Transition Invariants and Transition Predicate Abstraction for Program Termination

Andreas Podelski[1] and Andrey Rybalchenko[2]

[1] University of Freiburg
[2] Technische Universität München

Abstract. Originally, the concepts of transition invariants and transition predicate abstraction were used to formulate a proof rule and an abstraction-based algorithm for the verification of liveness properties of concurrent programs under fairness assumptions. This note applies the two concepts for proving termination of sequential programs. We believe that the specialized setting exhibits the underlying principles in a more direct way.

1 Introduction

Transition invariants allow one to combine several ranking functions into a single termination argument. Transition predicate abstraction automates the computation of transition invariants using automated theorem proving techniques. Together, transition invariants and transition predicate abstraction overcome critical deficiencies of the classical proof method for program termination. The classical method for proving program termination is based on the construction of a single ranking function for the entire program. This construction cannot be supported by the abstraction of a program into a finite-state program (each finite-state program with n states will contain a loop to accomodate executions with length greater than n).

Transition invariants and transition predicate abstraction were introduced in [3] and [4], respectively (we refer to [3,4] for the discussion of related work). Here, we use a uniform setting in order to present the two concepts together (as it was done in the earlier technical report [1]). Originally, the concepts of transition invariants and transition predicate abstraction were used to formulate a proof rule and an abstraction-based algorithm for the verification of liveness properties of concurrent programs under fairness assumptions. This note applies the two concepts for proving termination of sequential programs. The purpose of this note is to provide a short, direct, and comprehensive access to the underlying principles.

2 Preliminaries

We abstract away from a particular programming language and use transition relations to describe programs. To further simplify the presentation, our

P.A. Abdulla and K.R.M. Leino (Eds.): TACAS 2011, LNCS 6605, pp. 3–10, 2011.

```
l1: y := read_int();
l2: while (y > 0) {
      y := y-1;
    }
```

$\rho_1 : pc = \ell_1 \wedge pc' = \ell_2$

$\rho_1 : pc = \ell_2 \wedge pc' = \ell_2 \wedge y > 0 \wedge y' = y - 1$

$T_1 : pc = \ell_1 \wedge pc' = \ell_2$

$T_2 : y > 0 \wedge y' < y$

Fig. 1. Program ANY-Y contains unbounded non-determinism at line `l1`. The union of binary relations ρ_1 and ρ_2 is the transition relation of the program. Termination of ANY-Y cannot be proved with ranking functions ranging over the set of natural numbers (the initial rank must be at least the ordinal ω). The union of binary relations T_1 and T_2 is a transition invariant for ANY-Y. The binary relations T_1 and T_2 are both well-founded (T_1 does not have a chain longer than 1; T_2 does not have a chain longer than the value of y in its starting element). Thus we have a disjunctively well-founded transition invariant for ANY-Y, which proves the termination of ANY-Y.

definition of programs does not specify a particular set of initial states. We assume that the program can be started from any state.

Definition 1 (Transition-based program). *We define a* program *as a triple*

$$P = (\Sigma, \mathcal{T}, \rho),$$

consisting of:

- *a set of* states Σ,
- *a finite set of* transitions $\mathcal{T}$, *which can be thought of as labels of program statements, and*
- *a function* ρ *which assigns to each transition a binary transition relation over states,*

$$\rho_\tau \subseteq \Sigma \times \Sigma, \qquad \text{for } \tau \in \mathcal{T}.$$

The transition relation of P, *denoted* R_P, *comprises the transition relations* ρ_τ *of all transitions* $\tau \in \mathcal{T}$, *i.e.,*

$$R_P = \bigcup_{\tau \in \mathcal{T}} \rho_\tau.$$

A program state is a valuation of program variables, including the program counter. An assertion over program variables denotes a set of program states, while an assertion over program variables and their primed versions denotes a binary relation over program states. We identify sets and binary relations by assertions denoting them.

See Figure 1 for an example of a program and its formal representation in terms of its transition relations.

A program P is *terminating* if its transition relation R_P is well-founded. This means that the relation R_P does not have an infinite chain, i.e., an infinite sequence

$$s_1, s_2, s_3, \ldots$$

where each pair of successive states (s_i, s_{i+1}) is contained in the relation R_P.

3 Disjunctively Well-Founded Transition Invariants

In this section we give a brief description of terminology and results of [3] restricted to termination ([3] also deals with general liveness properties and fairness). We write r^+ to denote the transitive closure of a relation r.

