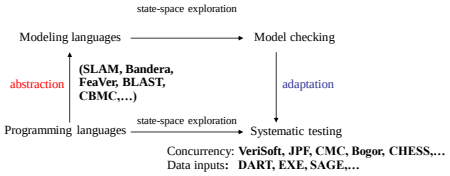
Lecture 3: Software Model Checking via Abstraction

Patrice Godefroid

Thanks: (most) slides for this lecture are borrowed from Ranjit Jhala and Rupak Majumdar

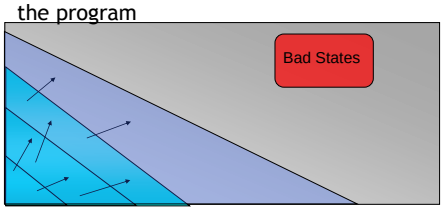
Software Model Checking

- How to apply model checking to analyze **software**?
- Two main approaches:



Model Checking Algorithm

- Graph Search
  - Linear time in the size of the graph
  - Exponential time (or worse) in the size of the program



Enumerative Model Checking

- For each state, compute successor states
- Implement classical graph algorithms
  - E.g., Depth-first or breadth-first search
  - Starting from initial states and searching forward for bad states
  - Or starting from bad states and searching backward for initial states

State Space Explosion

- Biggest problem is state space explosion
  - N bits $\Rightarrow 2^N$ states
- Techniques to fight state explosion:
  - Search on-the-fly
  - Partial order and symmetry reduction
  - Do not store dead variables
  - Etc.
- Many successful implementations
  - Spin, Murphi, VeriSoft, … [Protocol verification]

Symbolic Model Checking

- Idea: Work with sets of states, rather than individual states

Given: Transition graph G, target states σ^T
begin
- σ^R = set of Initial states
- repeat forever
  - if $\sigma^R \cap \sigma^T \neq \emptyset$ then return "yes" (error)
  - if $Post(\sigma^R) \subseteq \sigma^R$ then return "no" (safe)
  - $\sigma^R := \sigma^R \cup Post(\sigma^R)$

end

Here, $Post(\sigma) = \{s' \mid \exists s \in \sigma.\ s \rightarrow s'\}$

Encoding Sets through Formulas

- Idea: Represent sets of states symbolically, using constraints
- E.g., $1 \le x \le 100$ represents the 100 states x =1, x =2, …, x =100
- Represent both sets of initial states and transition relation implicitly

Representing States as *Formulas*

$[F]$ = set of states satisfying F $\{s \mid s \models F\}$	F = FOL formula over program variables
$[F_1] \cap [F_2]$	$F_1 \wedge F_2$
$[F_1] \cup [F_2]$	$F_1 \vee F_2$
$\overline{[F]}$	$\neg F$
$[F_1] \subseteq [F_2]$	$F_1 \Rightarrow F_2$

Symbolic Transition Graph

- A transition graph
  - A Formula Init(x) representing initial states
  - A Formula TR(x,x') representing the transition relation
- Example: C program

x:=e TR(x,x'): loc=pc∧loc'=pc'∧ x'=e ∧ {y'=y|y≠x}
Assume(p) TR(x,x'): loc=pc ∧ loc'=pc'∧ p

Symbolic Transition Graph

- Operations:
  - Post(X) = {s' | ∃s∈X. s → s'}
    = ∃s. X(s) ∧ TR(s,s')
  - Pre(X) = {s | ∃s'∈X. s → s'}
    = ∃s'. TR(s,s') ∧ X(s')
- Can implement using formula manipulations

Symbolic Model Checking

Given: Transition graph G, target states σ^T
begin
- σ^R = Formula representing set of Initial states
- repeat forever
  if $\sigma^R \wedge \sigma^T$ is satisfiable then return "yes" (error)
  if Post(σ^R) $\Rightarrow \sigma^R$ then return "no" (safe)
  $\sigma^R := \sigma^R \vee$ Post(σ^R)
  end

