Transition Invariants and Transition Predicate Abstraction for Program Termination

Andreas Podelski

January 10, 2012

Ramsey's theorem

every infinite complete[1] directed graph
that is edge-colored with finitely many colors
contains a monochrome[2] infinite complete subgraph

[1] every node has an edge to every other node
[2] all edges have the same color

termination

a program P with transition relation R_P is *terminating*

- iff there is no infinite computation $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$
- iff there is no sequence of states $s_1, s_2, \ldots$ such that the (s_i, s_{i+1})'s are contained in the transition relation R_P
- iff the relation R_P does not have an infinite chain
- iff the transition relation R_P is well-founded

here, for simplification, computations can start in any state (every state is an initial state of program P)

predicate abstraction for termination?

- we use predicate abstraction of program P to construct a finite abstract reachability graph, called $P^{\#}$
- every computation of P corresponds to path in $P^{\#}$ (but not every path corresponds to a computation)
- non-reachability by any path in graph $P^{\#} \Rightarrow$ non-reachability by any computation of program P
- finiteness of paths in $P^{\#} \Rightarrow$ finiteness of computations of P
- if computations of P have unbounded length,
  then paths in $P^{\#}$ have unbounded length
  $\Rightarrow$ exists cycle in $P^{\#}$
  $\Rightarrow$ exists infinite paths in $P^{\#}$

backward computation for termination?

- terminatingStates
  = states s that do not have an infinite computation
- program terminates iff initialStates $\subseteq$ terminatingStates
- exitStates
  = set of states without successor
- weakestPrecondition(exitStates) $\cup$ exitStates
  = set of states with computations of length ≤ 1
- etc.
- compute terminatingStates backwards, starting from exitStates, until a fixpoint is reached
- check of inclusion requires abstraction of fixpoint from below
- no good techniques for underapproximation known!

transition invariant

given a program P with transition relation R_P,

relation T is a *transition invariant*
if it contains the transitive closure of the transition relation:

$$R_P^+ \subseteq T$$

- T inductive transition invariant if

$$R_P \subseteq T \quad \text{and} \quad T \circ R_P \subseteq T$$

- relational composition:
  $R_1 \circ R_2 = \{(s, s'') \mid (s, s') \in R_1,\ (s', s'') \in R_2\}$

disjunctively well-founded relation

relation T is *disjunctively well-founded*
if it is a finite union of well-founded relations:

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

union of well-founded relations is itself not well-founded, in general

proof rule for termination

program P is terminating iff there exists a disjunctively well-founded transition invariant T for P

- transition invariant:

  $$R_P^+ \subseteq T$$

  validity shown via an inductive transition invariant that entails T

- disjunctively well-founded:

  $$T = T_1 \cup \cdots \cup T_n \ \text{ where } \ T_1, \ldots, T_n \text{ well-founded}$$

  well-foundedness of simple relations $T_1, \ldots, T_n$ decidable

completeness of proof rule

- “only if” $(\Rightarrow)$
- program P is terminating *implies* there exists a disjunctively well-founded transition invariant for P
- trivial:
- if P is terminating, then both R_P and R_P^+ are well-founded
- choose $n = 1$ and $T_1 = R_P^+$

ranking function and ranking relation

given: ranking function f
for program P with transition relation R_P
ranking relation r_f defined by:

$$r_f = \{(s_1, s_2) \mid f(s_2) < f(s_1)\}$$

- ranking relation r_f is well-founded
- $R_P \subseteq r_f$
- $R_P^+ \subseteq r_f$ (since r_f is transitive)

soundness of proof rule

- "If" ($\Leftarrow$):
- a program P is terminating *if* there exists a disjunctively well-founded transition invariant for P
- contraposition:
  if
  $R_P^+ \subseteq T$,
  $T = T_1 \cup \cdots \cup T_n$, and
  P is *not* terminating,
  then
  at least one of $T_1, \ldots, T_n$ is not well-founded

assume $R_P^+ \subseteq T$, $T = T_1 \cup \cdots \cup T_n$, P non-terminating

- there exists an infinite computation of P:

$$s_0 \to s_1 \to s_2 \to \ldots$$

- each pair (s_i, s_j) lies in one of T_1, ..., T_n
- one of T_1, ..., T_n contains infinitely many pairs (s_i, s_j)
- say, T_k
- contradiction if we obtain an infinite chain in T_k (since T_k is a well-founded relation)
- in general, the pairs (s_i, s_j) do not form a chain (are not consecutive)

Ramsey's theorem

every infinite complete[1] directed graph
that is edge-colored with finitely many colors
contains a monochrome[2] infinite complete subgraph

