution of A. Then there exists a collection $(\alpha'_i, m_i)_{0 \leq i \leq |\alpha|}$ of finite executions of B and mappings such that*

1. *m_i is an index mapping from $\alpha|_i$ to α'_i with respect to f, for all $0 \leq i \leq |\alpha|$, and*
2. *$\alpha'_{i-1} \leq \alpha'_i$ and $m_{i-1} = m_i \upharpoonright \{0, \ldots, i-1\}$ for all $0 < i \leq |\alpha|$.*

Theorem 2 (Execution Correspondence Theorem) *Let A and B be automata with the same external actions. If $A \leq_F B$ via f, then $(A, B) \in f$.*

Lemma 3 *Let A and B be automata with the same external actions and let R be a relation over $states(A) \times states(B)$. If $(\alpha, \alpha') \in R$, then $trace(\alpha) = trace(\alpha')$.*

2.3 Temporal Logic

We use a fragment of linear-time temporal logic [3] to express some properties of executions. In particular, we use the "infinitary" operators $\Box\Diamond$ (infinitely often) and $\Diamond\Box$ (eventually always).

Definition 5 *If U is a set of states, then $\alpha \models \Box\Diamond U$ means "α contains infinitely many states from U," and $\alpha \models \Diamond\Box U$ means "all but a finite number of states of α are from U." The operators can be combined with propositional connectives, so that, for example, $\alpha \models \Box\Diamond U' \Rightarrow \Box\Diamond U''$ means "if α contains infinitely many states from U', then it also contains infinitely many states from U'', and $\alpha \models \Diamond\Box\neg U$ means "all but a finite number of states of α are not from U."*

3 Simulation Relations for Liveness

The safe preorder, live preorder [6] embody our notions of correct implementation with respect to safety, liveness, respectively.

Definition 6 (Safe preorder, Live preorder) *Let A_1, A_2 be automata such that $ext(A_1) = ext(A_2)$, and let L_1, L_2 be sets of executions of A_1, A_2 respectively that give the liveness properties of A_1, A_2. We define:*

Safe preorder: $(A_1, L_1) \sqsubseteq_s (A_2, L_2)$ *if and only if* $traces(A_1) \subseteq traces(A_2)$

Live preorder: $(A_1, L_1) \sqsubseteq_\ell (A_2, L_2)$ *if and only if* $traces(L_1) \subseteq traces(L_2)$

From [8], we have that forward simulation implies the safe preorder, i.e., if $A_1 \leq_F A_2$, then $(A_1, L_1) \sqsubseteq_s (A_2, L_2)$. Our goal is a proof method for establishing the live preorder using reasoning over states/transitions only (in particular, avoiding reasoning over entire executions). We therefore formulate a liveness condition based on states rather than executions.

Definition 7 (Live Automaton) *A* live automaton *is a pair (A, L) where:*

1. *A is an automaton and,*
2. *L is a* complemented-pairs liveness condition *for A, that is, a countable set of pairs $\{\langle \mathsf{Red}^1_A, \mathsf{Green}^1_A \rangle, \ldots, \langle \mathsf{Red}^i_A, \mathsf{Green}^i_A \rangle, \ldots\}$ where $\mathsf{Red}^i_A \subseteq states(A)$ and $\mathsf{Green}^i_A \subseteq states(A)$ for all $i \geq 1$.*

The executions (execution fragments) of (A, L) are the executions (execution fragments) of A, respectively. The live executions *of (A, L) are all the infinite executions α of A such that, for all $i \geq 1$, if α contains infinitely many states in Red^i_A, then it also contains infinitely many states in Green^i_A. In addition, L satisfies the requirement that every finite execution can be extended to a live execution.*[3]

[3] We shall refer to this requirement as the *extensibility requirement for finite executions*. It was proposed in [1], where it is called *machine closure*.

The complemented-pairs liveness condition is essentially the acceptance condition for finite-state complemented-pairs automata on infinite strings [4], with the important difference that we generalize it to an arbitrary state space, and allow a possibly infinite set of pairs. In the sequel, we view Red^i_A, Green^i_A, as either subsets of $states(A)$ or as predicates over $states(A)$, as convenient. We also identify the liveness condition L with the set of infinite executions of A that satisfy L.

Our key idea is to refine a single complemented pair in the liveness condition of the abstract automaton into a finite "lattice" of complemented pairs in the liveness condition of the concrete automaton. Let $\prec$ be an irreflexive partial order over a set P of complemented pairs[4]. If $p \in P$, define $succ(p) = \{q \mid p \prec q \wedge \forall r : p \preceq r \prec q \Rightarrow p = r\}$, where $p \preceq q \stackrel{\text{df}}{=} p \prec q$ or $p = q$. Thus, $succ(p)$ is the set of all "immediate successors" of p in $(P, \prec)$. If p is a complemented pair, let $p.\mathsf{Red}$, $p.\mathsf{Green}$ be the Red, Green sets of p, respectively. We now impose two technical conditions: (1) for every pair p, the Green set of p must be a subset of the union of the Red sets of all the immediate successors of p (i.e., the $q \in succ(p)$), and (2) P has a single $\prec$-minimum element $bottom(P)$, and a single $\prec$-maximum element $top(P)$ (the "bottom" and "top" elements of P, respectively). Now let α be an arbitrary live execution. Then, $\alpha \models \Box\Diamond p.\mathsf{Red} \Rightarrow \Box\Diamond p.\mathsf{Green}$ and $\alpha \models \Box\Diamond q.\mathsf{Red} \Rightarrow \Box\Diamond q.\mathsf{Green}$, for all $q \in succ(p)$. Since $succ(p)$ is finite, it follows that, if $p.\mathsf{Green}$ holds infinitely often in α, then $q.\mathsf{Red}$ holds infinitely often in α, for some $q \in succ(p)$. Hence $\alpha \models \Box\Diamond p.\mathsf{Red} \Rightarrow \Box\Diamond \bigcup_{q \in succ(p)} q.\mathsf{Green}$. In effect, we can consider $(p.\mathsf{Red}, \bigcup_{q \in succ(p)} q.\mathsf{Green})$ to be a "virtual" complemented pair. Thus, the $\prec$ ordering provides a way of relating the complemented pairs of P so that the complemented-pairs property (infinitely often Red implies infinitely often Green) can be generalized to encompass a pair and its immediate successor pairs. By starting with the $\prec$-minimum pair $bottom(P)$, and applying the above argument inductively (using $\prec$ as the underlying ordering), we can establish the complemented-pairs property for $(bottom(P).\mathsf{Red}, top(P).\mathsf{Green})$, i.e., $\alpha \models \Box\Diamond bottom(P).\mathsf{Red} \Rightarrow \Box\Diamond top(P).\mathsf{Green}$. This is done in Lemma 4 below.

