Supervisory Control (4CM30)
Data-based synthesis

Michel Reniers

M.A.Reniers@tue.nl

2016-2017
Previously

Summary previous lectures:
- Supervisory control
- Proper supervisors: nonblocking, controllability
- Maximal permissiveness
- Basic supervisory control problem
- Synthesis procedure

Send your answers to exercises Chapter 3 and Chapter 4 to me by email. I will include some correct ones in “Answers to selection of exercises”.
Basic supervisory control problem

For plant $P$, find a maximally permissive proper supervisor $S$.

- Supervisory control synthesis is a method / algorithm that delivers a specific maximally permissive proper supervisor for a given plant, if it exists.

- It uses the uncontrolled system as a starting point.

- Assumption: single plant automaton without variables is input for synthesis procedure.

- Network of automata $\Rightarrow$ single automaton.

- Variables will be discussed today.

- Requirements will be discussed next time.
Synthesis for plants with variables

Is there a need for supervising this plant?
Synthesis for plants with variables

\[ x < 8 \\
\text{e}_1 \\
x := x + 2 \]

\[ x < 7 \\
\text{e}_2 \\
x := x + 1 \]

Is there a need for supervising this plant?

Application of synthesis then results in removing last edge!

Drawbacks:
- plant without variables has size $|L| \times |\text{dom}(x)|$
- supervisor is less understandable
Synthesis for plants with variables

Is there a need for supervising this plant?

Application of synthesis then results in removing last edge!
Synthesis for plants with variables

Is there a need for supervising this plant?

Application of synthesis then results in removing last edge!

Drawbacks:
- plant without variables has size $|L| \times |\text{dom}(x)|$
- supervisor is less understandable
Data-based synthesis

Data-based supervisory control problem:

For plant $P$ (which is now allowed to use variables), find a maximally permissive proper supervisor $S$
Data-based synthesis algorithm

- start from plant automaton
- adapt the plant automaton repeatedly until the situation arises that no more changes occur

Each of these iterations consists of three phases:

1. Compute nonblocking conditions for all locations of the plant.
2. Compute bad state conditions for all locations.
3. Adapt guards of transitions with controllable events to obtain an adapted automaton.
Some useful notation

\( \text{Pred}[u] \): predicate \( \text{Pred} \) in which all occurrences of the variables are replaced by the right-hand sides of their updates

Example

\[
(x + y \geq 3)[x := x + y] = (x + y) + y \geq 3 \rightarrow x + 2y \geq 3
\]

Exercise

\[
((x \cdot y \leq x)[x := y/2])[y := 2x]
\]

Exercise

\[
(x \cdot y \leq x)[x := y/2, y := 2x]
\]
Phase 1: Nonblocking for automata with variables

- Nonblocking depends on the actual value of the variables in the location.
- Associate with each location a predicate that states for which combinations of values of variables the location is nonblocking.

Exercise
Give nonblocking condition for each location:

\begin{align*}
x &:= x + 2 & x &\leq 4 \\
x &:= x + 1 & x &:= x − 1
\end{align*}

In general, not easy to “guess”
Phase 1: Nonblocking for automata with variables

- Nonblocking depends on the actual value of the variables in the location.
- Associate with each location a predicate that states for which combinations of values of variables the location is nonblocking.

**Exercise**
*Give nonblocking condition for each location*
Phase 1: Nonblocking for automata with variables

▶ Nonblocking depends on the actual value of the variables in the location.
▶ Associate with each location a predicate that states for which combinations of values of variables the location is nonblocking.

Exercise

*Give nonblocking condition for each location*

In general, not easy to “guess”
Computing nonblocking conditions

- Determining the nonblocking conditions is done iteratively
- Label all marked locations with nonblocking condition \textit{true} and all non-marked locations with nonblocking condition \textit{false}.
- In each next iteration:

\[ N_i := N_i \lor (g_j \land N_j[u_j]) \lor (g_k \land N_k[u_k]) \lor \cdots \]
Computed solution

\[ N_i := N_i \lor (g_j \land N_j[u_j]) \lor (g_k \land N_k[u_k]) \lor \cdots \]
Computed solution

\[ N_i := N_i \lor (g_j \land N_j[u_j]) \lor (g_k \land N_k[u_k]) \lor \cdots \]
Computation solution

\begin{table}[h]
\begin{tabular}{|c|c|c|}
\hline
Loc. 0 & Loc. 1 & Loc. 2 \\
\hline
false & false & true \\
false \lor (true \land false[x := x+2]) & false \lor (x \leq 4 \land true[x := x+1]) & true \\
false \lor (true \land x \leq 4[x := x+2]) & x \leq 4 \lor (x \leq 4 \land true[x := x+1]) & true \\
= false & = x \leq 4 & \\
= x \leq 2 & = x \leq 4 & \\
\hline
\end{tabular}
\end{table}