Here, Post(σ)(s') = ∃s. σ(s) ∧ TR(s,s')

Can be implemented using decision procedures for the language of formulas

Example: Mutual Exclusion

```
loop                                  ||   loop
  out:  x1 := 1; last := 1                   out:  x2 := 1; last := 2
  req:  await x2 = 0  or  last = 2           req:  await x1 = 0  or  last = 1
  in:   x1 := 0                              in:   x2 := 0
end loop.                                  end loop.
```

Symbolic representation has variables
pc1, pc2, x1, x2, last
Initial states:
pc1=out ∧ pc2=out ∧ x1=0 ∧ x2=0 No constraint on last
Transition relation:
pc1=out ∧ x1'=1∧ last'=1∧ pc2'=pc2 ∧ x2'=x2
∨ …

What about Software?

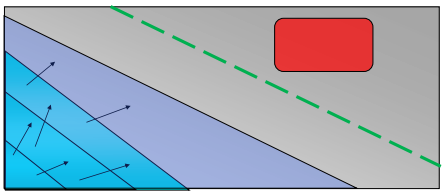
- Can construct an infinite state transition system from a program
- States: The state of the program
  - (stack, heap, pc location)
- Transitions: q → q' iff in the operational semantics, there is a transition of the program from q to q'
- Initial state: Initial state of the program

Termination

- Each operation can be computed
- But iterating Pre or Post operations may not terminate
- What do we do now?

Observation

- Often, we do not need the exact set of reachable states
  - We need a set of states that separates the reachable states from the bad states



Before we proceed

- What is the sign of the following product:

  - 12433454628 * 94329545771 ?

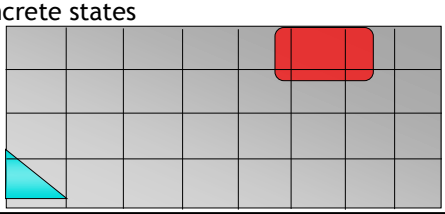
Idea

- One can "abstract" the behavior of the system, and yet reason about certain aspects of the program
- Abstraction:

  -ve * +ve = -ve

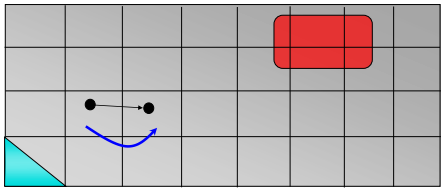
Abstract Interpretation

- The state transition graph is large/infinite
- Suppose we put a finite grid on top
- Abstract states are equivalence classes of concrete states



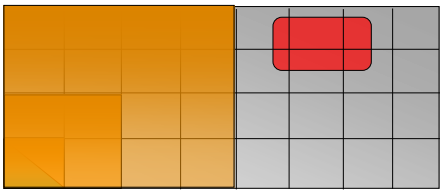
Conservative Abstraction

- Every concrete state s is abstracted by [s]
- Every time s → s', we put [s] → [s']
- This allows more behaviors ("may")



Abstract Model Checking

- Search the abstract graph until fixpoint
  - Can be much smaller than original graph
  - Can be finite, when original is infinite



Simulation Relations

- A relation ≤ ⊆ SxS is a simulation relation if s ≤ s' implies
  - Observation(s) = Observation(s')
  - For all t such that s → t
    there exists t' such that s' → t'
    and s' ≤ t'

Formally captures notion of "more behaviors"
Implies containment of reachable behaviors

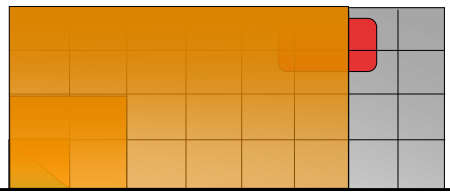
Main Theorem

- s ≤ [s] is a simulation relation
- If an error is unreachable in Abs(G) then it is unreachable in G
- Plan:

1. Find a suitable grid to make the graph finite state
2. Run the finite-state model checking algorithm on this abstract graph
3. If abstract graph is safe, say "safe" and stop

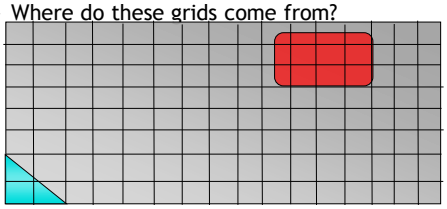
What if the Abstract Graph says Unsafe?