The remaining step of our refinement is to match $bottom(P).\mathsf{Red}$ to the Red set of the complemented pair being refined, and $top(P).\mathsf{Green}$ to the Green set of the complemented pair being refined. We augment the definition of forward simulation from [8] to perform this matching, resulting in the definition of liveness-preserving forward simulation (Definition 9) below. This matching, together with Lemma 4, allows us to show that every live execution of the concrete automaton has a "corresponding" live execution in the abstract automaton. Live trace inclusion follows immediately.

When constructing a lattice to refine a complemented pair, we can use requirement (1) above ($p.\mathsf{Green} \subseteq \bigcup_{q \in succ(p)} q.\mathsf{Red}$) as a constraint that suggests how to order the complemented pairs of the concrete automaton: $p.\mathsf{Green}$ must imply $\bigvee_{q \in succ(p)} q.\mathsf{Red}$ (viewing $p.\mathsf{Green}$ and $q.\mathsf{Red}$ as predicates).

[4] Following convention, we shall refer to this ordered set simply as P when no confusion arises.

Definition 8 (Complemented-pairs Lattice) *Let L be a complemented-pairs liveness condition. Then $(P, \prec)$ is a* complemented-pairs lattice over L *iff*[5]

1. *P is a finite subset of L,*
2. *$\prec$ is an irreflexive partial order over P,*
3. *P contains an element $top(P)$ which satisfies $\forall p \in P : p \preceq top(P)$, and an element $bottom(P)$ which satisfies $\forall p \in P : bottom(P) \preceq p$, and*
4. *$\forall p \in P - \{top(P)\} : p.\mathsf{Green} \subseteq \bigcup_{q \in succ(p)} q.\mathsf{Red}$.*

The elements $top(P)$ and $bottom(P)$ are necessarily unique. Let $lattices(L)$ denote the set of all complemented-pairs lattices over L.

Lemma 4 *Let (A, L) be a live automaton, $(P, \prec) \in lattices(L)$, $\bot = bottom(P)$, and $\top = top(P)$. Let α be an arbitrary live execution of (A, L). Then $\alpha \models \Box\Diamond\bot.\mathsf{Red} \Rightarrow \Box\Diamond\top.\mathsf{Green}$.*

Proof. Let α be an arbitrary live execution of (A, L). We assume $\alpha \models \Box\Diamond\bot.\mathsf{Red}$ and establish $\alpha \models \Box\Diamond\top.\mathsf{Green}$. First, we establish:

> If $p \in P$, $p \neq \top$, and $\alpha \models \Box\Diamond p.\mathsf{Red}$, then $\alpha \models \Box\Diamond q.\mathsf{Red}$ for some $q \in succ(p)$. (*)

Proof of (*): Assume the antecedent of (*). Since α is live, $\alpha \models \Box\Diamond p.\mathsf{Red} \Rightarrow \Box\Diamond p.\mathsf{Green}$ by Definition 7. Hence $\alpha \models \Box\Diamond p.\mathsf{Green}$. By Definition 8, $p.\mathsf{Green} \subseteq \bigcup_{q \in succ(p)} q.\mathsf{Red}$. Hence $\alpha \models \Box\Diamond \bigcup_{q \in succ(p)} q.\mathsf{Red}$. Since P is finite, $succ(p)$ is finite. It follows that $\alpha \models \Box\Diamond q.\mathsf{Red}$ for some $q \in succ(p)$. (End of proof of (*).)

We now construct a sequence $q_1, q_2, \ldots, q_i, \ldots$ of complemented pairs in P such that $\forall i \geq 1 : \alpha \models \Box\Diamond q_i.\mathsf{Red}$. We let $q_1 = \bot$, noting that $\alpha \models \Box\Diamond\bot.\mathsf{Red}$ by assumption. We derive q_{i+1} by applying (*) to q_i. Suppose $\top$ is not in $q_1, q_2, \ldots$ Then (*) can be applied indefinitely. Since $q_{i+1} \in succ(q_i)$, it follows that $q_j \prec q_{i+1}$ for all $j \in 1..i$. Hence $q_j \neq q_{i+1}$ for all $j \in 1..i$. Thus $q_1, q_2, \ldots$ is an infinite sequence of pairwise different complemented pairs in P. But this is impossible, since P is finite. Hence the assumption that $\top$ is not in $q_1, q_2, \ldots$ is false. By (*), we have $\forall i \geq 1 : \alpha \models \Box\Diamond q_i.\mathsf{Red}$, and so $\alpha \models \Box\Diamond\top.\mathsf{Red}$. Since α is live, $\alpha \models \Box\Diamond\top.\mathsf{Red} \Rightarrow \Box\Diamond\top.\mathsf{Green}$. Hence we have $\alpha \models \Box\Diamond\top.\mathsf{Green}$. □

As stated above, each complemented pair q of the liveness condition of the abstract automaton is refined into a complemented-pairs lattice P over the liveness condition of the concrete automaton. Our liveness preserving forward simulation extends the forward simulation of [8] by relating q to P: an occurrence of a $q.\mathsf{Red}$ state in an abstract (live) execution α' must be matched by at least one $bottom(P).\mathsf{Red}$ state in the corresponding concrete (live) execution α, and an occurrence of a $top(P).\mathsf{Green}$ state in α must be matched by at least one $q.\mathsf{Green}$ state in α'. Together with Lemma 4, this allows us to show that if an abstract and concrete execution correspond (according to the simulation), and the concrete execution is live, then the abstract execution is also live. Specifically, if $\alpha' \models \Box\Diamond q.\mathsf{Red}$, then $\alpha \models \Box\Diamond bottom(P).\mathsf{Red}$. Hence, by Lemma 4, $\alpha \models \Box\Diamond top(P).\mathsf{Green}$. Hence $\alpha' \models \Box\Diamond q.\mathsf{Green}$. Hence α' is live.

Definition 9 (Liveness-preserving Forward Simulation w.r.t. Invariants) *Let (A, L) and (B, M) be live automata with the same external actions. Let I_A, I_B be invariants of A, B respectively. Let $f = (g, h)$ be such that $g \subseteq states(A) \times states(B)$ and $h : M \mapsto lattices(L)$ is a total mapping over M*[6]*. Then f is a* liveness-preserving forward simulation *from (A, L) to (B, M) with respect to I_A and I_B iff:*

1. *If $s \in start(A)$, then $g[s] \cap start(B) \neq \emptyset$.*
2. *If $s \xrightarrow{a}_A s'$, $s \in I_A$, and $u_1 \in g[s] \cap I_B$, then there exists a finite execution fragment $u_1 \xrightarrow{b_1}_B u_2 \xrightarrow{b_2}_B u_3 \cdots \xrightarrow{b_{n-1}}_B u_n$ $(n \geq 1)$ such that $u_n \in g[s']$ and $trace(b_1, \ldots, b_{n-1}) = trace(a)$. Furthermore, for all $q \in M$, let $\bot_q = bottom(h(q))$ and $\top_q = top(h(q))$. Then*
   (a) *if $(\exists i \in 1..n : u_i \in q.\mathsf{Red})$ then $s \in \bot_q.\mathsf{Red}$ or $s' \in \bot_q.\mathsf{Red}$, and*
   (b) *if $s \in \top_q.\mathsf{Green}$ or $s' \in \top_q.\mathsf{Green}$ then $(\exists i \in 1..n : u_i \in q.\mathsf{Green})$.*
3. *Call a transition $s \xrightarrow{a}_A s'$* silent *iff $s \in I_A$ and for every finite execution fragment $u_1 \xrightarrow{b_1}_B u_2 \xrightarrow{b_2}_B u_3 \cdots \xrightarrow{b_{n-1}}_B u_n$ such that $u_1 \in g[s] \cap I_B$, $u_n \in g[s']$, and $trace(b_1, \ldots, b_{n-1}) = trace(a)$, we have $n = 1$. In other words, the transition $s \xrightarrow{a}_A s'$ is matched only by the empty transition in B. Then, g is such that every live execution of (A, L) contains an infinite number of non-silent transitions.*

