Model-Checking Frameworks: Outline

**Theory (Part 1)**
- Notion of Abstraction
- Aside: over- and under-approximation, simulation, bisimulation
- Counter-example-based abstraction refinement

**Abstraction and abstraction refinement in program analysis (Part 2)**
- Kinds of abstraction:
  - Data, predicate
- Building abstractions
  - Aside: weakest precondition
- Counter-example-based abstraction refinement

Outline, cont’d

**3-valued abstraction and abstraction-refinement (Part 3)**
- 3-valued logic
- Theory of 3-valued abstractions: combining over- and under-approximation
- 3-valued model-checking
- Building 3-valued abstractions
- Counter-example-based abstraction refinement
Acknowledgements

The following materials have been used in the preparation of this lecture:

Edmund Clarke
- “SAT-based abstraction/refinement in model-checking”, a course lecture at CMU

Corina Pasareanu
- Conference presentations at TACAS’01 and ICSE’01

John Hatcliff
- Course materials from Specification and Verification in Reactive Systems

Many thanks for providing this material!

Model Checking

Given a:
- Finite transition system $M(S, s_0, R, L)$
- A temporal property $\phi$

The model checking problem:
- Does $M$ satisfy $\phi$?

\[ M \models \phi \]
Model Checking (safety)

Add reachable states until reaching a fixed-point

= bad state

Too many states to handle!

= bad state
Abstraction

Abstraction Function: A Simple Example

Partition variables into visible(\(V\)) and invisible(\(I\)) variables.

The abstract model consists of \(V\) variables. \(I\) variables are made inputs.

The abstraction function maps each state to its projection over \(V\).
Abstraction Function: Example

\[
\begin{array}{cccc}
  x1 & x2 & x3 & x4 \\
  0 & 0 & 0 & 0 \\
  0 & 0 & 0 & 1 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 1 & 1 \\
\end{array}
\]

Group concrete states with identical visible part to a single abstract state.

Computing Abstractions

- **S** - concrete state space
- **S'** - abstract state space
- \(\alpha: S \rightarrow S'\) - abstraction function
- \(\gamma: S' \rightarrow S\) - concretization function
- Properties of \(\alpha\) and \(\gamma\):
  - \(\alpha(\gamma(A)) = A\), for \(A\) in \(S'\)
  - \(\gamma(\alpha(C)) \supseteq C\), for \(C\) in \(S\)
- The above properties mean that \(\alpha\) and \(\gamma\) are Galois-connected
Aside: simulations

\[ M = (s_0, S, R, L) \]
\[ M' = (t_0, S', R', L') \]

Definition: \( p \) is a simulation between \( M \) and \( M' \) if

1. \( (s_0, t_0) \in p \)
2. \( \forall (t, t_1) \in R' \exists (s, s_1) \in R \text{ s.t. } (s, t) \in p \) and \( (s_1, t_1) \in p \)

Intuitively, every transition in \( M' \) corresponds to some transition in \( M \)

Aside: bisimulation

\[ M = (s_0, S, R, L) \]
\[ M' = (t_0, S', R', L') \]

Definition: \( p \) is a bisimulation between \( M \) and \( M' \) if

1. \( p \) is a simulation between \( M \) and \( M' \) and
2. \( p \) is a simulation between \( M' \) and \( M \)
Computing Existential Transition Relation

\[ R^\exists [\text{Dams'97}]: (t, t_1) \in R' \text{ iff } \exists s \in \gamma(t) \text{ s.t. } \exists s_1 \in \gamma(t_1) \text{ and } (s, s_1) \in R \]

This ensures that \( M' \) is the over-approximation if \( M \), or \( M' \) simulates \( M \).

Abstract Kripke Structure

Abstract interpretation of atomic propositions

- \( I'(a, p) = \text{true} \) iff forall \( s \) in \( \gamma(a) \), \( I(s, p) = \text{true} \)
- \( I'(a, p) = \text{false} \) iff forall \( s \) in \( \gamma(a) \), \( I(s, p) = \text{false} \)

Abstract Transition Relation (2 choices)

- Over-Approximation (Existential)
  - Make a transition from an abstract state if at least one corresponding concrete state has the transition.
- Under-Approximation (Universal)
  - Make a transition from an abstract state if all the corresponding concrete states have the transition.
Existential Abstraction (Over-Approximation)

Let $\varphi$ be a universal temporal formula (ACTL, LTL)

Let $K'$ be an over-approximating abstraction of $K$

Preservation Theorem
- $K' \models \varphi$ implies $K \models \varphi$

Converse does not hold
- $K' \nvdash \varphi$ does not imply $K \nvdash \varphi$ !!!
Computing Transition Relation

\[ R^\forall^\exists \text{[Dams'97]}: (t, t_1) \in R' \text{ iff } \forall s \in \gamma(t) \exists s_1 \in \gamma(t') \text{ and } (s, s_1) \in R \]

This ensures that $M'$ is the under-approximation if $M$, or $M$ simulates $M'$.
Preservation via Under-Approximation

Let $\varphi$ be an existential temporal formula (ECTL)

Let $K'$ be an under-approximating abstraction of $K$

**Preservation Theorem**

- $K' \models \varphi$ implies $K \models \varphi$

**Converse does not hold**

- $K' \not\models \varphi$ does not imply $K \not\models \varphi$ !!!

