Introduction to Software Verification

Orna Grumberg

Lectures Material
winter 2017-18
Lecture 12

9.1.18
Summary

• Explicit model checking

State explosion problem

• BDD-based symbolic model checking
• SAT-based Bounded Model Checking (BMC)
Other solutions to the state-explosion problem

Small models replace the full, concrete model:

- Abstraction
- Compositional verification
- Partial order reduction
- Symmetry
example

Let $M$ be a communication system in which there are exactly 20 wait steps between a send and an ack.

$M::$

```
[send] → [wait] → [wait] → [wait] → [wait] → [wait] → [ack]
```

$s_0$ $s_1$ $s_2$ $s_{19}$ $s_{20}$ $s_{21}$

$M'$ includes all behaviors of $M$ and more:

$M'::$

```
[send] → [wait] → [ack]
```

$s'_0$
example

Every path in $M$ has a "representative path" in $M'$. Therefore, if we prove:

\[ M', s_0' \models A(\neg \text{ack } W \text{ send}) \]

We can conclude that also:

\[ M, s_0 \models A(\neg \text{ack } W \text{ send}) \]
Since $M'$ has more paths, if
$M',s'_0 \not\models AG (send \rightarrow F\ ack )$
then we cannot conclude that
$M,s_0 \not\models AG (send \rightarrow F\ ack )$

• A counterexample might be spurious
• Refinement might be needed
Equivalences and preorders

Goal: to define

- **Preorder** between models: $M_2 \geq M_1$ s.t.
  \[ M_2 \models \varphi \implies M_1 \models \varphi \]

- **Equivalence** between models: $M_1 \equiv M_2$ s.t.
  \[ M_1 \models \varphi \iff M_2 \models \varphi \]

Which properties are preserved?
We define:

**equivalence** between models that **strongly preserves** CTL*:

- If $M_1 \equiv M_2$ then for every CTL* formula $\varphi$,
  
  $M_1 |\varphi \iff M_2 |\varphi$

**preorder** between models that **weakly preserves** ACTL*:

- If $M_2 \geq M_1$ then for every ACTL* formula $\varphi$,
  
  $M_2 |\varphi \Rightarrow M_1 |\varphi$
**ACTL / ACTL**

- **No existential path quantifier (no E)**
  - Only A
- **Negation is applied to atomic propositions only**
- **Need \( \lor \) and \( \land \)**
- **U and the dual of U, V (release)**

\[
M, \pi \models (f_1 \lor f_2) \iff \forall j \geq 0 [ [\forall i < j. \pi^i \not\models f_1 ] \Rightarrow \pi^j \models f_2 ]
\]

- \( f_1 \lor f_2 \equiv \neg (\neg f_1 \lor \neg f_2) \)
ACTL

Universal CTL

• $p, \neg p$, for $p \in AP$

• $g_1 \lor g_2, g_1 \land g_2$

• $AX g_1, A (g_1 U g_2), A( g_1 V g_2)$
  
  – $AG g_1, AF g_1$ (can be expressed by $AU, AV$)

where $g_1, g_2$ are ACTL (state formulas)

Example: $AG AF$ restart is an ACTL formula
The simulation preorder [Milner]

Given two models over AP:
\[ M_1 = (S_1, I_1, R_1, L_1), \quad M_2 = (S_2, I_2, R_2, L_2) \]

\( H \subseteq S_1 \times S_2 \) is a simulation iff
for every \((s_1, s_2) \in H\):
- \( s_1 \) and \( s_2 \) satisfy the same propositions
- For every successor \( t_1 \) of \( s_1 \) there is a successor \( t_2 \) of \( s_2 \) such that \((t_1, t_2) \in H\)

Notation:
\( s_1 \leq s_2 \) if there is simulation \( H \), s.t. \((s_1, s_2) \in H\)
The simulation preorder [Milner]

Given two models over AP:

\[ M_1 = (S_1, I_1, R_1, L_1), \quad M_2 = (S_2, I_2, R_2, L_2) \]

\( H \subseteq S_1 \times S_2 \) is a simulation iff

for every \((s_1, s_2) \in H\):

- \( L_1(s_1) = L_2(s_2) \)

- \( \forall t_1 \ [ (s_1, t_1) \in R_1 \Rightarrow \exists t_2 \ [ (s_2, t_2) \in R_2 \land (t_1, t_2) \in H ] ] \)

Notation: \( s_1 \preceq s_2 \)
Simulation preorder (cont.)

\( H \subseteq S_1 \times S_2 \) is a simulation from \( M_1 \) to \( M_2 \) iff

\( H \) is a simulation and

for every \( s_1 \in I_1 \) there is \( s_2 \in I_2 \)

s.t. \( (s_1, s_2) \in H \)

Notation: \( M_1 \leq M_2 \)
Bisimulation relation [Park]

For models $M_1$ and $M_2$ over $AP$, 
$B \subseteq S_1 \times S_2$ is a bisimulation 
iff for every $(s_1, s_2) \in B$:

- $L_1(s_1) = L_2(s_2)$
- $\forall t_1 [(s_1, t_1) \in R_1 \Rightarrow \exists t_2 [(s_2, t_2) \in R_2 \land (t_1, t_2) \in B]]$
- $\forall t_2 [(s_2, t_2) \in R_2 \Rightarrow \exists t_1 [(s_1, t_1) \in R_1 \land (t_1, t_2) \in B]]$

Notation: $s_1 \equiv s_2$
Bisimulation relation (cont.)

