On-the-fly Parameterized Boolean Program Exploration

Peizun Liu, Zhaoliang Liu
Northeastern University, Boston, USA

FMCAD 2013 Student Forum
Portland, OR, USA

Oct. 21, 2013

Target: unbounded-thread replicated Boolean programs

Boolean Program

```
decl s := 1;
main(){
    decl l := 0;
 1: s := 0;
 2: goto 3, 6;
 3: assume(s);
 4: l := 1;
 5: goto 7;
 6: assume(!s);
 7: s:=!s;
 8: assert(!l);
}
```

n

Target: unbounded-thread replicated Boolean programs

C Program

```
int x = 1;      //shared
int main(){
    int y = 0; //local
    x = 0;
    if(x)
        y = 1;
    x = !x;
    assert(!y);
    return 0;
}
```

n

Boolean Program

```
decl s := 1;      //shared
main(){
    decl l := 0; //local
 1: s := 0;
 2: goto 3, 6;
 3: assume(s);
 4: l := 1;
 5: goto 7;
 6: assume(!s);
 7: s:=!s;
 8: assert(!l);
}
```

n

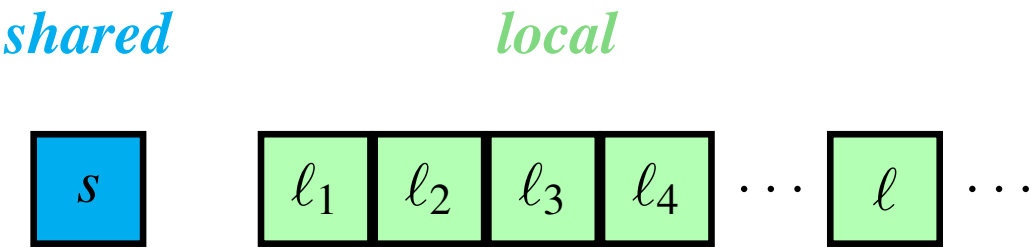
Why do we care?

- result from predicate-abstracting concurrent C programs

Goal: checking program state reachability

Definition

Given: *program state* (s, ℓ)*, with shared component* s *and local component* ℓ
Task: *check if there exists a reachable global state of the following form:*



A solved problem?

Classical solution: *Backward Reachability Analysis*

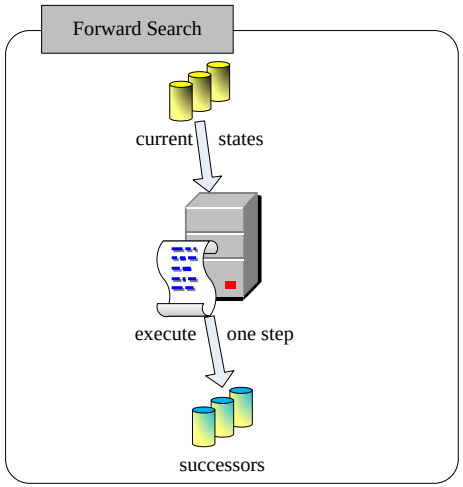
- *Well (and Better) Quasi-Ordered Transition Systems*, [P. Abdulla, 2010]

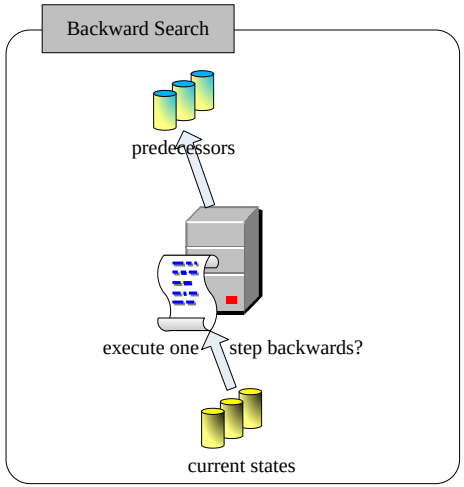
Limitation: accepts transition system as input, not realistic program

Our approach:

- perform backward reachability analysis on-the-fly
- operate directly on Boolean program instead of transition system, thus avoiding local state explosion

PreImage Computation





Challenges

- finding previous program state: need to "execute" program backwards
- creating threads in arbitrary local states