[1] every node has an edge to every other node
[2] all edges have the same color

assume $R_P^+ \subseteq T$, $T = T_1 \cup \dots \cup T_n$, P non-terminating

- there exists an infinite computation of P:

$$s_0 \to s_1 \to s_2 \to \dots$$

- take infinite complete graph formed by s_i's
- edge = pair (s_i, s_j) in R_P^+, i.e., in one of $T_1, \dots, T_n$
- edges can be colored by n different colors
- exists monochrome infinite complete subgraph
- all edges in subgraph are colored by, say, T_k
- infinite complete subgraph has an infinite path
- obtain infinite chain in T_k
- contradicition since T_k is a well-founded relation

assume $R_P^+ \subseteq T$, $T = T_1 \cup \cdots \cup T_n$, P non-terminating

- there exists an infinite computation of P:

$$s_0 \to s_1 \to s_2 \to \ldots$$

- 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 $k < \ell$
- 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

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

example program: ANY-Y

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

$$\rho_1 : pc = \ell_1 \wedge pc' = \ell_2$$
$$\rho_1 : pc = \ell_2 \wedge pc' = \ell_2 \wedge y > 0 \wedge y' = y - 1$$

$$T_1 : pc = \ell_1 \wedge pc' = \ell_2$$
$$T_2 : y > 0 \wedge y' < y$$

example program $\textsc{Bubble}$ (nested loop)

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

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

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

program CHOICE

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

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

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

prove termination of program P

- compute a disjunctively well-founded superset of the transitive closure of the transition relation of the program P, i.e.,
- construct a finite number of well-founded relations $T_1, \ldots, T_n$ whose union covers R_P^+

prove termination in 3 steps

1. find a finite number of relations $T_1, \ldots, T_n$
2. show that the inclusion $R_P^+ \subseteq T_1 \cup \cdots \cup T_n$ holds
3. show that each relation $T_1, \ldots, T_n$ is well-founded

transition predicate abstraction

- transition predicate: a binary relation over program states
- transition predicate abstraction: a method to compute transition invariants

$\mathcal{T}_{\mathcal{P}}^{\#}$, domain of abstract transitions

- given the set of transition predicates $\mathcal{P}$
- abstract transition = conjunction of transition predicates
- $$\mathcal{T}_{\mathcal{P}}^{\#} = \{p_1 \wedge \ldots \wedge p_m \mid 0 \leq m \text{ and } p_i \in \mathcal{P} \text{ for } 1 \leq i \leq m\} \ .$$
- $\mathcal{T}_{\mathcal{P}}^{\#}$ is closed under intersection
- $\mathcal{T}_{\mathcal{P}}^{\#}$ contains the assertion *true*
  empty intersection, corresponding to the case $m = 0$
  denotes set of all pairs of program states

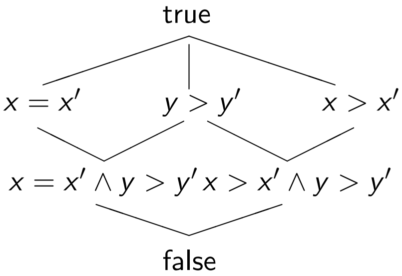
example

set of transition predicates:

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

set of abstract transitions:

$$\mathcal{T}_{\mathcal{P}}^{\#} = \{\mathsf{true}, x' = x, x' < x, y' < y, x' = x \wedge y' < y, x' < x \wedge y' < y, \mathsf{false}\}$$



add transition predicates: $x > 0$ and $y > 0$

special case, leave the primed variables unconstrained

abstraction function α

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 between states r the smallest abstract transition that is a superset of r, i.e.,

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

note that α is extensive:

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

program $\textsc{Choice}$

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

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

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

Algorithm (TPA)

Transition invariants via transition predicate abstraction.

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

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

correctness of algorithm TPA

let $\{ T_1, \ldots, T_n \}$ 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

- union of 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
- thus, the program P is terminating

example

consider program P and the set of transition predicates $\mathcal{P}$
output of Algorithm TPA is

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

both abstract transitions are well-founded
hence P is terminating

- each abstract transition is a conjunction of transition predicates
- 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, $x > 0 \wedge x > x'$ corresponds to `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
- we have fast and complete procedures that find ranking functions for simple while programs
  (next lecture)

conclusion

- disjunctively well-founded transition invariants: basis of a new proof rule for program termination
- (next) transition predicate abstraction: basis of automation of proof rule
- new class of automatic methods for proving program termination
  - combine multiple ranking functions for reasoning about termination of complex program fragments
  - rely on abstraction techniques to make this reasoning efficient