Definition 2 (Transition invariant). *Given a program $P = (\Sigma, \mathcal{T}, \rho)$, a transition invariant T is a binary relation over states T that contains the program's transition relation R_P^+, i.e.,*

$$R_P^+ \subseteq T \ .$$

Definition 3 (Disjunctively well-founded relation). *A relation T is* disjunctively well-founded *if it is a finite union of well-founded relations:*

$$T = T_1 \cup \cdots \cup T_n \ .$$

Theorem 1 (Proof rule for termination). *A program P is terminating if and only if there exists a disjunctively well-founded transition invariant for P.*

Proof. "Only if" ($\Rightarrow$) is trivial: if P is terminating, then both R_P and R_P^+ are well-founded. Choose $n = 1$ and $T_1 = R_P^+$.

"If" ($\Leftarrow$): we show that if P is *not* terminating and $T_1 \cup \cdots \cup T_n$ is a transition invariant, then some T_i is not well-founded. Nontermination of P means there exists an infinite computation:

$$s_0 \xrightarrow{\tau_1} s_1 \xrightarrow{\tau_2} s_3 \xrightarrow{\tau_3} \ldots$$

Let a choice function f satisfy

$$f(k, \ell) \in \{\ T_i \mid (s_k, s_\ell) \in T_i\ \}$$

for $k, \ell \in \mathbb{N}$ with $k < \ell$. (The condition $R_P^+ \subseteq T_1 \cup \cdots \cup T_n$ implies that f exists, but does not define it uniquely.) Define equivalence relation $\simeq$ on f's domain by

$$(k, \ell) \simeq (k', \ell') \text{ if and only if } \ f(k, \ell) = f(k', \ell')$$

Relation $\simeq$ is of finite index since the set of T_i's is finite. By Ramsey's Theorem there exists an infinite sequence of natural numbers $k_1 < k_2 < \ldots$ and fixed $m, n \in \mathbb{N}$ such that

$$(k_i, k_{i+1}) \simeq (m, n) \qquad \text{for all } i \in \mathbb{N}.$$

Hence $(s_{k_i}, s_{k_{i+1}}) \in T_{f(m,n)}$ for all i. This is a contradiction: $T_{f(m,n)}$ is not well-founded. □

The proof of Theorem 1 uses a weak version of Ramsey's theorem. This version states that every infinite complete graph that is colored with finitely many colors contains a monochrome infinite *path* (as opposed to a monochrome infinite *complete subgraph*, in the strong version of Ramsey's theorem).

```
l1: while (x => 0) {
      y := 1;
l2:   while (y < x) {
        y := y+1;
      }
      x := x-1;
    }
```

$$\rho_1 : pc = \ell_1 \wedge pc' = \ell_2 \wedge x \geq 0 \wedge x' = x \wedge y' = 1$$
$$\rho_2 : pc = \ell_2 \wedge pc' = \ell_2 \wedge y < x \wedge x' = x \wedge y' = y + 1$$
$$\rho_3 : pc = \ell_2 \wedge pc' = \ell_1 \wedge y \geq x \wedge x' = x - 1 \wedge y' = y$$

$$T_1 : pc = \ell_1 \wedge pc' = \ell_2$$
$$T_2 : pc = \ell_2 \wedge pc' = \ell_1$$
$$T_3 : x \geq 0 \wedge x' < x$$
$$T_4 : x - y > 0 \wedge x' - y' < x - y$$

Fig. 2. Program BUBBLE contains a nested loop. Termination of BUBBLE is classically shown with the *lexicographic* ranking function $\langle x, x - y \rangle$ defined by the pair of the ranking functions x and $x - y$. The disjunctively well-founded transition invariant shown (the union of binary relations $T_1, \ldots, T_4$) does not prescribe an order between the two ranking functions x and $x - y$ (which have T_3 and T_4 as the corresponding *ranking relations*).

```
l: while (x > 0 && y > 0) {
     if (read_int()) {
       (x, y) := (x-1, x);
     } else {
       (x, y) := (y-2, x+1);
     }
   }
```

$$\rho_1 : pc = pc' = \ell \wedge x > 0 \wedge y > 0 \wedge x' = x - 1 \wedge y' = x$$
$$\rho_2 : pc = pc' = \ell \wedge x > 0 \wedge y > 0 \wedge x' = y - 2 \wedge y' = x + 1$$

$$T_1 : x > 0 \wedge x' < x$$
$$T_2 : y > 0 \wedge y' < y$$
$$T_3 : x + y > 0 \wedge x' + y' < x + y$$

Fig. 3. Program CHOICE contains a non-determinstic choice between two simultaneous assignment statements in the loop body. The disjunctively well-founded transition invariant shown (the union of binary relations $T_1, \ldots, T_3$) presents the ranking relations for the three ranking functions x, y, and $x + y$, none of which by itself suffices to prove terminations.