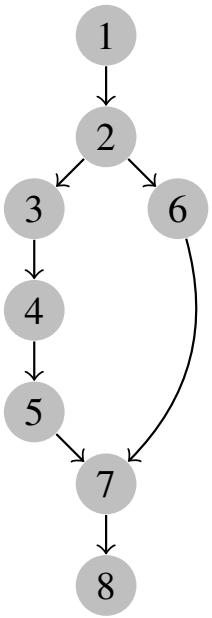
Solution

- Control Flow Graph + Weakest Precondition Propagation
- efficient iteration through candidate local states

An Example

Boolean Program

```
decl s := 1;      //shared
main(){
    decl l := 0; //local
 1: s := 0;
 2: goto 3, 6;
 3: assume(s);
 4: l := 1;
 5: goto 7;
 6: assume(!s);
 7: s:=!s;
 8: assert(!l);
}
```



On-the-fly Backward Exploration

$l_1 = 1 \wedge pc_1 = 8$

$l_1 = 1 \wedge pc_1 = 8 \wedge l_2 = 1 \wedge pc_2 = 7$

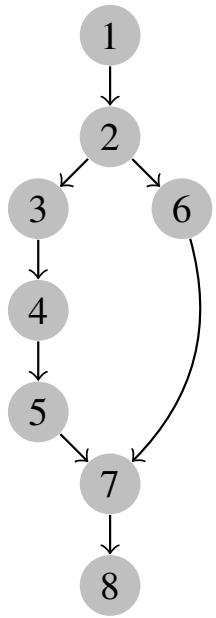
1. obtain possible predecessor program locations from CFG
2. obtain possible predecessor variable values via WP propagation

☞ $l_1 = 1 \wedge pc_1 = 8 \wedge l_2 = 1 \wedge pc_2 = 5$

An Example

Boolean Program

```
decl s := 1;      //shared
main(){
    decl l := 0; //local
 1: s := 0;
 2: goto 3, 6;
 3: assume(s);
 4: l := 1;
 5: goto 7;
 6: assume(!s);
 7: s:=!s;
 8: assert(!l);
}
```



On-the-fly Backward Exploration

$l_1 = 1 \wedge pc_1 = 8$

$l_1 = 1 \wedge pc_1 = 8 \wedge l_2 = 1 \wedge pc_2 = 7$

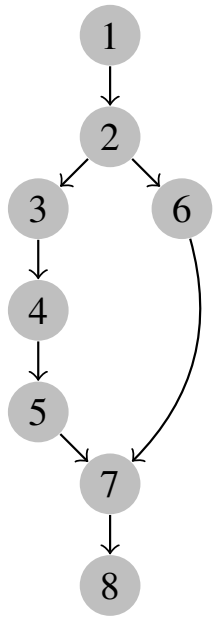
1. obtain possible predecessor program locations from CFG
2. obtain possible predecessor variable values via WP propagation

☞ $l_1 = 1 \wedge pc_1 = 8 \wedge l_2 = 1 \wedge pc_2 = 5$

An Example

Boolean Program

```
decl s := 1;      //shared
main(){
    decl l := 0; //local
 1: s := 0;
 2: goto 3, 6;
 3: assume(s);
 4: l := 1;
 5: goto 7;
 6: assume(!s);
 7: s:=!s;
 8: assert(!l);
}
```



On-the-fly Backward Exploration

$$
\begin{array}{llllllll}
 & l_1 = 1 & & & \wedge & & & pc_1 = 8 \\
 & l_1 = 1 & \wedge & pc_1 = 8 & \wedge & l_2 = 1 & \wedge & pc_2 = 7 \\
 & l_1 = 1 & \wedge & pc_1 = 8 & \wedge & l_2 = 1 & \wedge & pc_2 = 5 \\
 & & & pc_1 = 8 & \wedge & & & pc_2 = 4 \\
s = 1 & \wedge & & pc_1 = 8 & & & \wedge & pc_2 = 3 \\
s = 1 & \wedge & & pc_1 = 8 & & & \wedge & pc_2 = 2 \\
s = 0 & \wedge & & pc_1 = 7 & & & \wedge & pc_2 = 2 \\
 & & & pc_1 = 7 & \wedge & & & pc_2 = 1 \\
s = 0 & \wedge & & pc_1 = 6 & & & \wedge & pc_2 = 1 \\
s = 0 & \wedge & & pc_1 = 2 & & & \wedge & pc_2 = 1 \\
 & & & pc_1 = 1 & \wedge & & & pc_2 = 1
\end{array}
$$