Software Model Checking via Abstraction Refinement

Inzemamul Haque

March 14, 2016

[0] Material from lectures of Aditya Nori, Daniel Kroening, Thomas Ball and Sriram Rajamani

Outline

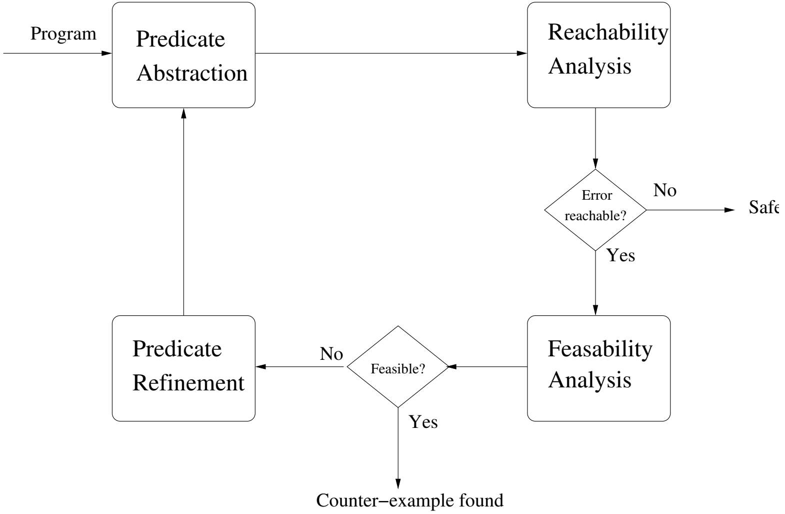
Motivation

- Model-checking is exhaustive exploration of state

Motivation

- Model-checking is exhaustive exploration of state
- Approach: Abstraction-refinement
  - **Construct an abstraction:** simple model of software having only variables and relationships important to the property to be checked
  - **Model check the abstraction:** it is easier because of smaller state space
  - **Refine the abstraction:** To reduce false errors as abstractions are over-approximations

Overview


Program
Predicate Abstraction
Reachability Analysis
Error reachable?
No
Safe
Yes
Feasability Analysis
Feasible?
No
Predicate Refinement
Yes
Counter–example found

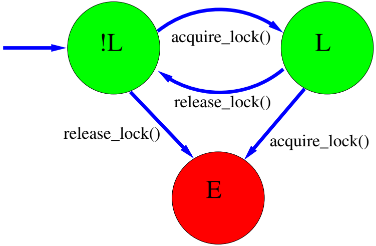
Example


```
1.do{
2.    acquire_lock();
3.    oldx = newx;
4.    if(*){
5.        release_lock();
6.        newx++;
    }
7.} while(newx != oldx);
8.release_lock();
```

Example

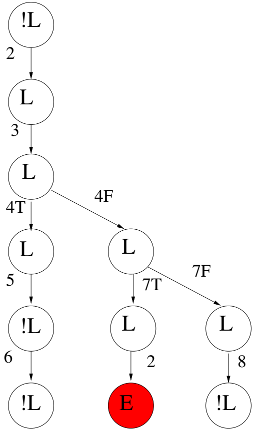
```
do{
    acquire_lock();
    oldx = newx;
    if(*){
        release_lock();
        newx++;
    }
} while(newx != oldx);
release_lock();
```

Does this code obey locking rule?



Example

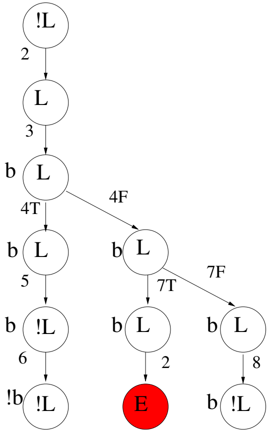
```
1.do{
2.     acquire_lock();
3.     oldx = newx;
4.     if(*){
5.          release_lock();
6.          newx++;
    }
7.} while(newx != oldx);
8.release_lock();
```