- The error may or may not be reachable in the actual system
  - Stop and say "Don't know"



What if the Abstract Graph says Unsafe?

- Or, put a finer grid on the state space
- And try again
  - Finer grid is more precise but more expensive
  - Where do these grids come from?

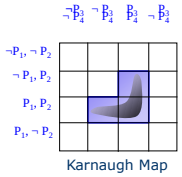


Grids: Predicate Abstraction

- Suppose we fix a set of facts about program variables
  - E.g., old = new, lock = 0, lock = 1
- Grid: Two states of the program are equivalent if they agree on the values of all predicates
  - N predicates = 2^N abstract states
- How do we compute the grid from the program?

Predicate Abstraction

Region Representation: formulas over predicates



Karnaugh Map

$P_1 : x = y$ $P_2 : z = t + y$

$P_3 : x \leq z+1$ $P_4 : *u = x$

Set of states

Abstract Set: $P_1P_2P_4 \vee \neg P_1 P_2 P_3 P_4$

Example

- I have predicates
  - lock=0, new=old
- My current region is lock=0 ∧ new=old
- Consider the assignment new = new+1
- What is abstract post?
  lock=0 ∧ ¬ (new=old)

Symbolic Search with Predicates

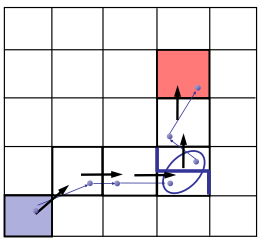
Symbolic representation:
Boolean formulas of (fixed set of) predicates

- Boolean operations: easy
- Emptiness check: Decision procedures
- Post: The abstract post computation algorithm
- Can now implement symbolic reachability search!
- (Similar with Pre instead of Post)

Big Question

- Who gives us these predicates?
- Answer 1: The user
  - Manual abstractions
    - Given a program and property, the user figures out what are the interesting predicates
  - Dataflow analysis
    - For "generic" properties, come up with a family of predicates that are likely to be sufficient for most programs

Answ2: Counterex.-Guided Refinement



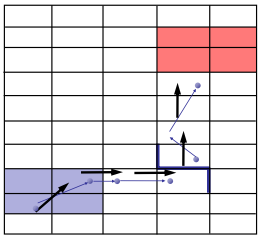
Solution

Use spurious **counterexamples** to **refine** abstraction

1. **Add predicates** to distinguish states across **cut**
2. Build **refined** abstraction

Imprecision due to **merge**

Iterative Abstraction-Refinement



[Kurshan et al 93] [Clarke et al 00]
[Ball-Rajamani 01]
Implemented in tools like SLAM and BLAST

Solution

Use spurious **counterexamples** to **refine** abstraction

1. Add predicates to distinguish states across **cut**
2. Build **refined** abstraction
   -eliminates counterexample
3. **Repeat** search
   Till real counterexample or system proved safe

Example:

Property3:
IRP Handler
Win NT DDK

Program	*Lines**	*Time (mins)*	*Predicates*	
			Total	*Average*
`kbfiltr`	12k	3	72	***6.5***
`floppy`	17k	25	240	***7.7***
`diskprf`	14k	13	140	***10***
`cdaudio`	18k	23	256	***7.8***
`parport`	61k	74	753	***8.1***
`parclss`	138k	77	382	***7.2***

localization

* Pre-processed