Behavior Models and Verification
Lecture 10
Jan Kofroň, František Plášil
Today

• Infinite families of finite-state systems
• Bounded model checking
  □ HW verification
Infinite / Bounded Model Checking

Model

Property specification

AG(start → AF heat)

Model checker

Property satisfied

Property violated

Error report

Jan Kofroň, František Plášil, Lecture 10
Infinite families of finite states systems
Infinite families

- Finite model checking is fine
  - we know that generally model checking infinite state spaces is undecidable

- **But:** Protocols and circuits specification can be parameterized, e.g.:
  - size of int in multiplication unit of CPU
  - number of processors connected to bus
  - ...

Indeed, it would be handy to reason about such parameterized designs (models)

Formally – infinite family of systems:

$$\mathcal{F} = \{M_i\}_{i=1}^{\infty}$$

For temporal formula $f$ verify that:

$$\forall i: M_i \models f$$

This is generally still undecidable, though
Having indexed specification, indexed formulae would be fine as well.

**Indexed CTL (ICTL)**
- Formula indexed by integer
- \(i\)-th formula applies to \(i\)-th component

ICTL allows for expressing: \(\land_i f(i)\) and \(\lor_i f(i)\)
- and also: \(\land_{j \neq i} f(j)\) and \(\lor_{j \neq i} f(j)\)
Recall: Token ring example

- Simple token ring algorithm
  - n – non-critical section, t – keeping token,
    c – critical section, r – receive token, s – send token
- One process Q and several \( P_i \) of this type
  - Q initially in t, \( P_i \) initially in n
Recall: Token ring example

- Synchronous $Q \parallel P$ composition, natural synchronizing on $s$ and $r$ resulting in $\tau$
Token ring family: $\mathcal{F} = \{Q \parallel P^i\}_{i=1}^{\infty}$

Desired property: $\bigwedge_i \text{AG}(c_i \Rightarrow \bigwedge_{j \neq i} \neg c_j)$

i.e., if process $i$ is in critical section, then no other process is
Invariants

- Let $\mathcal{F} = \{M_i\}_{i=1}^{\infty}$ be a family of structures.
- Let $\geq$ be a reflexive, transitive relation on structures.
- **Invariant $I$** is a structure such that
  \[ \forall i: I \geq M_i \]

- Properties that can be checked are determined by $\geq$:
  - bisimulation (strong preservation: $I \models f \iff M \models f$)
  - simulation preorder (weak preservation: $I \models f \Rightarrow M \models f$)
  - language equivalence (strong preservation)
  - language preorder (weak preservation)
Token rings of size $n$ and of size 2 are in simulation preorder.

So for any CTL property $f$ it is sufficient to verify:

$$(P \parallel Q) \models f$$
Lemma: Let $\geq$ be a reflexive, transitive relation and let $\parallel$ be a composition operator that is monotonic w.r.t. $\geq$. If $I \geq P$ and $I \geq I \parallel P$, then $\forall i: I \geq P^i$, where $F = \{P^i\}_{i=1}^{\infty}$.

$\parallel$ is monotonic w.r.t. $\geq \iff \forall P_1, P_1', P_2, P_2': P_1 \geq P_1' \land P_2 \geq P_2' \Rightarrow P_1 \parallel P_2 \geq P_1' \parallel P_2'$
More systematic approach

- This is more like “This holds once we have the bisimulation/simulation” than “How to find the relation”

- Finding a suitable relation is hard, not possible in automatic way
  - recall: the problem is undecidable in general
Bounded model checking
Bounded model checking

- Let $M = \{S, I, R, L\}$ be Kripke structure
- Define predicate $\text{Reach}(s, s')$ iff $R(s, s')$
- Define $\lbrack M \rbrack^k = \bigwedge_{i=0}^{k-1} \text{Reach}(s_i, s_{i+1})$
- $\lbrack M \rbrack^k$ contains states reachable in $k$ steps
- Idea: Look for counterexamples made of $k$ states
Bounded model checking

\[ M, \neg \varphi \]

\[ k = 0 \]

\[ \neg \varphi \text{ satisfiable in } [M]^k \]

\[ inc(k) \]

\[ k < \text{threshold} \]

\[ M \not\models \neg \varphi \]

\[ M \not\equiv_k \neg \varphi \]
Bounded model checking

\[ k=5 \]
Bounded model checking for programs

- Mean: Construction of formula describing the transitions in the program
  - and trying to reach assertion violation, i.e., violation of $\text{AG } p$
  - checking for satisifiability of the formula
    - using SAT solver
SAT solvers

• Tools taking logical formula and deciding whether it is satisfiable
  ▶ whether there is satisfying assignment of free variables
  ▶ formula in conjunctive normal form (CNF)
  ▶ can contain quantifiers $\rightarrow$ harder problem
  ▶ NP-complete problem
  ▶ if satisfiable $\rightarrow$ satisfying assignment
  ▶ if not $\rightarrow$ unsat core (subset of formula’s clauses)
Example

First step is unwinding loops (to cover the bound)

```c
1: int i=4;
2: int s=0;
3: while (1) {
4:   s+=i;
5:   if (i>0)
6:     i--;
7:   assert(s<10);
8: }
```