Example

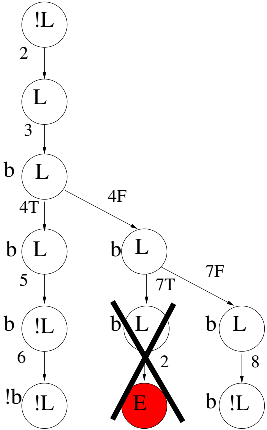
```
1.do{
2.     acquire_lock();
3.     oldx = newx;
4.     if(*){
5.          release_lock();
6.          newx++;
     }
7.} while(newx != oldx);
8.release_lock();
```

Adding b: (oldx==newx)



Example

```
1.do{
2.    acquire_lock();
3.    oldx = newx;
4.    if(*){
5.        release_lock();
6.        newx++;
   }
7.} while(newx != oldx);
8.release_lock();
```



Predicate Abstraction as Abstract Domain

- Given a set of predicates $p_1, p_2, \ldots, p_n$ over set of states S
- Abstract state is valuation of these predicates:

$$s_a = \mathbb{B}^n$$

- Abstraction function is:

$$\alpha(s) = \langle p_1(s), p_2(s), \ldots, p_n(s) \rangle$$

- Transfer functions are the strongest post-conditions

Predicate Abstraction

Different ways of Predicate Abstraction

- Abstract state graphs (Graf and Saidi)
- Boolean programs (SLAM)
- Abstract Reachability Tree (BLAST)

Boolean Program

A C-like program with only Boolean variables

```
1.do{
2.    acquire_lock();
3.    oldx = newx;
4.    if(*){
5.        release_lock();
6.        newx++;
    }
7.} while(newx != oldx);
8.release_lock();
```

```
do{
    locked=true;
    b=true;
    if(*){
        locked=false;
        b=b?false:*;
    }
} while(!b);
locked=false;
```

Reachability Analysis

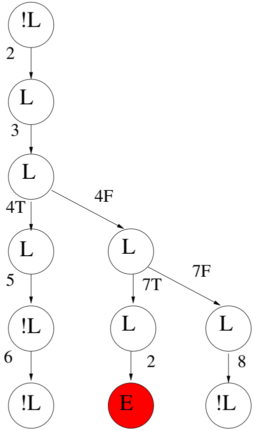
- We can use the idea of intersection of pushdown automata and DFA
- Pushdown automata is for call stack
- DFA is for the property we are going to check
- Other techniques like IDFS (RHS-95) can be used (to be discussed later)

Feasibility Analysis

- A path can be converted to corresponding path formula
- If formula is satisfiable $\implies$ path is feasible
- Else path is infeasible

Example

```
1.do{
2.     acquire_lock();
3.     oldx = newx;
4.     if(*){
5.          release_lock();
6.          newx++;
    }
7.} while(newx != oldx);
8.release_lock();
```



Refinement of Predicates

- Predicates are refined based on the counter-example found
- Deduce from the path formula which was unsatisfiable
- oldx=newx $\wedge$ newx!=oldx is unsatisfiable
- Add the predicate oldx=newx
- Perform all the steps again

BLAST

- **B**erkeley **L**azy **A**bstraction **S**oftware verification **T**ool
- Program is represented as a set of Control Flow Automata(CFA) for each function
- Abstract Reachability Tree (ART) is constructed for Model checking
- When ART is *complete*, BLAST terminates

Control Flow Automata(CFA)

- A CFA is a directed graph with
  - Vertices: program counter values
  - Edges: program operations
- Edges are labelled by the instruction
- An instruction can be
  - basic block of assignments
  - assume predicate
  - function call with call by value parameters
  - return instruction

Example

```
 Example() {
1:   if (*){
7:      do {
                got_lock = 0;
8:              if (*){
9:                 lock();
                   got_lock++;
                }
10:             if (got_lock){
11:                unlock();
                }
12:     } while (*)
     }
2:   do {
          lock();
          old = new;
3:        if (*){
4:           unlock();
             new++;
          }
5:   } while (new != old);
6:   unlock();
     return;
}
```

