Transition Invariants

Andreas Podelski Andrey Rybalchenko

Max-Planck-Institut für Informatik
Saarbrücken, Germany*

Abstract

Proof rules for program verification rely on auxiliary assertions. We propose a (sound and relatively complete) proof rule whose auxiliary assertions are transition invariants*. A transition invariant of a program is a binary relation over program states that contains the transitive closure of the transition relation of the program. A relation is* disjunctively well-founded *if it is a finite union of well-founded relations. We characterize the validity of termination or another liveness property by the existence of a disjunctively well-founded transition invariant. The main contribution of our proof rule lies in its potential for automation via abstract interpretation.*

1. Introduction

Temporal verification of concurrent programs is an active research topic; for entry points to the literature see e.g. [6, 9, 11, 13, 14, 15, 23]. In the unifying automata-theoretic framework of [23], a temporal proof is reduced to the proof of fair termination, which again can be done using deductive proof rules, e.g. [11]. The application of these proof rules requires the construction of auxiliary assertions. This construction is generally considered hard to automate, especially when ranking functions and well-founded (lexicographic) orderings are involved.

We propose a proof rule whose auxiliary assertions are *transition invariants*. We introduce the notion of a transition invariant as a binary relation over program states that contains the transitive closure of the transition relation of the program. We formulate an *inductiveness principle* for transition invariants. This principle allows us to identify a given relation as a transition invariant. We also introduce the notion of *disjunctive well-foundedness* as a property of relations. We characterize the validity of a liveness property by the existence of a disjunctively well-founded transition invariant. This is the basis of the soundness and relative completeness of the proof rule.

Applying our proof rule for verifying termination or another liveness property of the program amounts to the following steps: the automata-theoretic construction of a new program (the parallel composition of the original program and a Büchi automaton as in [23]), the inductive proof of the validity of the transition invariant for the new program, and, finally, the test of its disjunctive well-foundedness.

Using transition invariants, we account for the Büchi acceptance condition (and hence, for fairness) in a direct way, namely, by intersecting the transition invariant with a relation over the Büchi accepting states.

If the transition invariant is well-chosen, the test of disjunctive well-foundedness amounts to testing well-foundedness of transition relations of programs of a very particular form: each program is one while loop whose body is a simultaneous update statement. In the case of concurrent programs with linear-arithmetic expressions we obtain while loops for which efficient termination tests are already known [17, 22].

The main contribution of our proof rule lies in its potential for automation. It is a starting point for the development of automated verification methods for temporal properties *beyond safety* of [concurrent] programs over infinite state spaces. As detailed in Section 5, the inductiveness principle allows one to compute the auxiliary assertions of the proof rule. Namely, the transition invariants can be automatically synthesized by computing abstractions of least fixed points of an operator over the domain of relations. Methods to do this correctly and efficiently are studied in the framework of abstract interpretation [4]. Such methods have helped to realize the potential of the inductive proof rules for (state) invariants [14] for the automation of the verification of safety properties [1, 2, 3, 4, 5, 7, 8]. The realization of the analogous potential for transition invariants is not in the scope of this paper; see, however, [16].

*This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS) and by the German Federal Ministry of Education and Research (BMBF) in the framework of the Verisoft project under grant 01 IS C38.

IEEE COMPUTER SOCIETY

Examples We write example programs in the programming language SPL (Simple Programming Language [14]). To simplify the presentation, we ignore idling transitions for the presented concurrent programs. The depicted control-flow graphs treat each straight-line code segment as a single statement. For each of the example programs, we give a (non-inductive) transition invariant, along with an informal argument, in Sections 3 resp. 4; the corresponding formal argument is based on a stronger inductive transition invariant, which we present in Section 5.

LOOPS Usually the termination argument for the program LOOPS on Figure 1 is based on a lexicographic combination of well-founded orderings.

$$\begin{array}{ll} \textbf{in } n & : \textbf{integer where } n > 0 \\ \textbf{local } x, y & : \textbf{integer where } x = n \end{array}$$

$$\left[\begin{array}{l} \ell_0 : \textbf{while } x \geq 0 \textbf{ do} \\ \quad \left[\begin{array}{l} \ell_1 : y := 1 \\ \ell_2 : \textbf{while } y < x \textbf{ do} \\ \qquad \ell_3 : y := 2y \\ \ell_4 : x := x - 1 \end{array}\right] \end{array}\right]$$

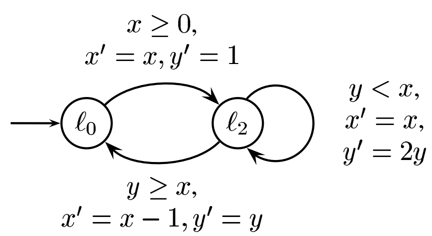

Figure 1. Program LOOPS**.**

We observe that there are only two kinds of loops, those that go through ℓ_0 at least once and decrease the non-negative integer x, and those that go only through ℓ_2 (and not through ℓ_0) and decrease the non-negative value $x - y$. Transition invariants allow one to use this observation for a formal proof of termination.

CHOICE For the termination of the program CHOICE on Figure 2, we observe that the execution of any fixed sequence of statements ℓ_a or ℓ_b decreases either of: x, y or $x+y$. Sections 2 and 3 show that this observation translates to a formal termination argument. Section 5 shows how one can formally justify this observation by an inductive proof.