See Figures 1, 2, 3, and 4 for examples of disjunctively well-founded transition invariants.

As a consequence of the above theorem, we can prove termination of a program P as follows. We compute a disjunctively well-founded superset of the transitive closure of the transition relation of the program P, i.e., we construct a finite number of well-founded relations $T_1, \ldots, T_n$ whose union covers R_P^+. We need to show that the inclusion $R_P^+ \subseteq T_1 \cup \cdots \cup T_n$ indeed holds, and we need to show that each of the relations $T_1, \ldots, T_n$ is indeed well-founded. Transition predicate abstraction can be used to obtain an automation of the three steps, as shown in the next section.

```
l: while (x > 0 && y > 0) {
     if (read_int()) {
       x := x-1;
       y := read_int();
     } else {
       y := y-1;
     }
   }
```

$$\rho_1 : pc = pc' = \ell \wedge x > 0 \wedge y > 0 \wedge x' = x - 1$$

$$\rho_2 : pc = pc' = \ell \wedge x > 0 \wedge y > 0 \wedge x' = x \wedge y' = y - 1$$

$$T_1 : x \geq 0 \wedge x' < x$$

$$T_2 : y > 0 \wedge y' < y$$

Fig. 4. The program XORY shown contains a non-deterministic choice between two simultaneous assignment statements in the loop body. The first one decrements x and erases y (assigns a non-deterministic value to y). The second decrements y. The validity of the disjunctively well-founded transition invariant shown (the union of the ranking relations for the two ranking functions x and y) is shown by transition predicate abstraction.

4 Transition Predicate Abstraction (TPA)

A transition predicate is a binary relation over program states. Transition predicate abstraction [4] is a method to compute transition invariants, just as predicate abstraction is a method to compute invariants.

Definition 4 (Set of abstract transitions $\mathcal{T}_{\mathcal{P}}^{\#}$). *Given the set of transition predicates $\mathcal{P}$, the set of* abstract transitions $\mathcal{T}_{\mathcal{P}}^{\#}$ *is the set that contains for every subset of transition predicates $\{p_1, \ldots, p_m\} \subseteq \mathcal{P}$ the conjunction of these transition predicates, i.e.,*

$$\mathcal{T}_{\mathcal{P}}^{\#} = \{p_1 \wedge \ldots \wedge p_m \mid 0 \leq m \text{ and } p_i \in \mathcal{P} \text{ for } 1 \leq i \leq m\} \ .$$

The set of abstract transitions $\mathcal{T}_{\mathcal{P}}^{\#}$ is closed under intersection, and it contains the assertion *true* (the empty intersection, corresponding to the case $m = 0$), which denotes the set of all pairs of program states.

Example 1. Consider the following set of transition predicates.

$$\mathcal{P} = \{x' = x, x' < x, y' < y\}$$

The set of abstract transitions $\mathcal{T}_{\mathcal{P}}^{\#}$ is

$$\{\mathsf{true}, x' = x, x' < x, y' < y, x' = x \wedge y' < y, x' < x \wedge y' < y, \mathsf{false}\} \ .$$

The abstract transition written as true is the set of all state pairs $\Sigma \times \Sigma$ and is the empty conjunction of transition predicates. The abstract transition written as false is the empty relation; e.g., the conjunction of $x = x'$ and $x > x'$ is false.

We next define a function that assigns to a binary relation T over states the least (wrt. inclusion) abstract transition that is a superset of T.

Definition 5 (Abstraction function α). *A set of transition predicates $\mathcal{P}$ defines the* abstraction function

$$\alpha : 2^{\Sigma \times \Sigma} \to \mathcal{T}^{\#}_{\mathcal{P}}$$

which assigns to a relation $r \subseteq \Sigma \times \Sigma$ the smallest abstract transition that is a superset of r, i.e.,

$$\alpha(r) = \bigwedge \{p \in \mathcal{P} \mid r \subseteq p\}.$$

We note that α is extensive, i.e., the inclusion

$$r \subseteq \alpha(r)$$

holds for any binary relation over states $r \subseteq \Sigma \times \Sigma$.