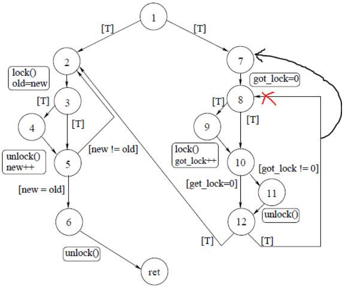
```
lock(){
   if (LOCK == 0){
      LOCK = 1;
   } else {
      ERROR
   }
}

unlock(){
   if (LOCK == 1){
      LOCK = 0;
   } else {
      ERROR
   }
}
```

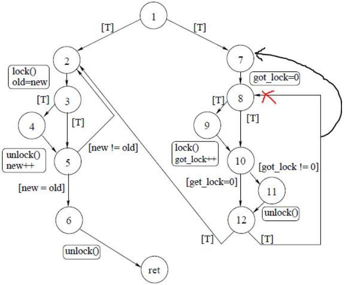
[0]Example taken from Henzinger et.al., Lazy Abstraction, POPL, 2002

Example
Control Flow Automaton


```
 Example() {
1:  if (*){
7:     do {
             got_lock = 0;
8:           if (*){
9:              lock();
                got_lock++;
             }
10:          if (got_lock){
11:             unlock();
             }
12:    } while (*)
    }
2:  do {
        lock();
        old = new;
3:      if (*){
4:          unlock();
            new++;
        }
5:  } while (new != old);
6:  unlock();
    return;
}
```

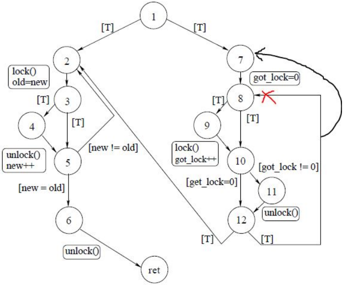


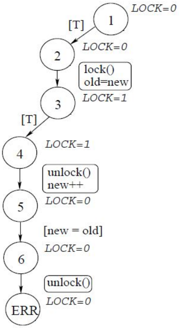
Example
Construction of Reachability Tree