ANY-DOWN The program ANY-DOWN on Figure 3 consists of two concurrent processes. Each of the processes can be scheduled to be executed by an external scheduler.

$$\begin{array}{ll} \textbf{local } x, y & : \textbf{integer} \end{array}$$

$$\left[\begin{array}{l} \textbf{while } x > 0 \wedge y > 0 \textbf{ do} \\ \quad \left[\begin{array}{l} \ell^a : (x, y) := (x - 1, x) \\ \quad \textbf{or} \\ \ell^b : (x, y) := (y - 2, x + 1) \end{array}\right] \end{array}\right]$$

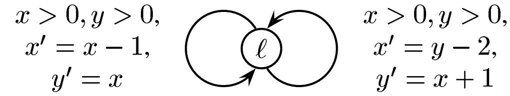

Figure 2. Program CHOICE**.**

$$\begin{array}{ll} \textbf{local } x, y & : \textbf{in teger where} x = 1, y = 0 \end{array}$$

$$P_1 :: \left[\begin{array}{l} \ell_0 : \textbf{while } x = 1 \textbf{ do} \\ \qquad \ell_1 : y := y + 1 \\ \ell_2 : \textbf{while } y > 0 \textbf{ do} \\ \qquad \ell_3 : y := y - 1 \end{array}\right] \parallel P_2 :: \left[\begin{array}{l} m_0 : x := 0 \\ m_1 : \end{array}\right]$$

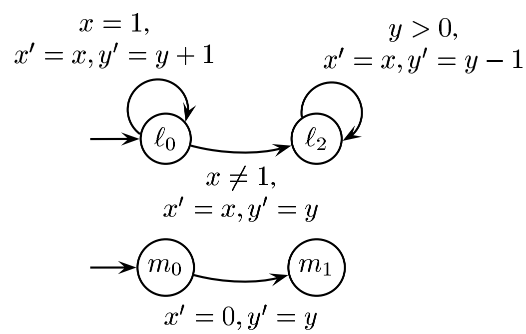

Figure 3. Program ANY-DOWN**.**

The program is not terminating if we consider all possible scheduler behaviors. *E.g.*, in the following infinite computation of ANY-DOWN the process P_2 is never executed (a program state is a tuple containing the location of P_1, the location of P_2, the value of x, and the value of y).

$$\langle \ell_0, m_0, 1, 0 \rangle, \quad \langle \ell_1, m_0, 1, 0 \rangle, \quad \langle \ell_0, m_0, 1, 1 \rangle, \quad \ldots$$

This computation is not *fair* because the process P_2 is never executed although it is continuously enabled. If we assume that the scheduling for each process is fair (see [11, 14] for a detailed treatment of fairness assumptions), then the program ANY-DOWN is terminating.

In Section 4 we show how we incorporate the fairness assumption into a fairness proof.

CONC-WHILES A termination proof for the program CONC-WHILES on Figure 4 requires a more complicated fairness assumption (each of the processes must be scheduled infinitely often, hence it is not possible that a process waits forever).

$$
\begin{array}{ll}
\textbf{local } x, y & : \textbf{integer where } x > 0, y > 0 \\
P_1 :: & \left[\begin{array}{l} \ell_0 : \textbf{while } x > 0 \textbf{ do} \\ \quad \ell_1 : y := x - 1 \\ \ell_2 : y := 0 \end{array}\right] \\
|| \quad P_2 :: & \left[\begin{array}{l} m_0 : \textbf{while } y > 0 \textbf{ do} \\ \quad m_1 : x := y - 1 \\ m_2 : x := 0 \end{array}\right]
\end{array}
$$

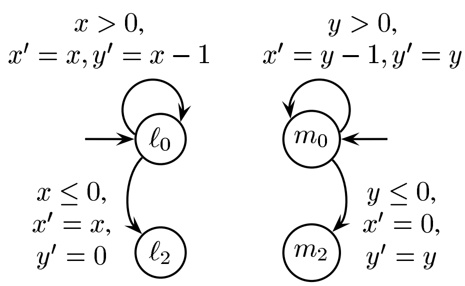

Figure 4. Program CONC-WHILES**.**

Our formal proof in Section 4 will follow the intuition that each infinite fair computation decreases the value of x as well as the value of y infinitely often.

2. Transition Invariants

This section deals with properties of general binary relations. For concreteness we formulate the properties for the transition relation of a program and its restriction to the set of accessible states. We next formalize programs.

Program P A *program* $P = \langle W, I, R \rangle$ consists of:

- W: a set of *states*,
- I: a set of *starting states*, such that $I \subseteq W$,
- R: a *transition relation*, such that $R \subseteq W \times W$.

A *computation* is a maximal sequence of states $s_1, s_2, \ldots$ such that:

- s_1 is a starting state, *i.e.*, $s_1 \in I$,
- $(s_i, s_{i+1}) \in R$ for all $i \geq 1$ (and $i \leq n - 1$, if the sequence is of the finite length n).

The set Acc of *accessible states* consists of all states that appear in some computation.

A finite segment $s_i, s_{i+1}, \ldots, s_j$ of a computation where $i < j$ is called a *computation segment*.

Definition 1 (Transition Invariant)
A transition invariant T *is a superset of the transitive closure of the transition relation R restricted to the accessible states Acc. Formally,*

$$R^+ \cap (Acc \times Acc) \subseteq T.$$

Thus, a transition invariant of the program is a relation T on the program states such that for every computation segment $s_i, s_{i+1}, \ldots, s_j$ the pair of states (s_i, s_j) is an element of T.

Note that the Cartesian product of the set of states with itself, *i.e.* the relation $W \times W$, is a transition invariant of the program. A superset of the transitive closure of the transition relation of the program is a transition invariant of the program; the converse does not hold.

A *state invariant* is a superset of Acc. Given the transition invariant T and the set of starting states I, the set

$$I \cup \{s' \mid s \in I \text{ and } (s, s') \in T\}$$

is a state invariant. Conversely, a transition invariant can be strengthened by restricting it to a given state invariant.

A program is *terminating* if it does not have infinite computations. This is equivalent to the fact that its transition relation restricted to the accessible states, *i.e.*, $R \cap (Acc \times Acc)$, is well-founded. We investigate the well-foundedness of a transition relation through a weaker property of its transition invariant, introduced next.

Definition 2 (Disjunctive Well-foundedness)
A relation T is disjunctively well-founded *if it is a finite union $T = T_1 \cup \cdots \cup T_n$ of well-founded relations.*

Every well-founded relation is disjunctively well-founded. The converse does not hold in the general case. *E.g.*, the relation ACK-REQ defined by

$$\{(ack, req)\} \cup \{(req, ack)\}$$

is disjunctively well-founded but not well-founded.

