Example: Model Checking and Repair

Barbara Jobstmann
Verimag/CNRS (Grenoble, France)

November 13, 2009
Homework

1. Given \( G = (S, S_0, E) \) and \( F \subseteq Q \), give an algorithm that computes the 0-Attractor(\( F' \)) in time \( O(|E|) \).

Solution:

1. Preprocessing: Compute for every state \( s \in S_1 \) outdegree \( \text{out}(s) \)
2. Set \( n(s) := \text{out}(s) \) for each \( s \in S_1 \)
3. To breadth-first search backwards from \( F \) with the following conventions:
   - mark all \( s \in F \)
   - mark \( s \in S_0 \) if reached from marked state
   - mark \( s \in S_1 \) if \( n(s) = 0 \), other set \( n(s) := n(s) - 1 \).

The marked vertices are the ones of Attr\(_0(\mathcal{F})\).
Homework (cont.)

2. Consider the game graph shown in below and the following winning conditions:

   (a) $\text{Occ}(\rho) \cap \{1\} \neq \emptyset$ and
   (b) $\text{Occ}(\rho) \subseteq \{1, 2, 3, 4, 5, 6\}$ and
   (c) $\text{Inf}(\rho) \cap \{4, 5\} \neq \emptyset$.

Compute the winning regions and corresponding winning strategies showing the intermediate steps (i.e., the Attractor and Recurrence sets) of the computation.
3. Given a game graph $G = (S, S_0, T)$ and a set $F \subseteq S$. Let $W_0$ and $W_1$ be the winning regions of Player 0 and Player 1, respectively, in the Buchi game $(G, F)$. Prove or disprove:

(a) The winning set of Player 0 in the safety game for $(G, W_0)$ is $W_0$,

(b) If $f_0$ is a winning strategy for Player 0 in the safety game for $(G, W_0)$, then $f_0$ is also a winning strategy for Player 0 in the Buchi game for $(G, F)$,

(c) the winning set of Player 1 in the guaranty game for $(G, W_0)$ is $W_1$, and

(d) if $f_1$ is a winning strategy for Player 1 in the guaranty game for $(G, W_0)$, then $f_1$ is also a winning strategy for Player 1 in the Buchi game $(G, F)$. 
MC and Repair Example
while(...) {
    if (...) {
        lock();
        gotlock++;
    }

    // ...

    if (gotlock!=0)
        unlock();
    gotlock--;
}
Properties

1. P1: do not acquire a lock twice
2. P2: do not call unlock without holding the lock
Transition System of P

Variables: line, gotlock

l=1, gl=-1
l=2, gl=-1
l=3, gl=-1
l=4, gl=-1
l=5, gl=-1
l=6, gl=-1
l=7, gl=-1
l=8, gl=-1
Recall LTL

Boolean Operators: ¬, ∧, ∨, →, ...

Temporal Operators:

- **next:** \( \bigcirc \varphi \) ... in the next step \( \varphi \) holds
- **until:** \( \varphi_1 U \varphi_2 \) ... at some point in the future \( \varphi_2 \) holds and until then \( \varphi_1 \) holds

Useful abbreviations:

- **eventually:** \( \Diamond \varphi = true U \varphi \)
- **always:** \( \square \varphi = \neg \Diamond \neg \varphi \)
- **weakuntil:** \( \varphi_1 W \varphi_2 = (\varphi_1 U \varphi_2) \lor \square \neg \varphi_1 \)

Note that
\[
\neg (\varphi_1 U \varphi_2) = (\neg \varphi_2 U \neg \varphi_1 \land \neg \varphi_2) \lor \square \neg \varphi_2 = \neg \varphi_2 W (\neg \varphi_1 \land \neg \varphi_2).
\]
Our properties in LTL

1. P1: do not acquire a lock twice
   Whenever we have called lock, we are not allowed to call it again before calling unlock.
Our properties in LTL

1. P1: do not acquire a lock twice
   Whenever we have called lock, we are not allowed to call it again before calling unlock. \(\Box((l = 3) \rightarrow \Diamond(\neg(l = 3) W(l = 6)))\)
Our properties in LTL

1. P1: do not acquire a lock twice
   Whenever we have called lock, we are not allowed to call it again before calling unlock. $\square((l = 3) \rightarrow \bigcirc(\neg(l = 3) W(l = 6)))$

2. P2: do not call unlock without holding the lock
Our properties in LTL

1. **P1**: do not acquire a lock twice
   Whenever we have called lock, we are not allowed to call it again before calling unlock. $\Box((l = 3) \rightarrow \Box(\neg(l = 3) \mathcal{W}(l = 6)))$

2. **P2**: do not call unlock without holding the lock
   $(\neg(l = 6) \mathcal{W}(l = 3)) \land (l = 6 \rightarrow \Box(\neg(l = 6) \mathcal{W}(l = 3)))$
From LTL to Automata: Expansion rules