Clause 3 is needed to ensure that a live execution of (A, L) has at least one corresponding infinite execution in (B, M). This execution can then be shown, using clause 2, to be live (see Lemma 6 below). If $s \xrightarrow{a}_A s'$ is silent, then a must be an internal action. Thus, in practice, clause 3 holds, since executions with an (infinite) suffix consisting solely of internal actions are not usually considered to be live. Clause 3 can itself be expressed as a complemented pair (which is added to L). Call an action of A non-silent iff all of the transitions arising from its execution are non-silent. By definition, any external action of A is non-silent. An internal action of A may or may not be non-silent, depending on the particular value of g. We introduce an auxiliary boolean variable *nonsilent* that is set to *true* each time a non-silent action of A is executed, and is set to *false* infinitely often by a new internal action of A whose precondition is *true* and whose effect is *nonsilent* := *false* (every execution of this new action can be simulated by the empty transition in B, since *nonsilent* has no effect on any other state component of A, nor on the execution

[5] Note that we use the term "lattice" in an informal sense, since our complemented-pairs lattices do not satisfy the mathematical definition of a lattice.

[6] That is, $h(q)$ is defined for all $q \in M$.

of other actions in A). Then the pair $\langle\!\langle true, nonsilent \rangle\!\rangle$ expresses that a non-silent action of A is executed infinitely often, which implies that each live execution of (A, L) contains an infinite number of non-silent transitions. The pair $\langle\!\langle true, nonsilent \rangle\!\rangle$ can then be refined at the next lower level (of refinement) in exactly the same way as all the other pairs in L. See Section 4 for an example of this technique.

It is clear that if (g, h) is a liveness-preserving forward simulation from A to B, then g is a forward simulation from A to B. We write $(A, L) \leq_F^\ell (B, M)$ if there exists a liveness-preserving forward simulation from (A, L) to (B, M), and $(A, L) \leq_F^\ell (B, M)$ via f if f is a liveness-preserving forward simulation from (A, L) to (B, M).

Liveness-preserving simulation relations induce a correspondence between the live executions of the concrete and the abstract automata. This correspondence is captured by the notion of R^ℓ-relation.

Definition 10 (R^ℓ-relation, Live Index Mapping) *Let (A, L) and (B, M) be live automata with the same external actions. Let $R^\ell = (R, H)$ where R is a relation over states$(A) \times$ states(B) and $H : M \mapsto$ lattices(L) is a total mapping over M. Furthermore, let α and α' be executions of (A, L) and (B, M), respectively:*

$$\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$$
$$\alpha' = u_0 b_1 u_1 b_2 u_2 \cdots$$

*Say that α and α' are R^ℓ-*related*, written $(\alpha, \alpha') \in R^\ell$, if there exists a total, nondecreasing mapping $m : \{0, 1, \ldots, |\alpha|\} \mapsto \{0, 1, \ldots, |\alpha'|\}$ such that:*

1. *$m(0) = 0$,*
2. *$(s_i, u_{m(i)}) \in R$ for all $0 \leq i \leq |\alpha|$,*
3. *$trace(b_{m(i-1)+1} \cdots b_{m(i)}) = trace(a_i)$ for all $0 < i \leq |\alpha|$,*
4. *for all $j, 0 \leq j \leq |\alpha'|$, there exists an i, $0 \leq i \leq |\alpha|$, such that $m(i) \geq j$, and*
5. *for all complemented pairs $q \in M$ and all $0 < i \leq |\alpha|$:*
   (a) *If $(\exists j \in m(i-1)..m(i) : u_j \in q.\mathsf{Red})$ then $s_{i-1} \in \bot.\mathsf{Red}$ or $s_i \in \bot.\mathsf{Red}$.*
   (b) *If $s_{i-1} \in \top.\mathsf{Green}$ or $s_i \in \top.\mathsf{Green}$ then $(\exists j \in m(i-1)..m(i) : u_j \in q.\mathsf{Green})$.*

   where $\bot = bottom(H(q))$, $\top = top(H(q))$.

The mapping m is referred to as a live index mapping *from α to α' with respect to R^ℓ. Write $((A, L), (B, M)) \in R^\ell$ if for every live execution α of (A, L), there exists a live execution α' of (B, M) such that $(\alpha, \alpha') \in R^\ell$.*

Note that $(\alpha, \alpha') \in R^\ell$ does not require α, α' to be live executions. By Definitions 4 and 10, it is clear that, if $R^\ell = (R, H)$, then $(\alpha, \alpha') \in R^\ell$ implies $(\alpha, \alpha') \in R$.

The following lemma establishes a correspondence between the prefixes of a live execution of the concrete automaton and an infinite family of finite executions of the abstract automaton. It is analogous to Lemma 1, except that we restrict to live executions of the concrete automaton (A, L), and show that the corresponding collection of execution/mapping pairs must be infinite.

Lemma 5 *Let (A, L) and (B, M) be live automata with the same external actions, and such that $(A, L) \leq_F^\ell (B, M)$ via f for some $f = (g, h)$. Let α be an arbitrary live execution of (A, L). Then there exists a collection $(\alpha'_i, m_i)_{0 \leq i}$ of finite executions of (B, M) and mappings such that:*

1. *m_i is a live index mapping from $\alpha|_i$ to α'_i with respect to f, for all $i \geq 0$, and*
2. *$\alpha'_{i-1} \leq \alpha'_i$ and $m_{i-1} = m_i \restriction \{0, \ldots, i-1\}$ for all $i > 0$, and*
3. *$\alpha'_{i-1} < \alpha'_i$ for infinitely many $i > 0$.*

Proof. Let $\alpha = s_0 a_1 s_1 a_2 s_2 \ldots$ and let I_A, I_B be invariants of A, B, respectively, such that f is a liveness-preserving forward simulation from (A, L) to (B, M) with respect to I_A and I_B. We construct α'_i and m_i by induction on i.

Since $s_0 \in start(A)$, we have $(s_0, u_0) \in g$ and $u_0 \in start(B)$ for some state u_0, by Definition 9, clause 1. Let $\alpha'_0 = u_0$ and let m_0 be the mapping that maps 0 to 0. Then, m_0 is a live index mapping from $\alpha|_0$ to α'_0 with respect to f (in particular, clause 5 of Definition 10 holds vacuously).

