Lecture 13

15.1.19
Other solutions to the state-explosion problem

Small models replace the full, concrete model:

• Abstraction
• Compositional verification
• Partial order reduction
• Symmetry
Given an abstraction function $h : S \rightarrow S_h$, the concrete states are grouped and mapped into abstract states:
How to define an abstract model:

Given $M$ and $\varphi$, choose

- $S_h$ - a set of abstract states
- $AP$ - a set of atomic propositions that label concrete and abstract states
- $h : S \rightarrow S_h$ - a mapping from $S$ on $S_h$ that satisfies:

  $$h(s) = h(t) \text{ only if } L(s) = L(t)$$

- $h$ is called appropriate w.r.t. $AP$
The abstract model
\[ M_h = (S_h, I_h, R_h, L_h) \]

- \( s_h \in I_h \iff \exists s \in I : h(s) = s_h \)
- \( (s_h, t_h) \in R_h \iff \exists s, t \in I : h(s) = s_h \land h(t) = t_h \land (s, t) \in R \)
- \( L_h(s_h) = L(s) \) for some \( s \) where \( h(s) = s_h \)

This is an exact abstraction
An approximated abstraction (an approximation)

\[
\begin{align*}
\cdot \ s_h & \in I_h \iff \exists s \in I : h(s) = s_h \\
\cdot \ (s_h, t_h) & \in R_h \iff \\
& \exists s, t \ [ h(s) = s_h \land h(t) = t_h \land (s, t) \in R ] \\
\cdot \ L_h & \text{ is as before}
\end{align*}
\]

Notation:
\[ M_r - \text{reduced (exact)} \quad M_h - \text{approximated} \]
Predicate Abstraction

• Given a program over variables \( V \)
• **Predicate** \( P_i \) is a first-order atomic formula over \( V \)
  Examples: \( x+y < z^2 \), \( x=5 \)

• Choose: \( AP = \{ P_1, \ldots, P_k \} \) that includes
  - the atomic formulas in the property \( \varphi \) and
  - conditions in **if**, **while** statements of the program
Predicate Abstraction - Example

while ($x \leq 1$) {
    ......
    if ($y=2$) { .... }
    ......
}

$\varphi = \text{AFAG}(x>y)$

$\text{AP} = \{x>y, x \leq 1, y = 2\}$
Predicate Abstraction

• Labeling of concrete states:

\[ L(s) = \{ P_i \mid s \models P_i \} \]
Example (concrete model)

Program over natural variables $x, y$

$S = \mathbb{N} \times \mathbb{N}$

$AP = \{ P_1, P_2, P_3 \}$ where

$P_1 = x \leq 1$, $P_2 = x > y$, $P_3 = y = 2$

$AP = \{ x \leq 1, x > y, y = 2 \}$

$L((0,0)) = L((1,1)) = L((0,1)) = \{ P_1 \}$

$L((0,2)) = L((1,2)) = \{ P_1, P_3 \}$

$L((2,3)) = \emptyset$
Abstract model - Definition

- \( AP = \{ P_1, \ldots, P_k \} \)
- Abstract states are defined over Boolean variables \( \{ B_1, \ldots, B_k \} \):
  \( S_h \subseteq \{0,1\}^k \)

- \( h(s) = s_h \iff \) for all \( 1 \leq j \leq k : [ s \models P_j \iff s_h \models B_j ] \)

- \( L_h(s_h) = \{ P_j \mid s_h \models B_j \} \)

- Is \( h \) appropriate for \( AP \)?
Example (concrete model)

Program over natural variables $x, y$

$S = \mathbb{N} \times \mathbb{N}$

$AP = \{ P_1, P_2, P_3 \}$ where

- $P_1 = x \leq 1$
- $P_2 = x > y$
- $P_3 = y = 2$

$AP = \{ x \leq 1, x > y, y = 2 \}$

$L((0,0)) = L((1,1)) = L(0,1)) = \{ P_1 \}$

$L((0,2)) = L((1,2)) = \{ P_1, P_3 \}$

$L((2,3)) = \emptyset$
Example - (abstract model)

\[ \text{AP=} \{ P_1=(x \leq 1), P_2=(x>y), P_3=(y=2) \} \]

\[ S_h \subseteq \{ 0,1 \}^3 \]

\[ h((0,0)) = h((1,1)) = h(0,1)) = (1,0,0) \]
\[ h((0,2)) = h((1,2)) = (1,0,1) \]

No concrete state is mapped to \((1,1,1)\)

\[ L_h((1,0,0)) = \{ P_1 \} \]
\[ L_h((1,0,1)) = \{ P_1, P_3 \} \]

The concrete state and its abstract state are labeled identically
Computing $R_h$ (same example)

$$(s_h, t_h) \in R_h \iff \exists s, t \ [ h(s) = s_h \land h(t) = t_h \land (s, t) \in R ]$$
Computing $R_h$ (same example)