- □φ = φ ∧ □□φ
- ◊φ = φ ∨ □◊φ
- φ₁ U φ₂ = φ₂ ∨ (φ₁ ∧ □φ₁ U φ₂)
- φ₁ W φ₂ = φ₂ ∨ (φ₁ ∧ □φ₁ W φ₂)

Example: □((l = 3) → □(¬(l = 3) W (l = 6)))

Shortcuts: l₃ for (l = 3) and l₆ for (l = 6)

φ = □(¬l₃ ∨ (l₃ ∧ □(¬l₃ W l₆)))

Expand: (¬l₃ ∨ (l₃ ∧ □(¬l₃ W l₆))) ∧ □φ

DNF: s₀ ∨ s₁ with s₀ = ¬l₃ ∧ □φ and s₁ = l₃ ∧ □(¬l₃ W l₆ ∧ φ)
Example

Expand: $\neg l_3 \mathcal{W} l_6 \land \varphi$

$$(l_6 \lor (\neg l_3 \land \lozenge (\neg l_3 \mathcal{W} l_6))) \land ((\neg l_3 \land \lozenge \varphi) \lor (l_3 \land \lozenge (\neg l_3 \mathcal{W} l_6 \land \varphi)))$$

(1) $l_6 \land \neg l_3 \land \lozenge \varphi : s_2$

(2) $l_6 \land l_3 \cdots = \text{false}$

(3) $(\neg l_3 \land \lozenge (\neg l_3 \mathcal{W} l_6)) \land (\neg l_3 \land \lozenge \varphi) : s_3$

(4) $(\neg l_3 \land \cdots \land l_3 \cdots = \text{false}$
Model Checking

\[ L(\text{Program}) \subseteq L(P1) \]

\[ L(\text{Program}) \cap L(\neg P1) = \emptyset \]
Automaton for $\neg P1$

$\neg P1 = \neg \Box (l_3 \rightarrow \bigcirc (\neg l_3 \lor l_6))$

$\neg P1 = \lozenge (l_3 \land \bigcirc (\neg l_6 \lor l_3))$

Simplified version:
Product of Program and Property

\[
\begin{array}{c}
l=1, gl=-1, s_1 \\
l=2, gl=-1, s_1 \\
l=3, gl=-1, s_1 \\
l=4, gl=-1, s_2 \\
\vdots
\end{array}
\]
Counterexample

1. Line 1: enter while loop
2. Line 2: skip over if
3. ...
4. Line 1: enter while loop
5. Line 2: enter if (call lock)
6. ...
7. Line 1: enter while loop
8. Line 2: enter if (call lock again)

```plaintext
... 1 while(...) {
  2 if (...) {
    3 lock();
    4 gotlock++;
  }
  ...
  ...
  5 if (gotlock!=0)
    6 unlock();
    7 gotlock--;  
  }
  8 ...
```
Repair
Repair: Step 1 - Free variables

1     while(...) {
2         if (...) {
3             lock();
4             gotlock=?;
5         }
6     ...
7     ...
8     ...
9     if (gotlock!=0)
10        unlock();
11        gotlock=?;
12     }
13     ...
14     ...
Game on P

Variables: line, gotlock

l=1, gl=-1  l=1, gl=0  l=5, gl=1
l=2, gl=-1  l=2, gl=0
l=3, gl=-1  l=3, gl=0
l=4, gl=-1  l=4, gl=0
l=5, gl=-1  l=5, gl=0
l=6, gl=-1  l=6, gl=0
l=7, gl=-1  l=7, gl=0
l=8, gl=-1  l=8, gl=0

l=1, gl=-1  l=2, gl=-1  l=3, gl=-1  l=4, gl=-1  l=5, gl=-1  l=6, gl=-1  l=7, gl=-1  l=8, gl=-1
Repair: Winning Condition

Note in MC: non-determinism due to input and due to automaton are treated the same way!

In Game: non-determinism may cause troubles.
Recall,

$$\varphi = \square(\neg l_3 \lor (l_3 \land \bigcirc(\neg l_3 W l_6)))$$

Note: this is a safety automaton.
Add Automaton to Game on P

Variables: line, gotlock
A Winning Strategy

Variables: line, gotlock

- l=1, gl=0
- l=2, gl=0
- l=3, gl=0
- l=4, gl=0
- l=5, gl=0
- l=5, gl=1
- l=6, gl=0
- l=6, gl=1
- l=7, gl=0
- l=7, gl=1
- l=8, gl=0
A Correct Program

```c
while(...) {
    if (...) {
        lock();
        gotlock=1;
    }
    ...
    ...
    ...
    if (gotlock!=0)
        unlock();
    gotlock=0;
}
...
```