Now inductively assume that m_{i-1} (for $i > 0$) is a live index mapping from $\alpha|_{i-1}$ to α'_{i-1} with respect to f. Let $u_1 = lstate(\alpha'_{i-1})$. Then, by clause 4 of Definition 10 and the fact that m_{i-1} is nondecreasing, we have $m_{i-1}(i-1) = |\alpha'_{i-1}|$ and $(s_{i-1}, u_1) \in g$. Since s_{i-1}, s_i, and u_1 are reachable, by definition, they satisfy their respective invariants. Hence, by Definition 9, clause 2, there exists a finite execution fragment $u_1 \xrightarrow{b_1}_B u_2 \xrightarrow{b_2}_B u_3 \cdots \xrightarrow{b_{n-1}}_B u_n$ of B such that $u_n \in g[s_i]$, $trace(b_1, \ldots, b_{n-1}) = trace(a)$, and for all complemented pairs $q \in M$:

1. if $(\exists j \in 1..n : u_j \in q.\mathsf{Red})$ then $s_{i-1} \in \bot.\mathsf{Red}$ or $s_i \in \bot.\mathsf{Red}$, and
2. if $s_{i-1} \in \top.\mathsf{Green}$ or $s_i \in \top.\mathsf{Green}$ then $(\exists j \in 1..n : u_j \in q.\mathsf{Green})$,

where $\bot = bottom(h(q))$, $\top = top(h(q))$. Now define $\alpha'_i = \alpha'_{i-1} \frown (u_1 \xrightarrow{b_1}_B u_2 \xrightarrow{b_2}_B u_3 \cdots \xrightarrow{b_{n-1}}_B u_n)$, and define m_i to be the mapping such that $m_i(j) = m_{i-1}(j)$ for all $0 \leq j \leq i-1$, and $m_i(i) = |\alpha'_i|$. We argue that m_i is a live index mapping from $\alpha|_i$ to α'_i with respect to f, i.e., that all clauses of Definition 10 hold. Clause 1 holds since $m_i(0) = m_{i-1}(0)$ by definition, and $m_{i-1}(0) = 0$ by the inductive hypothesis. Clause 2 holds by the inductive hypothesis and $u_n \in g[s_i]$. Clause 3 holds by the inductive hypothesis and $trace(b_1, \ldots, b_{n-1}) = trace(a)$. Clause 4 holds since $m_i(|\alpha|_i|) = m_i(i) = |\alpha'_i|$, by definition. Finally, clause 5 holds by the inductive hypothesis and the conditions for all complemented pairs $q \in M$ established above. Having established that m_i is a live index mapping from $\alpha|_i$ to α'_i with respect to f, we conclude that clause 1 of the lemma holds.

Clause 2 of the lemma holds by construction of α_i and m_i.

By clause 3 of Definition 9, for infinitely many $i > 0$, we can select the finite execution fragment $u_1 \xrightarrow{b_1}_B u_2 \xrightarrow{b_2}_B u_3 \cdots \xrightarrow{b_{n-1}}_B u_n$ so that $n > 1$. Hence, for infinitely many $i > 0$, we have $\alpha'_{i-1} < \alpha'_i$. Thus, clause 3 of the lemma holds. □

Our next lemma shows that, if infinite concrete and abstract executions correspond (in the sense of $(\alpha, \alpha') \in R^\ell$), and the concrete execution is live, then so is the abstract execution.

Lemma 6 *Let (A, L) and (B, M) be live automata with the same external actions. Let $R^\ell = (R, H)$ where R is a relation over states$(A) \times$ states(B) and $H : M \mapsto$ lattices(L) is a total mapping over M. Let α, α' be arbitrary infinite executions of (A, L), (B, M) respectively. If $(\alpha, \alpha') \in R^\ell$, then $(\alpha \in L \Rightarrow \alpha' \in M)$.*

Proof. We assume the antecedents of the lemma and establish $\alpha' \notin M \Rightarrow \alpha \notin L$. Let:

$$\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$$
$$\alpha' = u_0 b_1 u_1 b_2 u_2 \cdots$$

Since $(\alpha, \alpha') \in R^\ell$, there exists a live index mapping $m : \{0, 1, \ldots, |\alpha|\} \mapsto \{0, 1, \ldots, |\alpha'|\}$ satisfying the conditions in Definition 10. Suppose $\alpha' \notin M$. Then, by Definition 7, there exists a complemented pair q of M such that $\alpha' \models \Box\Diamond q.\mathsf{Red} \wedge \Diamond\Box\neg q.\mathsf{Green}$. Let $\bot = bottom(H(q))$ and $\top = top(H(q))$. We prove:

$$\alpha \models \Box\Diamond \bot.\mathsf{Red} \wedge \Diamond\Box\neg \top.\mathsf{Green}. \qquad (*)$$

Since $\alpha' \models \Box\Diamond q.\mathsf{Red}$, there exist an infinite number of pairs of states $(u_{m(i-1)}, u_{m(i)})$ along α' that contain a $q.\mathsf{Red}$-state between them (inclusive, i.e., the $q.\mathsf{Red}$-state could be $u_{m(i-1)}$ or $u_{m(i)}$). By clauses 2 and 3 of Definition 10, for each such pair there corresponds a pair of states (s_{i-1}, s_i) along α such that $(s_{i-1}, u_{m(i-1)}) \in R$, $(s_i, u_{m(i)}) \in R$, and $trace(b_{m(i-1)+1} \cdots b_{m(i)}) = trace(a_i)$. Hence, by clause 5a of Definition 10, $s_{i-1} \in \bot.\mathsf{Red}$ or $s_i \in \bot.\mathsf{Red}$. Since this holds for an infinite number of values of the index i, we conclude

$$\alpha \models \Box\Diamond \bot.\mathsf{Red}. \qquad (a)$$

Since $\alpha' \models \Diamond\Box\neg q.\mathsf{Green}$, there exists a state u_g along α' such that $\forall \ell \geq g : u_\ell \notin q.\mathsf{Green}$. Now suppose $\alpha \models \Box\Diamond \top.\mathsf{Green}$. Since m is nondecreasing and cofinal in $\{0, 1, \ldots, |\alpha'|\}$ (clause 4, Definition 10), there exists an s_{i-1} along α such that $s_{i-1} \in \top.\mathsf{Green}$ and $m(i-1) \geq g$. By clauses 2 and 3 of Definition 10, $(s_{i-1}, u_{m(i-1)}) \in R$, $(s_i, u_{m(i)}) \in R$, and $trace(b_{m(i-1)+1} \cdots b_{m(i)}) = trace(a_i)$. Hence, by clause 5b of Definition 10, at least one of $u_{m(i-1)}, u_{m(i-1)+1}, \ldots, u_{m(i)}$ is a $q.\mathsf{Green}$ state. Since $m(i-1) \geq g$, this contradicts $\forall \ell \geq g : u_\ell \notin q.\mathsf{Green}$ above. Hence the assumption $\alpha \models \Box\Diamond \top.\mathsf{Green}$ must be false, and so:

$$\alpha \models \Diamond\Box\neg \top.\mathsf{Green}. \qquad (b)$$

From (a) and (b), we conclude (*). By (*) and Lemma 4, we have $\alpha \notin L$. Hence $\alpha' \notin M \Rightarrow \alpha \notin L$ holds. □