...
Example

1: int i=4;
2: int s=0;
3: 
4: s+=i;
5: if (i>0)
6: i--;
7: assert(s<10);
8: s+=i;
9: if (i>0)
10: i--;
11: assert(s<10);

\[ f_1: (pc_1 = 1) \land (i_2 = 4) \land (pc_2 = 2) \]
\[ f_2: (pc_2 = 2) \land (i_3 = i_2) \land (s_3 = 0) \land (pc_3 = 3) \]
\[ f_3: (pc_3 = 3) \land (i_4 = i_3) \land (s_4 = s_3) \land (pc_4 = 4) \]
\[ f_4: (pc_4 = 4) \land (i_5 = i_4) \land (s_5 = s_4 + i_4) \land (pc_5 = 5) \]
\[ f_5: (pc_5 = 5) \land (i_6 = i_5) \land (s_6 = s_5) \land (pc_6 = 6) \]
\[ f_6: (pc_6 = 6) \land \left( ((i_6 > 0) \land (i_7 = i_6 - 1)) \lor ((i_6 \leq 0) \land (i_7 = i_6)) \right) \land (s_7 = s_6) \land (pc_7 = 7) \]
\[ f_7: (pc_7 = 7) \land (s_7 \geq 10) \land (pc_8 = 8) \]
\[ f_8: (pc_8 = 8) \land (i_9 = i_8) \land (s_9 = s_8 + i_8) \land (pc_9 = 9) \]
\[ f_9: (pc_9 = 9) \land (i_{10} = i_9) \land (s_{10} = s_9) \land (pc_{10} = 10) \]
\[ f_{10}: (pc_{10} = 10) \land \left( ((i_{10} > 0) \land (i_{11} = i_{10} - 1)) \lor ((i_{10} \leq 0) \land (i_{11} = i_{10})) \right) \land (s_{11} = s_{10}) \land (pc_{11} = 11) \]
\[ f_{11}: (pc_{11} = 11) \land (s_{11} \geq 10) \land (pc_{12} = 12) \]
Example

- Assertion expressions negated
- Main formula:

\[ \bigwedge_{i=0..k} f_i \]

- Satisfying assignment is found \( \rightarrow \) assertion is violated
- If not, we know that there is no assertion violation in \( k \) steps
HW application: Four-bit adder

\[ X_3 X_2 X_1 X_0 \quad X_3 X_2 X_1 X_0 \]

\[ \ldots \quad A \quad B \quad \ldots \]

\[ A \quad B \quad A \quad B \]

\[ S \quad C_{in} \quad C_{out} \quad S \]

\[ X_3 X_2 X_1 X_0 \quad X_3 X_2 X_1 X_0 \]

\[ C_0 = 0 \]
HW implementation of addition operation (1-bit):

- **A, B** – input bits, **C\textsubscript{in}, C\textsubscript{out}** – carry bits, **S** – output
**Logical representation of BIT-ADDER**

\[
\begin{align*}
(A \land B \land C_{in}) & \implies (S \land C_{out}) \\
(\neg A \land B \land C_{in}) & \implies \neg(S \land C_{out}) \\
(A \land \neg B \land C_{in}) & \implies \neg(S \land C_{out}) \\
(A \land B \land \neg C_{in}) & \implies \neg(S \land C_{out}) \\
(\neg A \land \neg B \land C_{in}) & \implies (S \land \neg C_{out}) \\
(\neg A \land B \land \neg C_{in}) & \implies (S \land \neg C_{out}) \\
(A \land \neg B \land \neg C_{in}) & \implies (S \land \neg C_{out}) \\
(\neg A \land \neg B \land \neg C_{in}) & \implies (\neg S \land \neg C_{out})
\end{align*}
\]

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C_{in}</td>
<td>S</td>
<td>C_{out}</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>--------</td>
<td>---</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation of $S$

\[ S_0 = A_0 \text{xor} B_0 \]
\[ C_{\text{out},0} = A_0 \text{ and } B_0 \]
\[ S_1 = A_1 \text{xor} B_1 \text{xor} C_{\text{out},0} \]
\[ C_{\text{out},1} = \left( (A_1 \text{xor} B_1) \text{ and } C_{\text{out},0} \right) \text{ or } (A_1 \text{ and } B_1) \]
\[ S_2 = A_2 \text{xor} B_2 \text{xor} C_{\text{out},1} \]
\[ C_{\text{out},2} = \left( (A_2 \text{xor} B_2) \text{ and } C_{\text{out},1} \right) \text{ or } (A_2 \text{ and } B_2) \]
\[ S_3 = A_3 \text{xor} B_3 \text{xor} C_{\text{out},2} \]
\[ C_{\text{out},3} = \left( (A_3 \text{xor} B_3) \text{ and } C_{\text{out},2} \right) \text{ or } (A_3 \text{ and } B_3) \]
Evaluation of $S$

$$S_0 = A_0 \text{xor} B_0$$

$$C_{out,0} = A_0 \text{ and } B_0$$

$$S_1 = A_1 \text{xor} B_1 \text{xor} C_{out,0}$$

$$C_{out,1} = \left( (A_1 \text{xor} B_1) \text{ and } C_{out,0} \right) \text{ or } (A_1 \text{ and } B_1)$$

$$S_2 = A_2 \text{xor} B_2 \text{xor} C_{out,1}$$

$$C_{out,2} = \left( (A_2 \text{xor} B_2) \text{ and } C_{out,1} \right) \text{ or } (A_2 \text{ and } B_2)$$

$$S_3 = A_3 \text{xor} B_3 \text{xor} C_{out,2}$$

$$C_{out,3} = \left( (A_3 \text{xor} B_3) \text{ and } C_{out,2} \right) \text{ or } (A_3 \text{ and } B_3)$$
Four bits model

- In each step, one bit of $S$ and one carry bit are computed.
- To reason about any bit, four steps are enough.
  - E.g., if we are interested in $C_{out,3}$ setting some flags.
- That means that from the model of hw we can easily set the threshold for bounded model checking.
Bounds are not that limiting...

- We obtain a minimal counterexamples
  - always the shortest found first
- Connected with loop invariants, properties of infinite paths can be verified
  - this way, unbounded (infinite) models can be analyzed
  - though not really model-checked
- If we manage to traverse entire state space, it is actually equal to unbounded MC