Given a disjunctively well-founded relation T, the implication:

$$R \text{ is well-founded if } R \subseteq T$$

does not hold (for a counterexample, take R and T to be the relation ACK-REQ). However, the implication:

$$R \text{ is well-founded if } R^+ \subseteq T$$

does hold, as we show below.

IEEE COMPUTER SOCIETY

Theorem 1 (Termination) *The program P is terminating if and only if there exists a disjunctively well-founded transition invariant for P.*

Proof. **if**-direction: Assume, for a proof by contraposition, that

$$T \stackrel{\text{def}}{=} T_1 \cup \cdots \cup T_n$$

is a disjunctively well-founded transition invariant for the program P, and that P is not terminating. We show that at least one sub-relation T_i of the transition invariant is not well-founded.

By the assumption that P is not terminating, there exists an infinite computation $\sigma \stackrel{\text{def}}{=} s_1, s_2, \ldots$.

We define a function f that maps an ordered pair of indices of the states in the computation σ to one of the sub-relations in the transition invariant T as follows.

$$\text{For } k < l,\ f(k,l) \stackrel{\text{def}}{=} T_i \text{ such that } (s_k, s_l) \in T_i$$

The function f exists because T is a transition invariant, and thus we can arbitrarily choose one relation from the (finite) set $\{T_i \mid (s_k, s_l) \in T_i\}$ as the image of the pair (k, l). Note that the range of the function f is finite.

For the fixed computation σ, the function f induces an equivalence relation $\sim$ on pairs of positive integers (in this proof we always consider pairs whose first element is smaller than the second one).

$$(k,l) \sim (k',l') \stackrel{\text{def}}{=} f(k,l) = f(k',l')$$

The equivalence relation $\sim$ has finite index, since the range of f is finite.

By Ramsey's theorem [18], there exists an infinite sequence of positive integers $K \stackrel{\text{def}}{=} k_1, k_2, \ldots$ such that all pairs of elements in K belong to the same equivalence class, say $[(m,n)]_\sim$ with $m, n \in K$. That is, for all $k, l \in K$ such that $k < l$ we have $(k,l) \sim (m,n)$. We fix m and n.

Let T_{mn} denote the relation $f(m,n)$. Since $(k_i, k_{i+1}) \sim (m,n)$ for all $i \geq 1$, the function f maps every pair (k_i, k_{i+1}) to T_{mn} for all $i \geq 1$. Hence, the infinite sequence $s_{k_1}, s_{k_2}, \ldots$ is induced by T_{mn}, *i.e.*,

$$(s_{k_i}, s_{k_{i+1}}) \in T_{mn},\ \text{for all } i \geq 1.$$

Hence, the sub-relation T_{mn} is not well-founded.

only if-direction: Assume that the program P is terminating. We define the relation T as the restriction of the transition relation to accessible states.

$$T \stackrel{\text{def}}{=} R^+ \cap (\mathit{Acc} \times \mathit{Acc})$$

Clearly, T is a transition invariant. Assume that $\sigma \stackrel{\text{def}}{=} s^1, s^2, \ldots$ is an infinite sequence of states such that $(s^i, s^{i+1}) \in T$ for all $i \geq 1$. Since the state s^1 is accessible, and for all $i \geq 1$ there is a non-empty computation segment leading from s^i to s^{i+1} (*i.e.* $(s^i, s^{i+1}) \in R^+$), there exists an infinite computation $s_1, \ldots, s^1, \ldots, s^2, \ldots$. This fact is a contradiction to our assumption that P is terminating. Hence, T is (disjunctively) well-founded. ☐

The relation ACK-REQ shows that we cannot drop the requirement that not just the transition relation of a program, but also its transitive closure must be contained in the disjunctively well-founded relation T.

The next example shows that we cannot drop the finiteness requirement in the definition of disjunctive well-foundedness. The following transition relation

$$R \stackrel{\text{def}}{=} \{(i, i+1) \mid i \geq 1\}$$

has a transition invariant $T \stackrel{\text{def}}{=} T_1 \cup T_2 \cup \ldots$ that is the union of well-founded relations T_i, where

$$T_i \stackrel{\text{def}}{=} \{(i, i+j) \mid j \geq 1\}, \quad \text{for all } i \geq 1.$$

However, the relation R is not well-founded.

3. Termination

Theorem 1 gives a (complete) characterization of program termination by disjunctively well-founded transition invariants.

We next present disjunctively well-founded transition invariants for the first resp. second program shown in the introduction. Here, we only give informal arguments; in Section 5 we will show how one can formally prove that the relations are indeed transition invariants, and give the formal argument in the form of (stronger) inductive transition invariants.

LOOPS The union of the relations T_1, T_2 and T_{ij} for $i \neq j \in \{0, \ldots, 4\}$ denoted by the following assertions over the unprimed and primed program variables is a transition invariant for the program LOOPS; here pc is a variable ranging over the set of location labels $\{\ell_0, \ldots, \ell_4\}$.

T_1	$x \geq 0 \wedge x' < x$	
T_2	$x - y > 0 \wedge x' - y' < x - y$	
T_{ij}	$pc = \ell_i \wedge pc' = \ell_j$	where $i \neq j \in \{0, \ldots, 4\}$

The intuitive argument that the union of the relations above indeed identifies a transition invariant may go as follows. We can distinguish three kinds of computation segments that lead a state s to a state s'. All pairs of states (s, s') in R^+ such that s goes to s' via the location ℓ_0 (and in particular the loops at ℓ_0) are contained in the relation T_1. All

IEEE COMPUTER SOCIETY

pairs of states (s, s') in R^+ such that s goes to s' via the location ℓ_2 and not ℓ_0 (and in particular the loops at ℓ_2) are contained in the relation T_2. Every pair of states in R^+ that has different location labels is contained in one of T_{ij}'s.

Obviously, the relations T_1 and T_2 as well as the relations T_{ij}'s are well-founded.

CHOICE The union the relations below is a transition invariant for the program CHOICE.

$$\begin{array}{ll} T_1 & x > 0 \wedge x' < x \\ T_2 & x + y > 0 \wedge x' + y' < x + y \\ T_3 & y > 0 \wedge y' < y \end{array}$$

Again, the relations T_1, T_2, and T_3 are obviously well-founded.

4. Liveness

We follow the automata-theoretic framework for the temporal verification of concurrent programs [23]. This framework allows us to assume that the temporal correctness specification, viz. a liveness property Ψ and a fairness assumption Φ, are given by a (possibly infinite-state) automaton $A_{\Phi,\Psi}$. The intuition is that the automaton $A_{\Phi,\Psi}$ accepts exactly the infinite Φ-fair sequences of program states that do not satisfy the property Ψ. We assume that the automaton $A_{\Phi,\Psi}$ is equipped with the Büchi acceptance condition.

Automaton $A_{\Phi,\Psi}$ We consider an alphabet consisting of the program states W. The *automaton* $A_{\Phi,\Psi} = \langle Q, Q^0, \Delta, F \rangle$ consists of:

- Q: a (possibly infinite) set of states,
- Q^0: the set of *starting* states, such that $Q^0 \subseteq Q$,
- Δ: the *transition relation*. It is a set of triples $(q, s, q') \in Q \times W \times Q$.
- F: the set of accepting states, such that $F \subseteq Q$.

A *run* of the automaton $A_{\Phi,\Psi}$ *on* the word $s_1, s_2, \ldots$ is a sequence of the automaton states $q_1, q_2, \ldots$ such that $q_1 \in Q^0$ and $(q_i, s_i, q_{i+1}) \in \Delta$ for all $i \geq 1$. The automaton *accepts* a word w if it has a run $q_1, q_2, \ldots$ on w such that for infinitely many i's we have $q_i \in F$.

Program $P_{\Phi,\Psi}$ The program P satisfies the liveness property Ψ under the fairness assumption Φ if there exists no infinite computation of P that satisfies the fairness condition Φ and falsifies the property Ψ, *i.e.*, all computations of the program P are rejected by the automaton $A_{\Phi,\Psi}$ (computations are infinite words over the alphabet W; finite computations are added an idling transition for the last state). We export the program computations to the automaton by the parallel composition $P_{\Phi,\Psi}$ of the program and the automaton, which we introduce next.

The program $P_{\Phi,\Psi}$, which in fact is equipped with the Büchi acceptance condition, is obtained by the synchronous parallel composition of P and $A_{\Phi,\Psi}$. The set of states of $P_{\Phi,\Psi}$ is the Cartesian product

$$W_Q \stackrel{\text{def}}{=} W \times Q.$$

The set of starting states is $I \times Q^0$. The transition relation of $P_{\Phi,\Psi}$ consists of pairs $((s, q), (s', q'))$ such that $(s, s') \in R$ and $(q, s, q') \in \Delta$. The set of accepting states is the product

$$W_F \stackrel{\text{def}}{=} W \times F.$$

A computation $(s_1, q_1), (s_2, q_2), \ldots$ of $P_{\Phi,\Psi}$ is *fair* if for infinitely many i's we have $(s_i, q_i) \in W_F$. The program P is correct with respect to the property Ψ under the fairness condition Φ if and only if all (infinite) computations of $P_{\Phi,\Psi}$ are not fair (see Theorem 4.1 in [23]). The terminology '$P_{\Phi,\Psi}$ *is fair terminating*' is short for 'all (infinite) computations of $P_{\Phi,\Psi}$ are not fair'.

The following theorem characterizes the validity of the temporal property Ψ (under the fairness assumption Φ) through the existence of a disjunctively well-founded transition invariant for the program $P_{\Phi,\Psi}$ (with the set W_F of Büchi accepting states).

Theorem 2 (Liveness) *The program P satisfies the liveness property Ψ under the fairness assumption Φ if and only if there exists a transition invariant T for $P_{\Phi,\Psi}$ such that $T \cap (W_F \times W_F)$ is disjunctively well-founded.*

Proof. **if**-direction (sketch): Assume, for a proof by contraposition, that the finite union

$$T \stackrel{\text{def}}{=} T_1 \cup \cdots \cup T_n,$$

such that $T_i \cap (W_F \times W_F)$ is well-founded for all $i \in \{1, \ldots, n\}$, is a transition invariant for the program $P_{\Phi,\Psi}$. Furthermore, we assume that $P_{\Phi,\Psi}$ has an (infinite) fair computation (*i.e.*, is not fair terminating). We prove that at least one relation $T_i \cap (W_F \times W_F)$ is not well-founded.

By the assumption that $P_{\Phi,\Psi}$ is not fair terminating, there exists an infinite fair computation $\sigma \stackrel{\text{def}}{=} s_1, s_2, \ldots$. Let $\pi \stackrel{\text{def}}{=} s^1, s^2, \ldots$ be an infinite subsequence of σ such that $s^i \in W_F$ for all $i \geq 1$.

Now we can follow the lines of the **if**-part of the proof of Theorem 1. We show that there exists an infinite subsequence of π and an index $i \in \{1, \ldots, n\}$ such that each

IEEE COMPUTER SOCIETY

pair of consecutive states in the subsequence is an element of the very same relation $T_i \cap (W_F \times W_F)$. Thus we obtain a contradiction to the assumption that $T_i \cap (W_F \times W_F)$ is well-founded for all $i \in \{1, \ldots, n\}$.

only if-direction: Assume that the program $P_{\Phi,\Psi}$ is fair terminating (*i.e.*, has no (infinite) fair computation). Let Acc denote the set of accessible states of $P_{\Phi,\Psi}$. We define the following relations on the accessible states of $P_{\Phi,\Psi}$.

$$T_1 \overset{\text{def}}{=} R^+ \cap (Acc \cap W_F \times Acc)$$
$$T_2 \overset{\text{def}}{=} R^+ \cap (Acc \setminus W_F \times Acc)$$

Clearly, the relation

$$T \overset{\text{def}}{=} T_1 \cup T_2$$

is a transition invariant. Assume that $\sigma \overset{\text{def}}{=} s^1, s^2, \ldots$ is an infinite sequence of states such that $(s^i, s^{i+1}) \in T_1$ for all $i \geq 1$. Since the state s^1 is accessible, and for all $i \geq 1$ there is a non-empty computation segment leading from s^i to s^{i+1} (*i.e.* $(s^i, s^{i+1}) \in R^+$) there exists an infinite fair computation $s_1, \ldots, s^1, \ldots, s^2, \ldots$. This fact is a contradiction to our assumption that P is fair terminating. Hence, T_1 is well-founded. Clearly, the intersection $T_2 \cap (W_F \times W_F)$ is empty. We conclude that the **only-if** direction holds. □

Examples We give a transition invariant for each of the programs $P_{\Phi,\Psi}$ obtained by the parallel composition of the program ANY-DOWN resp. CONC-WHILES with the Büchi automaton $A_{\Phi,\Psi}$ that encodes the appropriate fairness assumption Φ (the liveness property Ψ is termination; the automaton $A_{\Phi,\Psi}$ accepts exactly the infinite Φ-fair computations). We do not explicitely present $A_{\Phi,\Psi}$ and $P_{\Phi,\Psi}$ since they can be easily derived.

ANY-DOWN Here, the Büchi automaton $A_{\Phi,\Psi}$ encodes the fairness assumption "eventually the process P_2 leaves the location m_0" which is expressed by the temporal logic formula $\Phi \overset{\text{def}}{=} \mathsf{F}\,(pc_2 \neq m_0)$. The union of the relations below forms a transition invariant for $P_{\Phi,\Psi}$. The two variables pc_1 and pc_2 range over the location labels of the first resp. second process. The variable pc_Q ranges over the states q_0 and q_F of the Büchi automaton $A_{\Phi,\Psi}$ (where the state q_F is accepting).

T_1	$pc_Q = q_F \wedge y > 0 \wedge y' < y$
T_2	$pc_Q \neq q_F$
T_3	$pc_Q = q_0 \wedge pc'_Q = q_F$
T_4	$pc_2 = m_0 \wedge pc'_2 = m_1$
T_{ij}	$pc_1 = \ell_i \wedge pc'_1 = \ell_j$ where $i \neq j \in \{0, \ldots, 3\}$

The relation T_1 contains the pairs of states $((s,q),(s',q'))$ from the transitive closure R^+ of the program $P_{\Phi,\Psi}$ that are the initial and the final states of the loops starting in the Büchi accepting state. These loops are induced by the execution of the **while**-statement at the location ℓ_2. For the **while**-statement at the location ℓ_0 the initial-final state pairs are elements of T_2. The relations T_3, T_4, and T_{ij} where $i \neq j \in \{0, \ldots, 3\}$ contain pairs of states that have different location labels wrt. either the Büchi automaton or one of the processes.

The relations T_1, T_3, T_4, and T_{ij}'s are well-founded. According to the formal argument of this section, the relation T_2 is not considered; the restriction of T_2 to the Büchi accepting states is empty.

CONC-WHILES We encode the fairness assumption that no process can wait forever (except in the final location) by the temporal formula below.

$$\mathsf{GF}\,(pc_1 \neq 0) \wedge \mathsf{GF}\,(pc_1 \neq 1) \wedge$$
$$\mathsf{GF}\,(pc_2 \neq 0) \wedge \mathsf{GF}\,(pc_2 \neq 1)$$

The corresponding Büchi automaton has the four states $\{q_0, q_1, q_2, q_F\}$, where the state q_F is accepting.

The union of the following relations is a transition invariant for $P_{\Phi,\Psi}$.

T_1	$pc_Q = q_F \wedge x > 0 \wedge x' < x$
T_2	$pc_Q = q_F \wedge y > 0 \wedge y' < y$
T_3	$pc_Q \neq q_F$
T^4_{ij}	$pc_Q = q_i \wedge pc'_Q = q_j$ where $i \neq j \in \{0, 1, 2\}$
T^5_{ij}	$pc_1 = \ell_i \wedge pc'_1 = \ell_j$ where $i \neq j \in \{0, 1, 2\}$
T^6_{ij}	$pc_2 = m_i \wedge pc'_2 = m_j$ where $i \neq j \in \{0, 1, 2\}$

The relations T_1 and T_2 capture loops that start in the Büchi accepting state and contain execution steps of both processes P_1 and P_2. The loops that contain the executions of only P_1 or only P_2 are captured by the relation T_3. The relations T^4_{ij}, T^5_{ij}, and T^6_{ij} with $i \neq j \in \{0, 1, 2\}$ capture computation segments that are not loops wrt. the location labels of either the Büchi automaton or one of the processes.

The well-foundedness of the relations T_1, T_2, T^4_{ij}, T^5_{ij}, and T^6_{ij} for $i \neq j \in \{0, 1, 2\}$ is sufficient for proving the fair termination property; the restriction of T_3 to the Büchi accepting state is empty.

5. Inductiveness

In this section, we formulate a proof rule for verifying liveness properties of concurrent programs. The proof rule is based on inductive transition invariants.

Definition 3 (Inductive Relation) *Given a program with the transition relation R, a binary relation T on program states is* inductive *if it contains the transition relation R and it is closed under the relational composition with R. Formally,*

$$R \cup T \circ R \subseteq T.$$

As usual, the *composition operator* $\circ$ denotes the relational composition, *i.e.*, for $P, Q \subseteq W \times W$ we have

$$P \circ Q \stackrel{\text{def}}{=} \{(s, s') \mid (s, s'') \in P \text{ and } (s'', s') \in Q\}.$$

Replacing the inductiveness criterion above by $R \cup R \circ T \subseteq T$ yields an equivalent criterion. Replacing it by $R \cap (Acc \times Acc) \cup T \circ R \cap (Acc \times Acc) \subseteq T$ yields a slightly weaker criterion. This may be useful in some situations.

Remark 1 *An inductive relation for the program P is a transition invariant for P.*

Inductive relations are called *inductive transition invariants*.

Note that a transition invariant T, even if it is inductive, is in general not closed under the composition with itself, *i.e.*, in general

$$T \circ T \not\subseteq T.$$

In other words, a transition invariant, even if it is inductive, need not be transitive.

We note in passing a simple but perhaps curious consequence of Theorem 1 and Remark 1.

Corollary 1 (Compositionality) *A finite union of well-founded relations is well-founded if it is closed under relational composition with itself.*

Proof. Let the relation T be the finite union of the well-founded relations that is closed under the composition with itself, *i.e.* $T \circ T \subseteq T$.

By Remark 1, T is an inductive transition invariant for itself. Since T is disjunctively well-founded, we have that T is well-founded by Theorem 1. ☐

Proof Rule Theorem 2 and Remark 1 give rise to a proof rule for the verification of liveness properties; see Figure 5. Again, the formulation uses the automata-based framework for verification of concurrent programs [23]. We obtain a proof rule for termination by taking R as the transition relation of the program P, a relation $T \subseteq W \times W$ and replacing $T \cap (W_F \times W_F)$ by T in the premise P3.

In our examples we split the reasoning on disjunctive well-foundedness and inductiveness. This can be seen as using an alternative, equivalent formulation of the proof rule: one takes two relations T and T' such that T satisfies the premise P3 and T' is a subset of T that satisfies the premises P1 and P2 (*i.e.*, one identifies T as as transition invariant by strengthening T with the inductive relation T'). The two formulations are equivalent since the disjunctive well-foundedness of a relation is inherited by its subset.

Program P,
liveness property Ψ,
fairness assumption Φ,
Büchi automaton $A_{\Phi,\Psi}$,
parallel composition of P and $A_{\Phi,\Psi}$ is program $P_{\Phi,\Psi}$ with:
  transition relation R,
  states W_Q,
  accepting states W_F,
relation $T \subseteq W_Q \times W_Q$

P1:	R	$\subseteq$	T
P2:	$T \circ R$	$\subseteq$	T
P3:	$T \cap (W_F \times W_F)$	disjunctively well-founded	

P satisfies Ψ under Φ

Figure 5. Rule LIVENESS.

As already mentioned, a transition invariant can be strengthened by restricting it to a given state invariant S. *I.e.*, if T is a transition invariant and S is a state invariant, then

$$T' \stackrel{\text{def}}{=} T \cap (S \times S)$$

is a (stronger) transition invariant.

Validation of the Premises of the Proof Rule We assume that the transition relation R is given as a union of relations, say $R = R_1 \cup \cdots \cup R_m$. This is usually the case for concurrent programs, where each program statement denotes a transition R_i (an assertion over unprimed and primed program variables, as seen in the examples).

If T is given as the union $T = T_1 \cup \cdots \cup T_n$, then the composition $T \circ R$ is the union of the relations $T_i \circ R_j$ for $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$. Each relation $T_i \circ R_j$ is again represented by an assertion over unprimed and primed program variables.

Thus, the premises P1 and P2 can be established by entailment checks between assertions.

The premise P3 can be established using traditional methods for proving termination. In the extreme case, when $n = 1$, *i.e.*, the transition invariant or its partitioning are ill-chosen, the reduction to the disjunctive well-foundedness has not brought any simplification and is as hard as before the reduction. In the other cases, with a well-chosen transition invariant and partitioning, the premise P3 can be established by a number of pairwise independent 'simple' well-foundedness tests.

Note that all relations T_i in the transition invariants of the programs presented in this paper correspond to 'simple while' programs that consist of a single while loop with only update statements in its body.

More generally, the relation $g(\vec{X}) \wedge e(\vec{X}', \vec{X})$ is well-founded if and only if the while loop

$$\left[\begin{array}{c}\textbf{while}\ g(\vec{X})\ \textbf{do}\\ e(\vec{X}', \vec{X})\end{array}\right]$$

is terminating.

In the case of concurrent programs with linear-arithmetic expressions, the well-foundedness test in the premise P3 amounts to the termination test of simple while programs, for which an efficient test exists; see [17, 22].

In the special case of finite-state systems (a case that we do not target), each 'small' termination problem is to check whether a transition is a self-loop.

Inductive Transition Invariants for Examples Each of the relations T shown in Section 3 and 4 is not inductive (*i.e.*, the composition of one of the relations T_i and one of the program transitions R_j is not a subset of T). We formally identify each T as a transition invariant by presenting an inductive one that strengthens T (*i.e.*, is a subset of T). We thus complete the termination resp. liveness proof according to the proof rule.

LOOPS The union of the following relations is an inductive transition invariant for the program LOOPS (in the version according to the depicted control-flow graph).

$$\begin{array}{l}
pc{=}\ell_0 \wedge x \geq 0 \wedge x' \leq x \wedge pc'{=}\ell_2\\
pc{=}\ell_2 \wedge x' < x \wedge pc'{=}\ell_0\\
pc{=}\ell_2 \wedge x - y > 0 \wedge x' \leq x \wedge y' > y \wedge pc'{=}\ell_2\\
pc{=}\ell_0 \wedge x \geq 0 \wedge x' < x \wedge pc'{=}\ell_0\\
pc{=}\ell_2 \wedge x \geq 0 \wedge x' < x \wedge pc'{=}\ell_2
\end{array}$$

The inductiveness can be easily verified. *E.g.*, the composition of the relation below (which is the transition for the straight-line code from the location ℓ_2 to ℓ_0; it is obtained by composing the transition between the locations ℓ_2 and ℓ_4 and the transition from ℓ_4 to ℓ_0),

$$pc{=}\ell_2 \wedge y \geq x \wedge x' = x - 1 \wedge y' = y \wedge pc'{=}\ell_0$$

with the first of the five relations above yields the relation below, a relation that entails the fourth.

$$pc{=}\ell_0 \wedge x \geq 0 \wedge x' \leq x - 1 \wedge pc'{=}\ell_0$$

CHOICE The union of the relations below is an inductive transition invariant for the program CHOICE.

$$\begin{array}{l}
x > 0 \wedge y > 0 \wedge x' < x \wedge y' \leq x\\
x > 0 \wedge y > 0 \wedge x' < y - 1 \wedge y' \leq x + 1\\
x > 0 \wedge y > 0 \wedge x' < y - 1 \wedge y' < y\\
x > 0 \wedge y > 0 \wedge x' < x \wedge y' < y
\end{array}$$

ANY-DOWN We next present (the interesting part of) an inductive transition invariant for the parallel composition $P_{\Phi,\Psi}$ of the program ANY-DOWN with the Büchi automaton $A_{\Phi,\Psi}$ that accepts exactly the infinite sequences of program states that are fair, *i.e.*, where the second process does not wait forever. We do not present the relations where the values of one of the program counters are different before and after the transition; we only present the relations that are loops in the control flow graph for the program $P_{\Phi,\Psi}$. Each of the relations satisfy the conjunction of $pc_1 = pc_1'$ and $pc_2 {=} pc_2'$ and $pc_Q {=} pc_Q'$; we omit that conjunction in each of the assertions below.

$$\begin{array}{l}
pc_Q{=}q_F \wedge pc_1{=}\ell_2 \wedge pc_2{=}m_1 \wedge y > 0 \wedge x' = x \wedge y' < y\\
pc_Q{=}q_F \wedge pc_1{=}\ell_3 \wedge pc_2{=}m_1 \wedge y > 0 \wedge x' = x \wedge y' < y\\
pc_Q{\neq}q_F \wedge pc_1{=}\ell_0 \wedge pc_2{=}m_0 \wedge x' = x\\
pc_Q{\neq}q_F \wedge pc_1{=}\ell_1 \wedge pc_2{=}m_0 \wedge x' = x\\
pc_Q{\neq}q_F \wedge pc_1{=}\ell_2 \wedge pc_2{=}m_1 \wedge y > 0 \wedge x' = x \wedge y' < y\\
pc_Q{\neq}q_F \wedge pc_1{=}\ell_3 \wedge pc_2{=}m_1 \wedge y > 0 \wedge x' = x \wedge y' < y
\end{array}$$

CONC-WHILES The transition invariant for $P_{\Phi,\Psi}$ contains the following relations. We show only those that are loops wrt. the location labels; again, we omitted the conjunction $pc_1{=}pc_1' \wedge pc_2{=}pc_2' \wedge pc_Q{=}pc_Q'$ in each assertion below.

$$\begin{array}{l}
pc_Q{=}q_F \wedge x > 0 \wedge x' < x \wedge y' < x\\
pc_Q{=}q_F \wedge y > 0 \wedge x' < y \wedge y' < y\\
pc_Q \neq q_F \wedge x > 0 \wedge x' \leq x \wedge y' < x\\
pc_Q \neq q_F \wedge y > 0 \wedge x' \leq y \wedge y' \leq y
\end{array}$$

Soundness and Completeness The separation of the temporal reasoning from the reasoning about the auxiliary assertions in the 'relative' completeness statement below is common practice; see e.g. [13, 14].

Theorem 3 (Proof Rule LIVENESS) *The rule* LIVENESS *is sound, and complete relative to the first-order assertional validity and the well-foundedness validity of the relations that constitute the transition invariant.*

Proof. The soundness of the rule follows directly from Remark 1 and Theorem 2.

For proving the relative completeness, we observe that the transition invariant constructed in the proof of Theorem 2 is in fact inductive. In order to establish the completeness relative to assertional provability, we need to show that this inductive transition invariant is expressible by a first-order assertion.

We need to construct the assertion T over unprimed and primed program variables that denotes a transition invariant satisfying the premises of the rule LIVENESS. We omit the construction, which follows the lines of the method for constructing the assertion Acc that denotes the set of all accessible states [14]. $\square$

Automated Liveness Proofs Given a program with the transition relation R, we are interested in the subclass of its inductive transition invariants.

We define the operator F over relations by

$$F(T) \stackrel{\text{def}}{=} T \circ R.$$

We write $F^{\#} \supseteq F$ and say that $F^{\#}$ is an *approximation* of F, if $F^{\#}(S) \supseteq F(S)$ holds for all relations S.

The inductive transition invariants are (exactly the) least fixed points above R of operators $F^{\#}$ such that $F^{\#} \supseteq F$.

There are many techniques based e.g. on widening or predicate abstraction that have been applied with great success to the automated construction of least fixed points of approximation of the *post* operator [1, 2, 3, 4, 5, 7, 8]. Now we can start to carry over the abstract interpretation techniques in order to construct least fixed points of approximations of the operator F. Thus, relations T that satisfy the premises P1 and P2 can be constructed automatically.

As already mentioned, the validation of the premise P3 can be automated for interesting classes of concurrent programs over linear-arithmetic expressions (see [17] and [22]). Automated checks for other classes of programs are an open topic of research.

6. Related Work

There is a large body of work on proof rules for liveness properties of concurrent programs, see [6, 11, 13, 15]. They all rely on auxiliary well-founded (lexicographic) orderings for the transition relation, and not on independent orderings for sub-relations, as in our approach.

The automata-theoretic approach for verification of concurrent programs [23] reduces the verification problem to proving termination. It leaves open how to prove termination. We indicate one possible way.

A rank predicate [24] (a notion directly related to progress measures [9]) proves fair termination of a program if the rank does not increase in every computation step and decreases in the accepting states. In a disjunctively well-founded transition invariant a rank need not decrease in all sub-relations if an accepting state is visited, *i.e.*, the rank of one sub-relation must decrease and all other ranks may increase.

In [12], an axiomatic approach to prove total correctness (safety property + termination) of sequential programs uses assertions connecting the initial and final values of the program variables. This must not be confused with transition invariants that capture all pairs of intermediate values in computations of arbitrary length, possibly going through loops.

It is interesting to compare our use of Ramsey's theorem in the proofs of Theorems 1 and 2 with its use in the theory of (finite) Büchi automata (see e.g. [19, 21]). The equivalence classes over computation segments in our proofs are related to the state transformers in the *transition monoid* of the Büchi automaton. In both uses of Ramsey's theorem, the sets of transformers are finite and thus induce an equivalence relation of finite index (which is why Ramsey's theorem can be applied). However, our proofs consider *finite* sets of transformers over an *infinite* state space, as opposed to transformers over a finite state space.

The termination analysis for functional programs in [10] has been the starting point of our work. The analysis is based on the comparison of infinite paths in the control flow graph and in 'size-change graphs'; that comparison can be reduced to the inclusion test for Büchi automata. The transitive closure of a (finite) set of size-change graphs can be seen as a graph representation of a special case of a transition invariant.

7. Conclusion

We have presented a (sound and relatively complete) proof rule for the temporal verification of concurrent programs. In a well-chosen instantiation, this proof rule allows one to decompose the verification problem into a number of independent smaller verification problems: one for establishing a transition invariant, and the others for establishing the disjunctive well-foundedness. The former is done in a way that is reminiscent of establishing state invariants, using a familiar inductive reasoning. The other ones amount to testing the termination of simple while loops.

Our conceptual contribution is the notion of a transition invariant, and its usefulness in temporal proofs. This notion is at the basis of our proof rule. In particular, it allows one to account for Büchi accepting conditions (and hence for fairness) in a direct way, namely by intersecting relations.

Our technical contribution is the characterization of the validity of termination or another liveness property by the existence of a disjunctively well-founded transition invariant. The application of Ramsey's theorem allows us to replace the argument that the transition relation R is contained in the *(transitive)* well-founded relation r_f induced by a ranking function f (*i.e.*, $(s, s') \in r_f$ if $f(s) > f(s')$) by the argument that the transitive closure of R is contained in a union of well-founded relations. *I.e.*, we have

$$R \subseteq r_f \qquad \text{vs.} \qquad R^+ \subseteq T_1 \cup \dots \cup T_n.$$

As outlined in Section 5, our proof rule is a starting point for the development of automated verification methods for liveness properties of concurrent programs. This development is not in the scope of this paper. In [16], we have started one line of research based on predicate abstraction as used in the already existing tools for safety properties [1, 3, 8]; many different other ways are envisageable.

Another line of research are methods to reduce the size of the transition invariants by encoding relevant specific kinds of fairness, such as weak and strong fairness, in a more direct way than encoding them in Büchi automata.

Acknowledgments This work started with discussions with Neil Jones and Chin Soon Lee during their visit in Saarbrücken in September 2002. We thank Patrick Cousot, Kedar Namjoshi and Amir Pnueli for their remarks on ranking functions and finite-state abstraction during VMCAI in January 2003. We thank Amir Pnueli for comments and suggestions, and for coining the term "disjunctive well-foundedness". We thank Bernd Finkbeiner and Konstantin Korovin for comments and suggestions.

References

[1] T. Ball, R. Majumdar, T. Millstein, and S. Rajamani. Automatic predicate abstraction of C programs. In *Proc. of PLDI'2001: Programming Language Design and Implementation*, volume 36 of *ACM SIGPLAN Notices*, pages 203–213. ACM Press, 2001.

[2] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *Proc. of PLDI'2003: Programming Language Design and Implementation*, pages 196–207. ACM Press, June 7–14 2003.

[3] S. Chaki, E. Clarke, A. Groce, S. Jha, and H. Veith. Modular verification of software components in C. In *Proc. of ICSE'2003: Int. Conf. on Software Engineering*, pages 385–395, 2003.

[4] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of POPL'1977: Principles of Programming Languages*, pages 238–252. ACM Press, 1977.

[5] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. of POPL'1979: Principles of Programming Languages*, pages 269–282. ACM Press, 1979.

[6] Y. Fang, N. Piterman, A. Pnueli, and L. D. Zuck. Liveness with invisible ranking. In Steffen and Levi [20], pages 223–238.

[7] S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In *Proc. of CAV'1997: Computer Aided Verification*, volume 1254 of *LNCS*, pages 72–83. Springer, 1997.

[8] T. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy Abstraction. In *Proc. of POPL'2002: Principles of Programming Languages*, pages 58–70. ACM Press, 2002.

[9] N. Klarlund. Progress measures and stack assertions for fair termination. In *Proc. of PODC'1992: Principles of Distributed Computing*, pages 229–240. ACM Press, 1992.

[10] C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. In *Proc. of POPL'2001: Principles of Programming Languages*, volume 36, 3 of *ACM SIGPLAN Notices*, pages 81–92. ACM Press, 2001.

[11] D. Lehmann, A. Pnueli, and J. Stavi. Impartiality, justice and fairness: The ethics of concurrent termination. In *Proc. of ICALP'1981: Int. Colloq. on Automata, Languages and Programming*, volume 115 of *LNCS*, pages 264–277. Springer, 1981.

[12] Z. Manna and A. Pnueli. Axiomatic approach to total correctness of programs. *Acta Informatica*, (3):243–263, 1974.

[13] Z. Manna and A. Pnueli. Completing the temporal picture. *Theoretical Computer Science*, 83(1):91–130, 1991.

[14] Z. Manna and A. Pnueli. *Temporal verification of reactive systems: Safety*. Springer, 1995.

[15] Z. Manna and A. Pnueli. Temporal verification of reactive systems: Progress. Draft, 1996.

[16] A. Podelski and A. Rybalchenko. Transition predicate abstraction. Draft. Available from the authors.

[17] A. Podelski and A. Rybalchenko. A complete method for the synthesis of linear ranking functions. In Steffen and Levi [20], pages 239–251.

[18] F. P. Ramsey. On a problem of formal logic. In *Proc. London Math. Soc.*, volume 30, pages 264–285, 1930.

[19] P. A. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49(2–3):217–237, 1987.

[20] B. Steffen and G. Levi, editors. *Proc. of VMCAI'2004: Verification, Model Checking, and Abstract Interpretation*, volume 2937 of *LNCS*. Springer, 2004.

[21] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, pages 133–192. Elsevier and MIT Press, 1990.

[22] A. Tiwari. Termination of linear programs. In *Proc. of CAV'2004: Computer Aided Verification*, 2004. To appear.

[23] M. Y. Vardi. Verification of concurrent programs — the automata-theoretic framework. *Annals of Pure and Applied Logic*, 51:79–98, 1991.

[24] M. Y. Vardi. Rank predicates vs. progress measures in concurrent-program verification. *Chicago Journal of Theoretical Computer Science*, 1996.

IEEE COMPUTER SOCIETY