Example 2. After taking the transition predicates $x > 0$ and $y > 0$, which leave the primed variables unconstrained, into consideration the application of the abstraction function α to the transition relations ρ_1 and ρ_2 of the program in Figure 4 results in the following abstract transitions.

$$\alpha(\rho_1) = x > 0 \wedge y > 0 \wedge x' < x$$
$$\alpha(\rho_2) = x > 0 \wedge y > 0 \wedge x' = x \wedge y' < y$$

We next present an algorithm that uses the abstraction α to compute (a set of abstract transitions that represents) a transition invariant. The algorithm terminates because the set of abstract transitions $\mathcal{T}^{\#}_{\mathcal{P}}$ is finite.

Algorithm 1 (TPA).
Transition invariants via transition predicate abstraction.

Input: *program $P = (\Sigma, \mathcal{T}, \rho)$*
set of transition predicates $\mathcal{P}$
abstraction α defined by $\mathcal{P}$ (according to Def. 5)

Output: *set of abstract transitions $P^{\#} = \{T_1, \ldots, T_n\}$*
such that $T_1 \cup \cdots \cup T_n$ is a transition invariant

$P^{\#} := \{\alpha(\rho_\tau) \mid \tau \in \mathcal{T}\}$
repeat
  $P^{\#} := P^{\#} \cup \{\alpha(T \circ \rho_\tau) \mid T \in P^{\#},\ \tau \in \mathcal{T},\ T \circ \rho_\tau \neq \emptyset\}$
until *no change*

Our notation $P^{\#}$ for the set of abstract transitions computed by the TPA algorithm stems from [4]. There, $P^{\#}$ is called an abstract transition program. In contrast to [4] we do not consider edges between the abstract transitions, since they only needed for keeping track of fairness assumption when proving fair termination and liveness properties.

Theorem 2 (TPA). *Let $P^\# = \{T_1, \ldots, T_n\}$ be the set of abstract transitions computed by Algorithm* TPA*. If every abstract relation $T_1, \ldots, T_n$ is well-founded, then program P is terminating.*

Proof. The union of the abstract relations $T_1 \cup \cdots \cup T_n$ is a transition invariant. If every abstract relation $T_1, \ldots, T_n$ is well-founded, the union $T_1 \cup \cdots \cup T_n$ is a disjunctively well-founded transition invariant and by Theorem 1 the program P is terminating. □

Example 3. Consider the program P in Figure 4 and the set of transition predicates $\mathcal{P}$ in Example 1. The output of Algorithm TPA is

$$P^\# = \{x > x', \quad x = x' \wedge y > y'\}$$

Both abstract transitions in $P^\#$ are well-founded. Hence P is terminating. In fact, it is sufficient to show the well-foundedness of the (simpler) binary relations T_1 and T_2 given in Figure 4. The transition invariant given there is valid because it contains the one defined by $P^\#$.

Each abstract transition in $P^\#$ (the representation of a transition invariant computed by the transition-predicate abstraction-based algorithm) is a conjunction of transition predicates. Thus it corresponds to a conjunction $g \wedge u$ of a *guard* formula g which contains only unprimed variables, and an *update* formula u which contains primed variables, for example $x > 0 \wedge x > x'$. Thus it denotes the transition relation of a *simple* while program of the form `while` g { u }, for example, `while` $(x > 0)$ { `assume`$(x > x')$; $x := x'$ }. The well-foundedness of the abstract transition is thus equivalent to the termination of the simple while program. Since we have fast and complete procedures that find ranking functions for such programs [2], we can automate also the third step of transition invariant-based termination proofs, as outlined in the previous section.

5 Conclusion

We have presented disjunctively well-founded transition invariants as the basis of a new proof rule for program termination, and transition predicate abstraction as the basis of its automation. As a result, we obtain the foundation for a new class of automatic methods for proving program termination.

Acknowledgements. Discussions with Neil Jones and Chin Soon Lee started this work. We thank Amir Pnueli for inspiration and encouragement, and for suggesting the terminology of disjunctively well-founded transition invariants.

References

1. Podelski, A., Rybalchenko, A.: Software model checking of liveness properties via transition invariants. Technical Report MPI-I-2003-2-004, Max-Planck-Institut für Informatik (December 2003),
`http://domino.mpi-inf.mpg.de/internet/reports.nsf/NumberView/2003-2-004`

2. Podelski, A., Rybalchenko, A.: A complete method for the synthesis of linear ranking functions. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 239–251. Springer, Heidelberg (2004)
3. Podelski, A., Rybalchenko, A.: Transition invariants. In: LICS 2004: Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, Washington, DC, USA, pp. 32–41. IEEE Computer Society Press, Los Alamitos (2004)
4. Podelski, A., Rybalchenko, A.: Transition predicate abstraction and fair termination. In: POPL 2005: Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, vol. 32, pp. 132–144. ACM, New York (2005)