Example

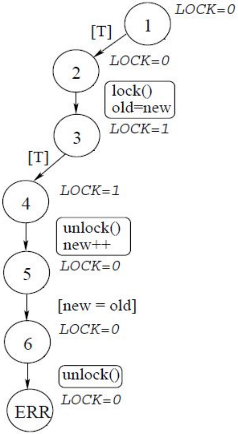
Construction of Reachability Tree





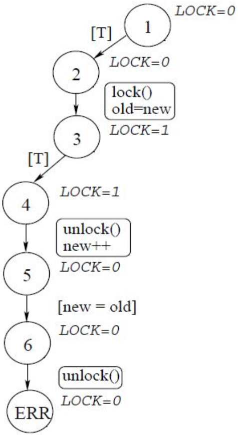
Example

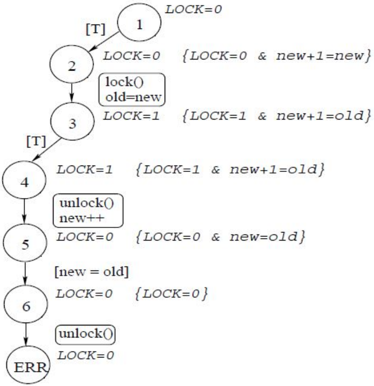
Construction of Reachability Tree



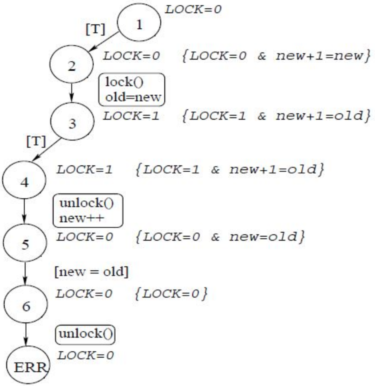
Example

Construction of Reachability Tree

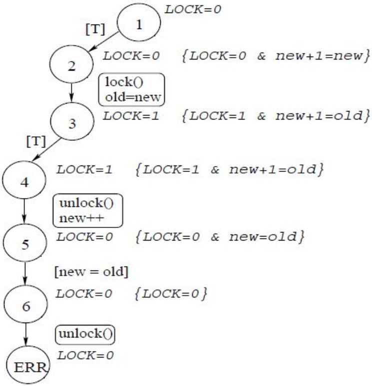


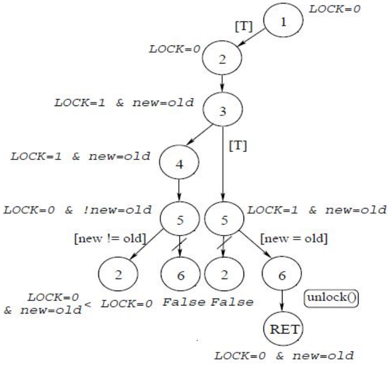


Example
Construction of Reachability Tree



Example
Construction of Reachability Tree





Craig's Interpolant

If $\phi_1 \wedge \phi_2 = \mathit{false}$ then there exists ψ such that

- $\phi_1 \implies \psi$
- $\psi \wedge \phi_2 = \mathit{false}$
- ψ contains variables which are common to both ϕ_1 and ϕ_2

ψ is called interpolant

Example

$$\phi_1 : x = 0 \wedge y > x$$
$$\phi_2 : y < z \wedge z < 0$$

Example

- $\phi_1 \wedge \phi_2 = \mathit{false}$

$\phi_1 : x = 0 \wedge y > x$

$\phi_2 : y < z \wedge z < 0$

Example

$\phi_1 : x = 0 \wedge y > x$
$\phi_2 : y < z \wedge z < 0$

- $\phi_1 \wedge \phi_2 = false$
- $y > 0$ is an interpolant
- $(x = 0 \wedge y > x) \implies y > 0$
- $y > 0 \wedge (y < z \wedge z < 0) = false$
- y is the only common variable in ϕ_1 and ϕ_2

Motivation

- A path can be broken in two parts, ϕ_1 and ϕ_2
- ψ_i represents a set of states somewhere in the path from where you cannot go on by taking ϕ_2.

Motivation for using Craig's Interpolation

1:	$x := ctr;$	$\langle x,1\rangle = \langle ctr,0\rangle$	$x = ctr$
2:	$ctr := ctr+1;$	$\langle ctr,1\rangle = \langle ctr,0\rangle + 1$	$x = ctr - 1$
3:	$y := ctr;$	$\langle y,2\rangle = \langle ctr,1\rangle$	$x = y - 1$
4:	$\texttt{assume}(x = m);$	$\langle x,1\rangle = \langle m,0\rangle$	$y = m + 1$
5:	$\texttt{assume}(y \neq m+1);$	$\langle y,2\rangle = \langle m,0\rangle + 1$	

Figure 2. Infeasible trace; constraints; predicates.

Proof of Unsatisfiability

Definition 1. *A* proof of unsatisfiability Π *for a set of clauses* C *is a directed acyclic graph* (V_Π, E_Π), *where* V_Π *is a set of clauses, such that*

- *for every vertex* $c \in V_\Pi$, *either*
  - $c \in C$, *and* c *is a root, or*
  - c *has exactly two predecessors,* c_1 *and* c_2, *such that* c *is the resolvent of* c_1 *and* c_2, *and*
- *the empty clause is the unique leaf.*

Craig's Interpolation

Definition 2. *Let (A, B) be a pair of clause sets and let Π be a proof of unsatisfiability of $A \cup B$, with leaf vertex* FALSE. *For all vertices $c \in V_\Pi$, let $p(c)$ be a boolean formula, such that*

- *if c is a root, then*
  - *if $c \in A$ then $p(c) = g(c)$,*
  - *else $p(c)$ is the constant* TRUE.
- *else, let c_1, c_2 be the predecessors of c and let v be their pivot variable:*
  - *if v is local to A, then $p(c) = p(c_1) \vee p(c_2)$,*
  - *else $p(c) = p(c_1) \wedge p(c_2)$.*

The Π-interpolant of (A, B), denoted $\text{ITP}(\Pi, A, B)$ *is* $p(\text{FALSE})$.