Program with one statement: $x := x+1$

$$\exists xyx'y' \ [ \begin{align*}
    P_1(x, y) & \iff b_1 \land \\
    P_2(x, y) & \iff b_2 \land \\
    P_3(x, y) & \iff b_3 \land \\
    x' = x+1 & \land y' = y \land \\
    \end{align*} \]$$

$h(s) = s_h \land h(t) = t_h$
Depending on $h$ and the size of $M$, $M_h$ (i.e. $I_h$, $R_h$) can be built using:

- **BDDs**, if $S$ is finite and not too big
- **SAT solver**, if $S$ is finite and possibly big
- **Theorem prover (SMT)**, $S$ might be infinite
Logic preservation Theorem

- **Theorem** If $\varphi$ is an ACTL/ACTL* specification over $AP$, then
  \[ M_h \models \varphi \Rightarrow M \models \varphi \]

- However, the reverse may not be valid.
Traffic Light Example

Property:
\[ \varphi = \text{AG AF } \neg \text{(state=red)} \]

Abstraction function \( h \) maps green, yellow to go.

M |= \varphi \iff M_h |= \varphi
If the abstract model invalidates a specification, the actual model may still satisfy the specification.

- Property: \( \phi = AG \ AF (state=red) \)
- \( M \models \phi \) but \( M_h \not\models \phi \)
- Spurious Counterexample: \( \langle red, go, go, \ldots \rangle \)
CounterExample-Guided Abstraction-Refinement (CEGAR)
The CEGAR Methodology

- **Generate initial abstraction**
- **Model check**
- **Refinement:**
  - Generate new abstraction
- **Check spurious counterexample**

- If $M$ and $\varphi$ is not spurious, then:
  - $M_h \models \varphi$
  - $M_h \models \varphi$
  - $T_h$ is not spurious
  - **Stop**

- If $M$ and $\varphi$ is spurious, then:
  - $M_h \not\models \varphi$
  - Generate counterexample $T_h$
  - $T_h$ is spurious
  - **Refinement:**
    - Generate new abstraction
  - **Model check**
  - **Check spurious counterexample**
Model Check the Abstract Model

Given the abstract model $M_h$

- If $M_h \not\models \varphi$, then the model checker generates a counterexample trace ($T_h$)
- Most current model checkers generate paths or loops
- Question: is $T_h$ spurious?
Counterexamples

- For $AGp$ it is a path to a state satisfying $\neg p$
- For $AFp$ it is an infinite path represented by a path+loop, where all states satisfy $\neg p$

On the other hand

- For $EFp$ we need to return the whole computation tree (the whole model)
- For $AX(AGp \lor AGq)$ we need to return a computation tree demonstrating $EX(EF\neg p \land EF\neg q)$
**Path Counterexample**

Assume that we have four abstract states

\[
\{1,2,3\} \leftrightarrow \alpha \quad \{4,5,6\} \leftrightarrow \beta \\
\{7,8,9\} \leftrightarrow \gamma \quad \{10,11,12\} \leftrightarrow \delta
\]

Abstract counterexample \( T_h = \langle \alpha, \beta, \gamma, \delta \rangle \)

\( T_h \) is not spurious, therefore, \( M \models \varphi \)
Spurious Path Counterexample

The concrete states mapped to the failure state are partitioned into 3 sets

<table>
<thead>
<tr>
<th>states</th>
<th>dead-end</th>
<th>bad</th>
<th>irrelevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>reachable</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>out edges</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

$T_h$ is spurious
Refining The Abstraction

- **Goal**: refine $h$ so that the dead-end states and bad states do not belong to the same abstract state.

- For this example, two possible solutions.
Refining the abstraction

• Refinement separates dead-end states from bad states, thus, eliminating the spurious transition from $S_{i-1}$ to $S_i$

• This can be done, for instance, by adding a new predicate to the abstract model and building a new, refined abstract model
Completeness of CEGAR

If $M$ is finite

- Our methodology refines the abstraction until either the property is proved or a real counterexample is found

- **Theorem** Given a finite model $M$ and an ACTL* specification $\phi$ whose counterexample is either path or loop, our algorithm will find a model $M_a$ such that

$$M_a \models \phi \iff M \models \phi$$
Conclusion on Abstraction-Refinement

We presented a framework for Counterexample Guided Abstraction Refinement (CEGAR) that

- Automatically constructs an initial abstraction, based on the checked property and the system

- If the abstract system contains a spurious counterexample then the abstraction is automatically refined in order to eliminate the counterexample
In this course

- Program verification (Floyd)
  - Path condition, path transformation
  - invariants
- Temporal logic
- Model checking
  - Explicit state algorithm
- State explosion problem
  - BDD-based model checking
  - SAT-based Bounded model checking
- (bi)simulations, Abstraction
Advanced topics

• 3-valued abstraction - returns T, F, ?
• Compositional verification
  - learning-based automatic construction of assumption on environment
• LTL model checking
• Finite automata on infinite words
  - Model checking with automata
• Full verification with SAT
• Finding security vulnerabilities with model checking
More advanced topics

• Automatic program repair
• Identifying program differencing
• Synthesis of programs from specification
  - Correct by construction