Software Model Checking via Abstraction Refinement

Inzemamul Haque

March 14, 2016

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

1. Overview
2. Predicate Abstraction
3. Reachability Analysis
4. Feasibility Analysis
5. Refinement of Predicates
6. BLAST
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

Feasibility Analysis

Predicates Refinement

Error reachable? → Yes → Safe

No → Counter-example found

Feasible? → Yes → Safe

No → Counter-example found

Reachability Analysis
Example

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

Example

Example

Does this code obey locking rule?
Example

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

Does this code obey locking rule?

![Diagram showing the locking rule with nodes L, !L, and E, and edges labeled acquire_lock() and release_lock()]

Overview
Predicate Abstraction
Reachability Analysis
Feasibility Analysis
Refinement of Predicates
BLAST
Example

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

1. do{
2.    acquire_lock();
3.    oldx = newx;
4.    if(*){
5.       release_lock();
6.       newx++;
7.    }
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:
  \[ s_a = B^n \]
- Abstraction function is:
  \[ \alpha(s) = \langle p_1(s), p_2(s), \ldots, p_n(s) \rangle \]
- Transfer functions are the strongest post-conditions
Different ways of Predicate Abstraction

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

9. do{
10.  locked=true;
11.  b=true;
12.  if(*){
13.    locked=false;
14.    b=b?false:*;
15.  } while(!b);
16.  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.    }
8.} while(newx != oldx);
9. release_lock();
Refinement of Predicates

- Predicates are refined based on the counter-example found
- Deduce from the path formula which was unsatisfiable
  \[ \text{oldx} = \text{newx} \land \text{newx} \neq \text{oldx} \] is unsatisfiable
- Add the predicate \text{oldx} = \text{newx}
- Perform all the steps again
BLAST

- Berkeley Lazy Abstraction Software verification Tool
- 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

```c
Example() {  
1:   if (*){  
7:     do {  
8:       if (*){  
9:         lock();  
10:        got_lock++;  

7:     } while (*)
12:   } while (*)

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

lock(){  
if (LOCK == 0){  
   LOCK = 1;
12: } else {
13:    ERROR
14: }
15: }

unlock(){  
if (LOCK == 1){  
   LOCK = 0;
12: } else {
13:    ERROR
14: }
15: }
```

Example taken from Henzinger et.al., Lazy Abstraction, POPL, 2002
Example
Control Flow Automaton

Example()
{ }
1: if (*){
7: do {
10: if (got_lock){
11: unlock();
}
12: } while (*)
}
2: do {
3: if (*){
4: unlock();
5: } while (new != old);
6: unlock();
7: return;
}
Example
Construction of Reachability Tree
Example
Construction of Reachability Tree
Example
Construction of Reachability Tree

Software Model Checking via Abstraction Refinement
Inzemamul Haque

Overview Predicate Abstraction Reachability Analysis Feasibility Analysis Refinement of Predicates BLAST

Example
Construction of Reachability Tree

Software Model Checking via Abstraction Refinement
Inzemamul Haque

Overview Predicate Abstraction Reachability Analysis Feasibility Analysis Refinement of Predicates BLAST

Example
Construction of Reachability Tree

Software Model Checking via Abstraction Refinement
Inzemamul Haque

Overview Predicate Abstraction Reachability Analysis Feasibility Analysis Refinement of Predicates BLAST

Example
Construction of Reachability Tree
Example
Construction of Reachability Tree

Software Model Checking via Abstraction Refinement
Inzemamul Haque

Overview
Predicate Abstraction
Reachability Analysis
Feasibility Analysis
Refinement of Predicates
BLAST
Example
Construction of Reachability Tree

Software Model Checking via Abstraction Refinement
Inzemamul Haque

Overview Predicate Abstraction Reachability Analysis Feasibility Analysis Refinement of Predicates BLAST

Example
Construction of Reachability Tree

Predicate Abstraction Reachability Analysis Feasibility Analysis Refinement of Predicates BLAST
Example
Construction of Reachability Tree

1. LOCK=0
2. LOCK=0 \{LOCK=0 & new+1=new\}
3. LOCK=1 \{LOCK=1 & new+1=old\}
4. LOCK=1 \{LOCK=1 & new+1=old\}
5. LOCK=0 \{LOCK=0 & new=old\}
6. LOCK=0 \{LOCK=0\}

BLAST

Example
Construction of Reachability Tree
Craig’s Interpolant

If $\phi_1 \land \phi_2 = false$ then there exists $\psi$ such that

- $\phi_1 \implies \psi$
- $\psi \land \phi_2 = false$
- $\psi$ contains variables which are common to both $\phi_1$ and $\phi_2$

$\psi$ is called interpolant
Example

\[ \phi_1 : x = 0 \land y > x \]
\[ \phi_2 : y < z \land z < 0 \]
Example

\[ \phi_1 \land \phi_2 = \text{false} \]

\[ \phi_1 : x = 0 \land y > x \]
\[ \phi_2 : y < z \land z < 0 \]
Example

\[ \phi_1 : x = 0 \land y > x \]
\[ \phi_2 : y < z \land z < 0 \]

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

- A path can be broken in two parts, $\phi_1$ and $\phi_2$
- $\psi_i$ represents a set of states somewhere in the path from where you cannot go on by taking $\phi_2$. 
Motivation for using Craig’s Interpolation

1:  \( x := \text{ctr}; \)  
2:  \( \text{ctr} := \text{ctr} + 1; \)  
3:  \( y := \text{ctr}; \)  
4:  \( \text{assume}(x = m); \)  
5:  \( \text{assume}(y \neq m + 1); \)

\( \langle x, 1 \rangle = \langle \text{ctr}, 0 \rangle \)  
\( \langle \text{ctr}, 1 \rangle = \langle \text{ctr}, 0 \rangle + 1 \)  
\( \langle y, 2 \rangle = \langle \text{ctr}, 1 \rangle \)  
\( \langle x, 1 \rangle = \langle m, 0 \rangle \)  
\( \langle y, 2 \rangle = \langle m, 0 \rangle + 1 \)

\( x = \text{ctr} \)  
\( x = \text{ctr} - 1 \)  
\( x = y - 1 \)  
\( y = m + 1 \)

**Figure 2. Infeasible trace; constraints; predicates.**
Proof of Unsatisfiability

**Definition 1.** A proof of unsatisfiability $\Pi$ for a set of clauses $C$ is a directed acyclic graph $(V_\Pi, E_\Pi)$, where $V_\Pi$ 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.
Definition 2. Let \((A, B)\) be a pair of clause sets and let \(\Pi\) 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 \(p(c) = p(c_1) \lor p(c_2)\),
  - else \(p(c) = p(c_1) \land p(c_2)\).

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