We can now establish a correspondence theorem for live executions. Our theorem states that, if a liveness-preserving forward simulation f is established from a concrete automaton to an abstract automaton, then for every live execution α of the concrete automaton, there exists a corresponding (in the sense of $(\alpha, \alpha') \in f$) live execution α' of the abstract automaton. Our proof uses Lemma 5 to establish the existence of an infinite family of finite executions corresponding to prefixes of α. We then construct α' from this infinite family using the "diagonalization" technique of [6] (see, for example, the proof of the Execution Correspondence Theorem in that paper). Finally, we invoke Lemma 6 to show that α' is live (given that α is live).

Theorem 7 (Live Execution Correspondence Theorem) *Let (A, L) and (B, M) be live automata with the same external actions. If $(A, L) \leq^\ell_F (B, M)$ via f, then $((A, L), (B, M)) \in f$.*

Proof. Let $\alpha = s_0 a_1 s_1 a_2 s_2 \ldots$ be an arbitrary live execution of (A, L), and let $(\alpha'_i, m_i)_{0 \leq i}$ be a collection of finite executions of (B, M) as defined in Lemma 5. By definition of $((A, L), (B, M)) \in f$, we must show that there exists a live execution α' of (B, M) such that $(\alpha, \alpha') \in f$.

By definition, α is infinite. Let m be the unique mapping over the natural numbers defined by $m(i) = m_i(i)$, for all $i \geq 0$. Let α' be the limit of α'_i under the prefix ordering. Hence α' is the unique execution of (B, M) defined by $\alpha'|_{m(i)} = \alpha'_i$ with the restriction that for any index j of α', there exists an i such that $\alpha'|_j \leq \alpha'_i$. Furthermore, by Lemma 5, clause 3, α' is infinite.

We now show that m is a live index mapping from α to α' with respect to f. The proof that m is nondecreasing and total and satisfies clauses 1–4 of Definition 10 proceeds in exactly the same way that the proof of the corresponding assertions does in the proof of the Execution Correspondence Theorem (see [6]).

Now assume that m violates clause 5 of Definition 10. Then, there exists a pair $q \in M$ and an $i > 0$ for which clause 5 is invalidated. However, this contradicts the fact that, for all $i > 0$, m_i is a live index mapping from $\alpha|_i$ to α'_i with respect to f (Lemma 5). Hence m satisfies clause 5 of Definition 10. Since m satisfies all clauses of Definition 10, m is a live index mapping from α to α' with respect to f, and so $(\alpha, \alpha') \in f$. Since $\alpha \in L$, $(\alpha, \alpha') \in f$, and α, α' are both infinite, we can apply Lemma 6 to conclude $\alpha' \in M$, i.e., α' is live, which establishes the theorem. □

We now establish our main result: liveness-preserving forward simulation implies the live preorder.

Theorem 8 (Soundness of $\leq^\ell_F$) *If $(A, L) \leq^\ell_F (B, M)$ then $(A, L) \sqsubseteq_\ell (B, M)$.*

Proof. From $(A, L) \leq^\ell_F (B, M)$, we have $(A, L) \leq^\ell_F (B, M)$ via f for some liveness-preserving forward simulation $f = (g, h)$. We establish $traces(L) \subseteq traces(M)$. Let β be an arbitrary trace in $traces(L)$. By definition of trace, $\beta = trace(\alpha)$ for some live execution α of (A, L). By the Live Execution Correspondence Theorem, there exists a live execution α' of (B, M) such that $(\alpha, \alpha') \in f$. Since $(\alpha, \alpha') \in f$, we have $(\alpha, \alpha') \in g$ by Definitions 4 and 10. Hence, by Lemma 3, $trace(\alpha) = trace(\alpha')$. Hence $\beta = trace(\alpha')$, and so $\beta \in traces(M)$, since $\alpha' \in M$. □

4 Example—The Eventually Serializable Data Service

The eventually-serializable data service of [5] is a replicated, distributed data service that trades off immediate consistency for improved efficiency. A shared data object is replicated, and the response to an operation at a particular replica may be out of date, i.e., not reflecting the effects of other operations that have not yet been received by that replica. Thus, operations may be reordered after the response is issued. Replicas communicate amongst each other the operations they receive, so that eventually every operation "stabilizes," i.e., its ordering is fixed with respect to all other operations. Clients may require an operation to be *strict*, i.e., stable at the time of response (and so cannot be reordered after the response is issued). Clients may also specify, in the invocation of an operation x, a set of other operations that should precede x.

The liveness condition used in (the conference version of) [5] is that every request should eventually receive a response, and every operation should stabilize. We express this as the following complemented-pairs liveness condition M for the specification $ESDS\text{-}II \parallel Users$:[7]

- $\langle\!\langle x \in wait, x \notin wait \rangle\!\rangle$ for all x, i.e., every request eventually receives a response.
- $\langle\!\langle x \in wait, x \in stabilized \rangle\!\rangle$ for all x, i.e., every operation eventually stabilizes.

Because the number of submitted operations x in general grows without bound with time, a countably infinite number of pairs is needed to express this liveness condition in the natural manner illustrated above. Note that we use predicates to denote sets of states.

We now refine $\langle\!\langle x \in wait, x \notin wait \rangle\!\rangle$ into a lattice over L, the complemented-pairs liveness condition for the implementation $ESDS\text{-}Alg \parallel Users$. At the implementation level, the natural liveness hypothesis is that each continuously enabled action is eventually executed, and each message in transit eventually arrives. We use this hypothesis to justify the pairs in L. Figure 1 shows the complemented-pairs lattice that refines $\langle\!\langle x \in wait, x \notin wait \rangle\!\rangle$. $c = client(x)$ is the client that invoked operation x. We display the portion of the lattice corresponding to a single replica r. The $\vdots$ indicate where isomorphic copies corresponding to the other replicas occur. Let L consist of all the pairs in Figure 1. It is straightforward to verify that Figure 1 satisfies all the conditions of Definition 8. We justify the complemented pairs in Figure 1 as follows:

1. $\langle\!\langle x \in wait_c, \exists r :< \text{"request"}, x >\in channel_{cr} \rangle\!\rangle$. $\mathsf{send}_{cr}(< \text{"request"}, x >)$ is continuously enabled and eventually happens, for at least one replica r.
2. $\langle\!\langle < \text{"request"}, x >\in channel_{cr}, x \in pending_r \cap rcvd_r \rangle\!\rangle$. Liveness of $channel_{cr}$.
3. $\langle\!\langle x \in pending_r \cap rcvd_r, x \in pending_r \cap done_r[r] \rangle\!\rangle$. If $x.prev \subseteq done_r[r]$, then $\mathsf{do\_it}_r$ is enabled and eventually happens (making $x \in done_r[r]$ true), or is disabled because $x \in done_r[r]$ becomes true (due to a gossip message). Establishing $x.prev \subseteq done_r[r]$ essentially requires a "sublattice" for each $x' \in x.prev$. This sublattice is a "chain" consisting of three pairs:
   (a) $\langle\!\langle x \in pending_r \cap rcvd_r, x' \in pending_{r'} \cap rcvd_{r'} \rangle\!\rangle$ is the bottom element. It is justified since each client includes in $x.prev$ only operations that have already been requested. Thus $x' \in x.prev$ is eventually received by some replica r', at which point $x' \in pending_{r'} \cap rcvd_{r'}$ holds.
   (b) $\langle\!\langle x' \in pending_{r'} \cap rcvd_{r'}, x' \in pending_{r'} \cap done_{r'}[r'] \rangle\!\rangle$ is the middle element. It is justified "inductively," i.e., it can be expanded into a sublattice in exactly the same way as $\langle\!\langle x \in pending_r \cap rcvd_r, x \in pending_r \cap done_r[r] \rangle\!\rangle$. This "nested" expansion is guaranteed to terminate however, since $x.prev$ is finite.
   (c) $\langle\!\langle x' \in done_{r'}[r'], x' \in done_r[r] \rangle\!\rangle$ is the top element. It is justified since r' eventually sends a gossip message to r.

   An alternative to constructing one large lattice that takes care of all $x' \in x.prev$ is to refine the pair $\langle\!\langle x \in pending_r \cap rcvd_r, x \in pending_r \cap done_r[r] \rangle\!\rangle$ at the next lower level of abstraction. That is, construct a refinement in which the automaton is the same, but some complemented pairs are refined.
4. $\langle\!\langle x \in pending_r \cap done_r[r] \wedge x.strict, < \text{"response"}, x, v >\in channel_{rc} \rangle\!\rangle$. This is justified by the following sublattice ($x \in pending_r$, $x.strict$, are implicit conjuncts of all the predicates in the sublattice, and are omitted for clarity):
   (a) $\langle\!\langle x \in \cap done_r[r], x \in \cap_{r'} done_{r'}[r'] \rangle\!\rangle$. Justified since r sends gossip messages to every other replica r'.
   (b) $\langle\!\langle x \in \cap_{r'} done_{r'}[r'], x \in stable_r[r] \rangle\!\rangle$. Justified since each r' sends gossip messages to r.
   (c) $\langle\!\langle x \in stable_r[r], x \in \cap_{r'} stable_{r'}[r'] \rangle\!\rangle$. Justified since r sends gossip messages to every other replica r'.
   (d) $\langle\!\langle x \in \cap_{r'} stable_{r'}[r'], x \in \cap_{r'} stable_r[r'] \rangle\!\rangle$. Justified since each r' sends gossip messages to r.

   $x \in \cap_{r'} stable_r[r']$ holding allows us to conclude that $\mathsf{send}_{rc}(< \text{"response"}, x, v >)$ is continuously enabled and eventually happens.
5. $\langle\!\langle x \in pending_r \cap done_r[r] \wedge \neg x.strict, < \text{"response"}, x, v >\in channel_{rc} \rangle\!\rangle$. $\mathsf{send}_{rc}(< \text{"response"}, x, v >)$ is continuously enabled and eventually happens.
6. $\langle\!\langle < \text{"response"}, x, v >\in channel_{rc}, (x, v) \in rept_c \rangle\!\rangle$. Liveness of $channel_{rc}$.

[7] Throughout this section, our notation is consistent with (the journal version of) [5], to which we refer the reader for details of the eventually serializable data service.

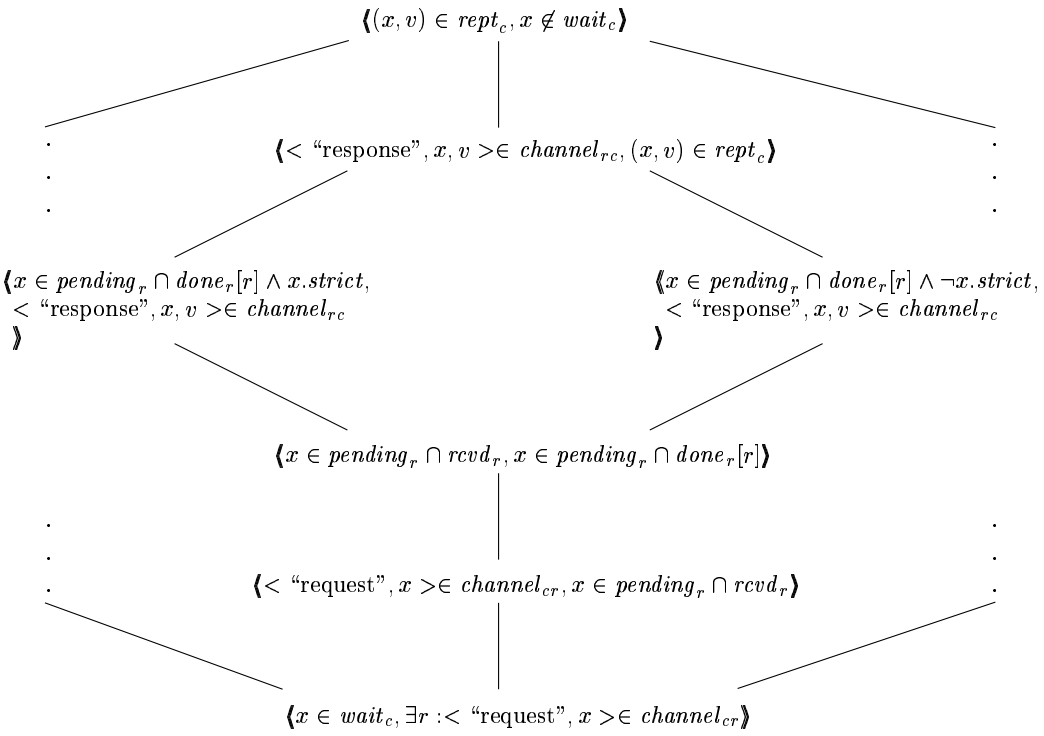

Figure 1: Complemented pairs lattice that refines $\langle\!\langle x \in wait, x \notin wait \rangle\!\rangle$ ($c = client(x)$).

7. $\langle\!\langle (x,v) \in rept_c, x \notin wait_c \rangle\!\rangle$.
   $\mathsf{response}(x,v)$ is continuously enabled and eventually happens.

In [5], a forward simulation F from *ESDS-Alg* $\|$ *Users* to *ESDS-II* $\|$ *Users* is given. We establish that F is also a liveness-preserving forward simulation. By Definition 3, F already satisfies clause 1 of Definition 9. We argue that F also satisfies clauses 2 and 3. Let $SpReq = \langle\!\langle x \in wait, x \notin wait \rangle\!\rangle$, $\bot = \langle\!\langle x \in wait_c, \exists r :<$ "request", $x > \in channel_{cr} \rangle\!\rangle$, and $\top = \langle\!\langle (x,v) \in rept_c, x \notin wait_c \rangle\!\rangle$.

