Transition Predicate Abstraction and Fair Termination

Andreas Podelski and Andrey Rybalchenko

Max-Planck-Institut für Informatik Saarbrücken, Germany

POPL 2005
Introduction

- Verification
- Tools using predicate abstraction
  - Finite state machine
- Safety properties
  - Guarantee the absence of bad events (e.g., deadlock)
  - Check automated
- Liveness properties
  - Ensure that good events eventually happen
  - manual work (so far) 😞
Introduction

- Presented method
  - liveness property under fairness assumptions
  - Automated 😊
Goal

- **Given**
  - Program

- **Requested**
  - liveness properties
    - do hold
    - don’t hold
Solution

Procedure

1. Reduction
2. Program $P \rightarrow P#$
3. Termination checks
4. Fairness checks
5. Interpretation of result

while (...) {
    ...
}
good();

Is good() called?

Fair termination

Yes! / No!
1. Reduction

- Reduction of the verification problem for general temporal properties to the one for fair termination

Verification problem → Fair termination problem
2. $P \rightarrow P#$

- Transition predicate abstraction-based transformation

```
while (...) {
    ...
}  
good();
```

Program $P$ abstract-transition program $P#$
2. $P \rightarrow P#$

- **Algorithm**
- **Informal idea**
  - **Build a graph**
  - **Nodes are abstract transitions**
    - represents some reachability relationships from the starting node
  - **Edges are transitions**
    - Like arrows in a finite state machine

```
input
P: program with finite set of transitions $T$
$\mathcal{P}$: finite set of transition predicates
output
abstract-transition program $P#$ with:
  $V$: set of nodes labeled by abstract transitions
  $E$: set of edges labeled by transitions $\tau$
begin
  $Q := \text{empty queue}$
  $\alpha := \lambda T. \ W \{ p \in P \mid T \subseteq p \}$
  $v_0 := \text{new node labeled by } Id$
  $V := \{ v_0 \}$
  enqueue($Q$, $v_0$)
  $E := \emptyset$
  while $Q$ not empty do
    $u := \text{dequeue}(Q)$
  foreach $\tau \in T$ do
    $T := \alpha(T_u \circ p_\tau)$
    if $T = \emptyset$ then continue with next $\tau$ fi
    if exists $w \in V$ such that $T = T_w$ then
      $v := w$
    else
      $v := \text{new node labeled by } T$
      $V := V \cup \{ v \}$
      enqueue($Q$, $v$)
    fi
    $(u, v) := \text{new edge labeled by } \tau$
    $E := E \cup \{ (u, v) \}$
  od
  od
end.
```
2. $P \rightarrow P^\#$

- **Example program**

  local $y$ : natural

  $\ell_0$: while $y > 0$ do
  $\ell_1$: $y := y - 1$
  $\ell_2$: skip

  $\ell_3$: 

  | program $P$ |

  abstract-state program $P'$

  $\tau_1$: $y > 0$, $y' = y - 1$

  $\tau_2$: $y = 0$, $y' = y$

  $S_1$: at $\ell_0$, $y > 0$

  $S_2$: at $\ell_0$, $y = 0$

  $S_3$: at $\ell_3$, $y = 0$

  abstract-transition program $P^\#$

  $T_1$: at $\ell_0, \ell'_0$, $y > 0, y' \leq y - 1$

  $T_2$: at $\ell_0, \ell'_3$
2. $P \rightarrow P^#$

- **Given**
  - program $P$
  - $P = \{ y > 0, y' \leq y - 1 \}$

- **Requested**
  - Graph $P^#$

- **Example**
  - $y' = y - 1$ becomes
  - $y' \leq y - 1$

**abstract-transition program $P^#$**

**abstract-state program $P'$**
3. Termination checks

- Mark nodes of P# as „terminating“

![Diagram showing program P# and terminating nodes]
3. Termination checks

- For all nodes in P#
  - If well-founded(node)
    - terminating

- A set S is well-founded iff
  Every non-empty subset of S has a minimal element

- More details: See Paper
4. Fairness checks

- Mark nodes of P# as „fair/unfair“
4. Fairness checks

- **Just Fairness** means being just to everyone.
- **Compassionate Fairness** means being compassionate to everyone.
5. Interpretation of resulting P#

- Return „property verified“ if each fair node is marked terminating

![Diagram showing a program P# with nodes labeled unfair and fair & t, indicating property verified.]

program P#
fair/unfair nodes
Summary

- Automated method for the verification of liveness properties under full fairness assumptions (justice and compassion).
- Extended the applicability of predicate abstraction-based program verification to the full set of temporal properties.

```
while (..) {
    ...
}  
good();
```

1. Is good() called?
2. Fair termination
3. + 4
4. Yes! / No!
Personal opinion

- Confusing names
  - justice vs. compassion
  - transition vs. abstract transition
- Hard to understand
- Interesting research topic
Termination is an example of a basic liveness property. We are working on the next generation of TERMINATOR that will prove general liveness properties under fairness assumptions.

- Microsoft Research
- Max-Planck-Institut für Informatik
- Universität Freiburg, Institut für Informatik
- EPFL
Questions?
Reduction

\[
\text{reduction} \quad \downarrow \quad \text{Verification} \quad \downarrow \quad \text{Fair termination}
\]

\text{Fairness} \quad \text{Termination}

- Justice
- Compassion
- Well-foundedness
4. Fairness checks

- **Justice**

  \[ T_2 \]

  \[ T_1 \]

  \[ \text{continuously enabled } T_1: \overline{11111}.. \]

- **Compassion**

  \[ T_3 \]

  \[ T_4 \]

  \[ T_2 \]

  \[ T_1 \]

  \[ \text{infinitely often } T_2 \text{ enabled: } \overline{1001001}.. \]
Justice

- Justice is sensitive to the enabledness of transitions.
  - A transition $\tau$ is enabled on the state $s$ if the set of states $\{s' \mid (s, s') \in \rho_\tau\}$ is not empty.
- We write $\text{En}(\tau)$ for the set of states on which the transition $\tau$ is enabled.
- Justice requirement is represented by a set $J$ of just transitions, $J \subseteq T$. Every just transition that is continually enabled beyond a certain point must be taken infinitely often.
Further example

\[
\begin{aligned}
\tau_1: \quad & x = 1, x' = x, \\
& y' = y + 1 \\
\tau_4: \quad & x = 1, x' = 0, \\
& y' = y \\
\tau_2: \quad & x = 0, x' = 0, \\
& y' = y \\
\tau_3: \quad & x = 0, x' = x, \\
& y > 0, y' = y - 1 \\
\end{aligned}
\]

local \(x, y\) : integer where \(x = 1, y = 1\)

\[
P_1 ::= \begin{cases}
\ell_0: \text{while } x - 1 \\
\ell_1: y := y + 1 \\
\ell_2: \text{while } y > 0 \\
\ell_3: y := y - 1
\end{cases}
\]

\[
P_2 ::= \begin{cases}
m_0: x := 0 \\
m_1:
\end{cases}
\]

\[
\begin{aligned}
x & = 1, \\
x' & = x, y' = y + 1 \\
x & = 0, \\
x' & = x, y' = y - 1
\end{aligned}
\]

\[
\begin{aligned}
x & = 0, \\
x' & = x, y' = y - 1
\end{aligned}
\]