B ⊆ S₁ x S₂ is a
Bisimulation between M₁ and M₂ iff

• B is a bisimulation, and
• for every s₁ ∈ I₁ there is s₂ ∈ I₂
  s.t. (s₁, s₂) ∈ B and
• for every s₂ ∈ I₂ there is s₁ ∈ I₁
  s.t. (s₁, s₂) ∈ B

Notation: M₁ ≡ M₂
Bisimulation equivalence $M_1 \equiv M_2$

$B = \{ (1, 1'), (2, 4'), (4, 2'), (3, 5'), (3, 6'), (5, 3'), (6, 3') \}$
Simulation preorder

\( M_1 \preceq M_2 \)
\[ M_1 \leq M_2 \]
$\mathcal{M}_1 \geq \mathcal{M}_2$
\[ M_1 \leq M_2 \text{ and } M_1 \geq M_2 \text{ but not } M_1 \equiv M_2 \]

since they do not agree on all CTL.

Example: \( M_2 \models \text{EX AX } c \) \( M_1 \not\models \text{EX AX } c \)
(bi)simulation and logic preservation

Theorem:
If $M_1 \equiv M_2$ then for every $CTL^*$ formula $\phi$, $M_1 \models \phi \iff M_2 \models \phi$

If $M_2 \geq M_1$ then for every $ACTL^*$ formula $\phi$, $M_2 \models \phi \Rightarrow M_1 \models \phi$
Lemma:
If $B(s, s')$ then

- for every path $\pi = s_0, s_1, \ldots$ from $s$ there is a path $\pi' = s'_0, s'_1, \ldots$ from $s'$ such that for every $i$: $B(s_i, s'_i)$

- for every path $\pi' = s'_0, s'_1, \ldots$ from $s'$ there is a path $\pi = s_0, s_1, \ldots$ from $s$ such that for every $i$: $B(s_i, s'_i)$

We say that $\pi$ and $\pi'$ correspond and write $B(\pi, \pi')$
Proof:
Assume $B(s,s')$ and let $\pi = s_0, s_1, \ldots$ be a path from $s$.
We construct $\pi' = s'_0, s'_1, \ldots$ from $s'$ by induction on the location $i$ on $\pi'$.

Base:
We choose $s'_0$ to be $s'$. Therefore $B(s_0,s'_0)$.

Inductive step:
Assume $B(s_i,s'_i)$. $R(s_i, s_{i+1})$ since they are consecutive on $\pi$.
Therefore, there is $t'$ such that $R(s'_i, t')$ and $B(s_{i+1},t')$.
We choose $s'_{i+1}$ to be $t'$.
The proof that for every $\pi'$ there is a corresponding $\pi$ is similar.
Proof (continued):

Note: induction can prove a property only for a finite (possibly unbounded) set. Not for infinite sets.

Here: $\pi$ is infinite.

We proved that for every prefix of $\pi$ there is a corresponding prefix of $\pi'$
Proof (continued):

Assume there is no path starting from $s'$ that corresponds to $\pi$. Then for every path from $s'$ there is an $i$ such that $B(s_i, s'_i)$ does not hold. But this contradicts the previous proof which shows that $\pi'$ we constructed has $B(s_j, s'_j)$ for every $j$. 
Theorem:
Let $B(s, s')$. Then for every $\text{CTL}^*$ formula $f$, $s \models f \iff s' \models f$

Proof:
We show a simpler proof for $\text{CTL}$. By induction of the structure of the formula.

Base:
• $f \in \text{AP}$

Step:
• $f = \neg f_1$
• $f = f_1 \lor f_2$
• $f = \text{EX } f_1$
• $f = E (f_1 \cup f_2)$
• $f = EG f_1$
Abstractions

• They are one of the most useful ways to **fight** the state explosion problem

• They should **preserve properties of interest**: properties that hold for the abstract model should hold for the concrete model

• Abstractions should be **constructed directly from the program**
Abstraction

- Removes or simplifies details
- Removes entire components

that are irrelevant to the property under consideration, thus reducing the model size (number of states and transitions)
• Manual abstraction requires great creativity

• Goal:
  Automatically construct an abstract model that will preserve the required property
Use

• In model checking, a small abstract model $M_A$ will replace the full, concrete model $M$

• The abstract model $M_A$ has
  - less states and transitions
  - More behaviors

• $M_A$ is an over-approximation of $M$

• $M_A$ preserves ACTL / ACTL* properties
  - If $M_A \models f$ then $M \models f$
Outline for abstraction

• **Define** an abstract model that preserves the checked property

• **Consider different types** of abstractions

• **Automatically construct** an abstract model
  - Different constructions for different types

• **Automatically refine** it, if the abstraction is not detailed enough
We first define an abstract model $M_h$ based on a concrete (full) model $M$ of the system.

Goal: constructing $M_h$ directly from the program text.
Abstraction preserving ACTL/ACTL*

We use **Existential Abstraction** in which the abstract model is an **over-approximation** of the concrete model:

- The abstract model has **more behaviors**
- But no concrete behavior is lost

• Every **ACTL/ACTL**\(^*\) property true in the abstract model is also true in the concrete model
**ACTL***

- **ACTL*** is a subset of **CTL*** where
  - only *universal* path quantifiers are used
  - negation is restricted to atomic formulas

**AG AF restart** is an **ACTL*** formula
Given an abstraction function $h : S \rightarrow S_h$, the concrete states are grouped and mapped into abstract states: