On proving liveness properties of programs

Alexey Gotsman
University of Cambridge

joint work with Byron Cook,
Andreas Podelski, and Andrey Rybalchenko

BCTCS’06, 6 April 2006
# State-of-the-art

<table>
<thead>
<tr>
<th>Systems</th>
<th>Model checking</th>
<th>Abstraction</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLAM, BLAST</td>
<td>Symbolic</td>
<td>Automatic</td>
<td>Safety</td>
</tr>
<tr>
<td>JPF, Bandera</td>
<td>Explicit</td>
<td>User-defined</td>
<td>Full LTL</td>
</tr>
<tr>
<td>?</td>
<td>Symbolic</td>
<td>Automatic</td>
<td>Full LTL</td>
</tr>
</tbody>
</table>
Formal setting

- Program $P$ (transition system)
- Property $\varphi$ – LTL
- Fairness requirements

Does program $P$ satisfy property $\varphi$ under the given fairness requirements?
Fairness requirements

- $\mathcal{C}$ – set of compassion requirements $\langle p, q \rangle$
  - a.k.a. strong fairness
- Computation $\sigma$ is fair wrt compassion requirement $\langle p, q \rangle$ if
  - either there exist finitely many $p$-states in $\sigma$
  - or there exist infinitely many $q$-states in $\sigma$
- Intuition: if you request something sufficiently many times ($p$), then eventually you will receive it ($q$)
- Computation is fair if it is fair wrt all the compassion requirements
From liveness to fair termination

- A program is fair terminating if it has no infinite fair computation
- Property $\varphi \Rightarrow$ Streett automaton $A_{\neg \varphi}$
- Program $P_{\neg \varphi} = P || A_{\neg \varphi}$
- Compassion requirements on $P_{\neg \varphi}$:
  - requirements on $P$
  - requirements from the accepting condition of $A_{\neg \varphi}$
- The program $P$ satisfies the property $\varphi$ under the fairness requirements iff the program $P_{\neg \varphi}$ is fair terminating
Fair computation segments

- $\sigma$ – computation segment
  - a finite fragment of a computation
- $\sigma$ is fair wrt the compassion requirement $\langle p, q \rangle$ if it
  - either does not visit any $p$-states
  - or visits some $q$-state
- $\sigma$ is fair if it is fair wrt every compassion requirement
- Intuition: repeating a fair computation segment gives a fair computation
Proving fair termination

- Binary reachability relation for fair termination:
  \[ \mathcal{R} = \{ \langle s_1, s_n \rangle \mid \exists \text{ fair computation segment } \sigma = s_1, \ldots, s_n \} \]

- Relation \( T \) is disjunctively well-founded iff it is a finite union of well-founded relations.

Theorem (Pnueli, Podelski, Rybalchenko, 2005)

*The program \( P \) is fair terminating iff there exists a disjunctively well-founded relation \( T \) such that \( \mathcal{R} \subseteq T \)*

We will construct the relation \( T \) by counterexample-guided refinement
Fair computation paths

- $\pi$ – path
  - a finite sequence of program statements
- Each computation has the corresponding path
- $\pi$ is fair if some computation segment $\sigma$ obtained by executing statements in $\pi$ is fair
- Path relation of a path $\pi = \tau_1 \ldots \tau_n$: $\rho_\pi = \rho_{\tau_1} \circ \ldots \circ \rho_{\tau_n}$
- We will try to cover $\rho_\pi$ for each $\pi$ by a disjunctively well-founded relation
Construction of fair termination arguments

input
Program $P$ with fairness assumptions
begin
$T := \emptyset$
repeat
if exists path $\pi$ such that $\text{fair}(\pi)$ and $\rho_{\pi} \not\subseteq T$ then
if well-founded($\rho_{\pi}$) then
$T := T \cup \{\rho_{\pi}\}$
else
return “Counterexample path $\pi$”
else
return “Fair termination argument $T$”
end.
end.
Program transformation (1)

Solution: Transform program \( P \) to program \( \hat{P} \) such that the set of reachable states of \( \hat{P} \) corresponds the relation \( R \)

Variables of the program \( \hat{P} \):
- Variables of the program \( P \): \( v_1, \ldots, v_n, pc \)
  - record the current state (the end of the current computation segment)
- Pre-variables: \( 'v_1, \ldots, 'v_n, 'pc \)
  - record the beginning of the current computation segment
  - initially equal to their counterparts in \( P \)
- Variables for keeping track of fairness: \texttt{in\_p}_1, \ldots, \texttt{in\_p}_m, \texttt{in\_q}_1, \ldots, \texttt{in\_q}_m
  - \( \texttt{in\_p}_i = 1 \) iff there was a \( p \)-state on the current computation segment
  - \( \texttt{in\_q}_i = 1 \) iff there was a \( q \)-state on the current computation segment
Program transformation (2)

\[
L: \text{stmt;}
\]

\[
\Downarrow
\]

\[
L: \text{fair} = ((!p_1 \&\& \neg \text{in}_p_1) \parallel q_1 \parallel \text{in}_q_1) \&\& \\
\ldots \\
((!p_m \&\& \neg \text{in}_p_m) \parallel q_m \parallel \text{in}_q_m);
\]

assert(!fair \parallel T(pc, \text{'pc}, v_i, \text{'v}_i));

if (nondet()) {
    \'
v_i = v_i; \quad /* for each i \in 1..n */
    \'
p_c = L;
    in_p_i = 0; \quad /* for each i \in 1..m */
    in_q_i = 0; \quad /* for each i \in 1..m */
}

if (p_i) in_p_i = 1; \quad /* for each i \in 1..m */
if (q_i) in_q_i = 1; \quad /* for each i \in 1..m */
stmt;
Error-state is unreachable in program $\hat{P}$ iff $T$ is a valid fair termination argument

Can apply a safety checker (SLAM, BLAST) to verify this

If the check fails, the counterexample produced by model checker is the required path $\pi$
Experimental results

- Prototype implementation for C programs
- SLAM as a safety checker
- Podelski & Rybalchenko’s algorithm for synthesis linear of ranking functions
- Property:
  \[ G(KeEnterCriticalRegion \Rightarrow F KeLeaveCriticalRegion) \]

<table>
<thead>
<tr>
<th>Driver</th>
<th>Time (seconds)</th>
<th>Lines of code</th>
<th>True bugs</th>
<th>False bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td>1K</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>314</td>
<td>7K</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2344</td>
<td>15K</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>3122</td>
<td>20K</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1R</td>
<td>16</td>
<td>1K</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4R</td>
<td>3217</td>
<td>20K</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>