Which abstraction to use?

<table>
<thead>
<tr>
<th>Property Type</th>
<th>Expected Result</th>
<th>Abstraction to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Universal (ACTL, LTL)</td>
<td>True</td>
<td>Over-</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>Under-</td>
</tr>
<tr>
<td>Existential (ECTL)</td>
<td>True</td>
<td>Under-</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>Over-</td>
</tr>
</tbody>
</table>

But what about mixed properties?!
Our specific problem

- Let $\varphi$ be a universally-quantified property (i.e., expressed in LTL or ACTL) and $M'$ simulates $M$.

- Preservation Theorem
  
  
  $M' \models \varphi \rightarrow M \models \varphi$

- Converse does not hold
  
  $M' \not\models \varphi \not\rightarrow M \not\models \varphi$

- The counterexample may be spurious

Checking the Counterexample

- Counterexample : $(c_1, ..., c_m)$
  - Each $c_i$ is an assignment to $V$.

- Simulate the counterexample on the concrete model.
Checking the Counterexample

Concrete traces corresponding to the counterexample:

\[ \phi = I(s_1) \land (\text{Initial State } \leftarrow s_0 \text{ in our case}) \]
\[ \bigwedge_{i=1}^{m-1} R(s_i, s_{i+1}) \land (\text{Unrolled Transition Relation}) \]
\[ \bigwedge_{i=1}^m \text{visible}(s_i) = c_i \quad (\text{Restriction of } \forall \text{ to Counterexample}) \]

Abstraction-Refinement Loop

\[ (M, \phi, \alpha) \xrightarrow{\text{Abstract}} (M', \phi) \xrightarrow{\text{Model Check}} \begin{cases} \text{Pass} & \text{No Bug} \\ \text{Fail} & \begin{cases} \text{Spurious} & \text{Refine} \\ \text{Real} & \text{Check Counterexample} \end{cases} \end{cases} \]
Refinement methods...

Localization
(R. Kurshan, 80’s)

Abstraction/refinement with conflict analysis
(Chauhan, Clarke, Kukula, Sapra, Veith, Wang, FMCAD 2002)

- Simulate counterexample on concrete model with SAT
- If the instance is unsatisfiable, analyze conflict
- Make visible one of the variables in the clauses that lead to the conflict
Why spurious counterexample?

Problem: Deadend and Bad States are in the same abstract state.

Solution: Refine abstraction function.

The sets of Deadend and Bad states should be separated into different abstract states.

Refinement
Refinement

\[ \alpha' \rightarrow \alpha' \rightarrow \alpha' \rightarrow \alpha' \rightarrow \alpha' \rightarrow \alpha' \]

Refinement: \( \alpha' \)

\[ \phi_D = I(s_1) \land \bigwedge_{i=1}^{f-1} R(s_i, s_{i+1}) \land \bigwedge_{i=1}^{f} \text{visible}(s_i) = c_i \]
Refinement

\[ \phi_B = R(s_f, s_{f+1}) \land \]
\[ \text{visible}(s_f) = c_f \land \text{visible}(s_{f+1}) = c_{f+1} \]

Refinement as Separation

<table>
<thead>
<tr>
<th></th>
<th>d₁</th>
<th>b₁</th>
<th>b₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Refinement: Find subset \( \mathcal{U} \) of \( \mathcal{I} \) that separates between all pairs of deadend and bad states. Make them visible.

Keep \( \mathcal{U} \) small!
Refinement as Separation

Refinement: Find subset $\mathcal{U}$ of $\mathcal{I}$ that separates between all pairs of deadend and bad states. Make them visible.

Keep $\mathcal{U}$ small!

The state separation problem
Input: Sets $D, B$
Output: Minimal $\mathcal{U} \in \mathcal{I}$ s.t.:
\[
\forall d \in D, \forall b \in B, \exists u \in \mathcal{U}. \quad d(u) \neq b(u)
\]

The refinement $\alpha'$ is obtained by adding $\mathcal{U}$ to $\mathcal{V}$. 

Refinement as Separation
Two separation methods

- **ILP-based separation**
  - Minimal separating set.
  - Computationally expensive.

- **Decision Tree Learning based separation.**
  - Not optimal.
  - Polynomial.

We will not talk about these in class