Let s, u range over the states of *ESDS-Alg* $\|$ *Users*, *ESDS-II* $\|$ *Users* respectively. F relates states s and u only if $u.wait = \bigcup_c s.wait_c$ (definition of F in [5]). Hence $x \in u.wait$ iff $x \in s.wait_c$, where $c = client(x)$. Thus u is a $SpReq.\mathsf{Red}$ state iff s is a $\bot.\mathsf{Red}$ state, and u is a $SpReq.\mathsf{Green}$ state iff s is a $\top.\mathsf{Green}$ state.

Let $s \xrightarrow{a}_A s'$ and consider all possibilities for a. If a is one of send (along any channel), $\mathsf{receive}$ (from any channel), or $\mathsf{do\_it}_r$ (for any replica r), then a does not change $wait_c$ (for any client c), and the actions of *ESDS-II* $\|$ *Users* that simulate a do not change *wait*. Hence if $u_1 \xrightarrow{b_1}_B u_2 \xrightarrow{b_2}_B u_3 \cdots \xrightarrow{b_{n-1}}_B u_n$ is the simulating execution fragment of the specification *ESDS-II* $\|$ *Users* (corresponding to $s \xrightarrow{a}_A s'$) then we immediately conclude that (1) all $u_i, i \in 1..n$ have the same value of *wait*, (2) s and s' have the same value of $\bigcup_c wait_c$. Together with $u_1.wait = \bigcup_c s.wait_c$, this allows us to conclude ($\exists i \in 1..n : u_i \in SpReq.\mathsf{Red}$) iff $s \in \bot.\mathsf{Red}$ or $s' \in \bot.\mathsf{Red}$, and $s \in \top.\mathsf{Green}$ or $s' \in \top.\mathsf{Green}$ iff ($\exists i \in 1..n : u_i \in SpReq.\mathsf{Green}$). Thus clause 2 of Definition 9 is satisfied in this case.

If a is $\mathsf{request}(x)$, this is simulated by the same action in *ESDS-II* $\|$ *Users*. $\mathsf{request}(x)$ adds x to $wait_c$ in *ESDS-Alg* $\|$ *Users*, and adds x to *wait* in *ESDS-II* $\|$ *Users*. Hence we easily verify that clause 2 of Definition 9 is satisfied in this case. The argument for $a = \mathsf{response}(x,v)$ is similar. This concludes our argument that clause 2 of Definition 9 holds in all cases.

For clause 3 of Definition 9, we argue that every live execution of *ESDS-Alg* $\|$ *Users* contains an infinite number of $\mathsf{request}(x)$ transitions. This is evident, since there is always some x such that $x.prev \subseteq request.id$. Since this condition is stable, $\mathsf{request}(x)$ is continuously enabled and so is eventually executed. Effectively, we are assuming that, in an infinite live execution, the *Users* make infinitely many requests. This is a reasonable liveness assumption. Since $\mathsf{request}(x)$ is non-silent (see the definition of F in [5]), clause 3 of Definition 9 is satisfied. The preceding argument can be formalized by expressing the liveness assumption that *Users* make infinitely many requests as the complemented pair $\langle\!\langle true, newreq \rangle\!\rangle$, where *newreq* is an auxiliary boolean variable that is set by $\mathsf{request}(x)$, and reset by a new internal action whose precondition is *true* and whose only effect is $newreq := false$. This new action is simulated by the empty transition in *ESDS-II* $\|$ *Users*, and so it does not destroy the simulation F. This complemented pair (which we add to L) can then be refined at the next lower level exactly like any other pair in the liveness condition.

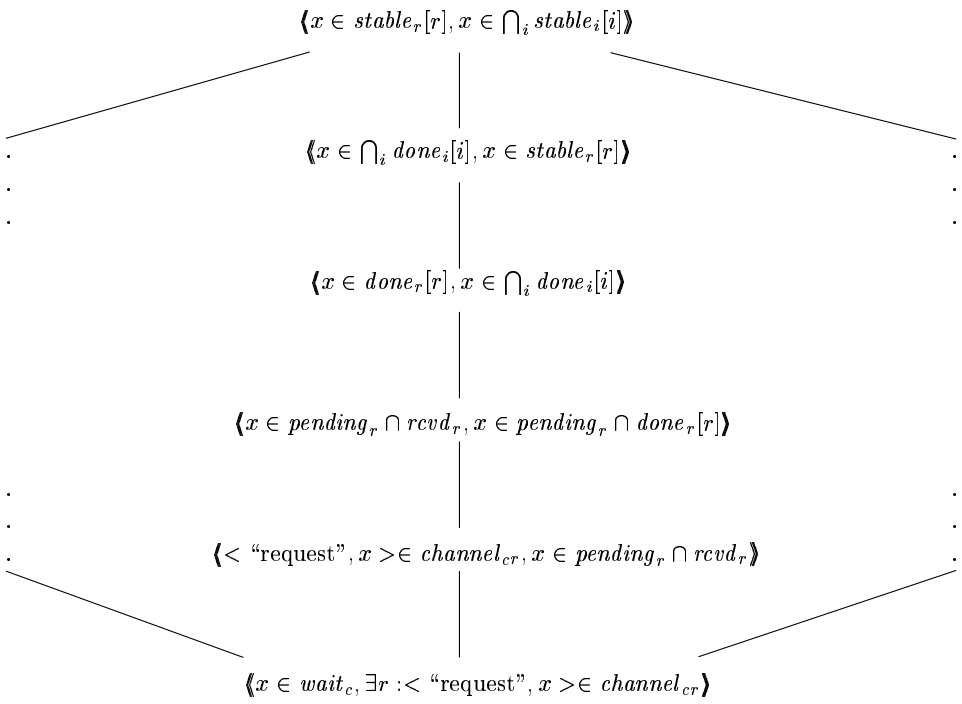

Figure 2: Complemented pairs lattice that refines $\langle\!\langle x \in wait, x \in stabilized \rangle\!\rangle$ ($c = client(x)$).

The pair $\langle\!\langle x \in wait, x \in stabilized \rangle\!\rangle$ can be refined in a similar way to $\langle\!\langle x \in wait, x \notin wait \rangle\!\rangle$. Figure 2 shows the lattice that refines $\langle\!\langle x \in wait, x \in stabilized \rangle\!\rangle$. The bottom three complemented pairs in Figure 2 also occur in Figure 1, and have therefore already been justified. We justify the remaining pairs with the same argument as used for the sublattice of the complemented pair

$$\langle\!\langle x \in pending_r \cap done_r[r] \wedge x.strict, \\ < \text{"response"}, x, v > \in channel_{rc} \rangle\!\rangle$$

(list item 4 above). Now F relates states s and u only if $u.wait = \bigcup_c s.wait_c$ and $u.stabilized = \bigcap_i s.stable_i[i]$ (definition of F in [5]). Hence, it is straightforward to show that F matches the pair $\langle\!\langle x \in wait, x \in stabilized \rangle\!\rangle$ to the lattice of Figure 2 correctly, i.e., according to clause 2 of Definition 9.

We now redefine L so that it contains all the complemented pairs needed to refine both $\langle\!\langle x \in wait, x \notin wait \rangle\!\rangle$ and $\langle\!\langle x \in wait, x \in stabilized \rangle\!\rangle$. Since Definition 9 is satisfied, we have established $(ESDS\text{-}Alg \parallel Users, L) \leq^{\ell}_{F} (ESDS\text{-}II \parallel Users, M)$. Hence, applying Theorem 8, we conclude $(ESDS\text{-}Alg \parallel Users, L) \sqsubseteq_{\ell} (ESDS\text{-}II \parallel Users, M)$, as desired.

5 Related Work

[6] presents an extension of the simulation-based techniques of [8] to deal with liveness properties. In that paper, a liveness property of an automaton A is modeled as a subset L of the executions of A.[8] However, the method presented there requires reasoning over entire executions: in addition to establishing a simulation, we have to show that if an execution α of the implementation A corresponds to an execution α' of the specification B, and $\alpha \in L$, then $\alpha' \in M$[9]. Merely establishing a simulation between A and B is insufficient to show this, since the simulation relation makes no reference to the liveness conditions of A and B.

[10] proposes *proof lattices*, and [9] proposes *proof diagrams*, both for establishing liveness properties of concurrent programs. While these are similar in spirit to our complemented-pairs lattices, they cannot be used to refine liveness properties, since they relate a program to a liveness property expressed in temporal logic. Our complemented-pairs lattices relate a liveness property of a program (i.e., automaton) to a liveness property of a lower level program, and thus form the basis for a multi-stage proof technique that refines high-level liveness properties down to the liveness properties of an implementation in several manageable steps (our use of "sublattices" in Section 4 is an example of this). We feel that this ability to decompose a liveness proof into multiple stages is necessary for dealing successfully with liveness properties of realistic systems, and is one of our main contributions.

[2] proposes the use of complemented-pairs acceptance conditions to define liveness properties. However it restricts the conditions to contain only a finite number of pairs. As our example in Section 4 shows, it is very convenient to be able to specify an infinite number of

[8] L must satisfy the extensibility requirement for finite executions.

[9] See [6], page 89.

pairs—in this case, we were able to use two pairs for each operation x submitted to the data service (one pair to check for response, and the other to check for stabilization). It would be quite difficult to specify the liveness properties of the data service using only a finite number of pairs. If however, the system being considered is finite-state, then we remark that much of the work on temporal logic model checking seems applicable. For example, the algorithm of [4] for model checking under fairness assumptions can handle the complemented-pairs acceptance condition. Finally, while [2] gives rules for compositional and modular reasoning, it does not provide a method for refining liveness properties. As stated above, we believe this is a crucial aspect of a successful methodology for dealing with liveness. It should be clear that Figure 1 provides a very succinct presentation for the refinement of the liveness property expressed by $\langle\!\langle x \in wait, x \notin wait \rangle\!\rangle$, namely that every request eventually receives a response.

[7] presents various extensions of simulation that take fairness into account. Fairness is expressed using either Buchi or Streett (i.e., complemented-pairs) acceptance conditions. However, the fair simulation notions are defined using a game-theoretic semantics, and require a priori that fair executions of the concrete automaton have matching fair executions in the abstract automaton. There is no method of matching the Red and Green states in the concrete and abstract automata to assure fair trace containment. Also, the setting is finite state, and the paper concentrates on algorithms for checking fair simulation.

6 Conclusions and Further Work

We have presented a forward simulation relation that allows us to refine liveness properties. Our results also apply to refinements and history relations [8] since these are special cases of forward simulation. Unlike [6], our method for refining liveness requires reasoning only over individual states/transitions, rather than reasoning over entire executions. We believe that the use of simulation-based refinement together with complemented-pairs lattices for expressing and combining liveness properties provides a very powerful and general framework for refining liveness properties.

While we used the complemented-pairs acceptance condition to express liveness properties, it may also be worthwhile to investigate the other well-known acceptance conditions: the Buchi, Muller, and Rabin (pairs) conditions. We also intend to extend backward simulation [8] to refine liveness, and, for (liveness-preserving) forward simulation, to extend our trace containment results to tree containment [7].

Acknowledgments. I would like to thank Nancy Lynch for challenging me to work on the problem of refining liveness properties, and Victor Luchangco for very helpful discussions about the eventually serializable data service.

References

[1] Abadi, M., and Lamport, L. Composing specifications. *ACM Trans. Program. Lang. Syst. 15*, 1 (1993), 73–132.

[2] Alur, R., and Henzinger, T. Local liveness for compositional modeling of fair reactive systems. In *CAV 95: Computer-aided Verification* (1995), P. Wolper, Ed., Lecture Notes in Computer Science 939, Springer-Verlag, pp. 166–179.

[3] Emerson, E. A. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, J. V. Leeuwen, Ed., vol. B, *Formal Models and Semantics*. The MIT Press/Elsevier, Cambridge, Mass., 1990.

[4] Emerson, E. A., and Lei, C. Modalities for model checking: Branching time logic strikes back. In *12'th Ann. ACM Symp. on Principles of Programming Languages* (Oct. 1985), pp. 84–96.

[5] Fekete, A., Gupta, D., Luchangco, V., Lynch, N., and Shvartsman, A. Eventually-serializable data services. *Theoretical Computer Science, Special Issue on Distributed Algorithms* (1998). Conference version appears in the Proceedings of the 15'th Annual ACM Symposium on Principles of Distributed Computing, 1996.

[6] Gawlick, R., Segala, R., Sogaard-andersen, J., and Lynch, N. Liveness in timed and untimed systems. Tech. Rep. MIT/LCS/TR-587, MIT Laboratory for Computer Science, Boston, Mass., Nov. 1993.

[7] Henzinger, T., Kupferman, O., and Rajamani, S. Fair simulation. In *CONCUR 97: Concurrency Theory* (1997), A. Mazurkiewicz and J. Winkowski, Eds., Lecture Notes in Computer Science 1243, Springer-Verlag, pp. 273–287.

[8] Lynch, N., and Vaandrager, F. Forward and backward simulations — part I: Untimed systems. Tech. Rep. MIT/LCS/TM-486, MIT Laboratory for Computer Science, Boston, Mass., 1993.

[9] Manna, Z., and Pnueli, A. How to cook a temporal proof system for your pet language. In *Proceedings of the ACM Symposium on Principles of Programming Languages* (New York, 1983), ACM.

[10] Owicki, S., and Lamport, L. Proving liveness properties of concurrent programs. *ACM Trans. Program. Lang. Syst. 4*, 3 (July 1982), 455–495.