\[ N_i := N_i \lor (g_j \land N_j[u_j]) \lor (g_k \land N_k[u_k]) \lor \cdots \]
Computed solution

\[
\begin{align*}
\text{Loc. 0} & \quad \text{false} \\
& \quad \text{false} \lor (\text{true} \land \text{false}[x := x+2]) \\
& \quad = \text{false} \\
& \quad \text{false} \lor (\text{true} \land x \leq 4[x := x+2]) \\
& \quad = x \leq 2 \\
& \quad \text{x} \leq 2 \\
& \quad \text{false} \lor (\text{true} \land N_j[u_j]) \\
& \quad \text{true} \\
\text{Loc. 1} & \quad \text{false} \\
& \quad false \lor (x \leq 4 \land true[x := x+1]) \\
& \quad = x \leq 4 \\
& \quad \text{x} \leq 4 \lor (x \leq 4 \land true[x := x+1]) \\
& \quad = x \leq 4 \\
& \quad \text{x} \leq 4 \\
\text{Loc. 2} & \quad \text{true} \\
& \quad \text{false} \lor (\text{true} \land N_j[u_j]) \\
& \quad \text{true} \\
& \quad \text{false} \lor (\text{true} \land N_j[u_j]) \\
& \quad \text{false} \lor (\text{true} \land N_j[u_j]) \\
& \quad \text{false} \lor (\text{true} \land N_j[u_j]) \\
\end{align*}
\]

\[N_i := N_i \lor (g_j \land N_j[u_j]) \lor (g_k \land N_k[u_k]) \lor \cdots\]
Exercise: compute nonblocking conditions

\[ x = 0 \]

\[ x < 8 \]
\[ e_1 \]
\[ x := x + 2 \]

\[ x < 7 \]
\[ e_2 \]
\[ x := x + 1 \]

\[ x < 8 \]

\[ x < 7 \]

Iteration | Location 0 | Location 1
Exercise: compute nonblocking conditions

\[ x \leq 8 \]
\[ e_1 \]
\[ x := x + 2 \]
\[ x = 0 \]
\[ e_2 \]
\[ x := x + 1 \]

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Location 0</th>
<th>Location 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>
Exercise: compute nonblocking conditions

\[ x < 8 \]
\[ e_1 \]
\[ x := x + 2 \]

\[
\begin{array}{c}
\text{Iteration} & \text{Location 0} & \text{Location 1} \\
0 & \text{true} & \text{false} \\
1 & \text{true} & x < 7 \\
\end{array}
\]
Exercise: compute nonblocking conditions

\[ x = 0 \]

\[ e_1 \]

\[ x < 8 \]

\[ x := x + 2 \]

\[ e_2 \]

\[ x < 7 \]

\[ x := x + 1 \]

\[ x < 7 \]

\[ e_1 \]

\[ x := x + 2 \]

\[ x < 8 \]

\[ e_2 \]

\[ x < 7 \]

\[ x := x + 1 \]

\[ x < 7 \]

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Location 0</th>
<th>Location 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>1</td>
<td>true</td>
<td>x &lt; 7</td>
</tr>
<tr>
<td>2</td>
<td>true</td>
<td>x &lt; 7</td>
</tr>
</tbody>
</table>
Phase 2: Bad state conditions

▶ purpose of nonblocking conditions

▶ not allowed to enforce the required restrictions by adapting guard for uncontrollable events

▶ such restrictions are propagated backwards until an edge with a controllable event is encountered

▶ iterative bad state condition computation
Computing bad state conditions

- start from plant automaton and computed nonblocking conditions
- initial bad state condition for each location is the logical negation of the nonblocking condition of that location:
  \[ B_i := \neg N_i \]
- In each next iteration:
  \[ B_i := B_i \lor (g_k \land B_k[u_k]) \lor \cdots \]

- Only for uncontrollable events!
Computed bad state conditions, solution

\[ x := x + 2 \quad \text{from} \quad 0 \rightarrow 1 \]

\[ x := x - 1 \quad \text{from} \quad 1 \rightarrow 0 \]

\[ x := x + 1 \quad \text{from} \quad 1 \rightarrow 2 \]

\[ x \leq 4 \]

<table>
<thead>
<tr>
<th>Location 0</th>
<th>Location 1</th>
<th>Location 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x &gt; 2 )</td>
<td>( x &gt; 4 )</td>
<td>false</td>
</tr>
<tr>
<td>( x &gt; 2 )</td>
<td>( x &gt; 4 )</td>
<td>false</td>
</tr>
</tbody>
</table>

\[ B_i := B_i \lor (g_k \land B_k[u_k]) \lor \cdots \]
Exercise

Nonblocking conditions: $N_0 = true$ and $N_1 = x < 7$
Nonblocking conditions: \( N_0 = true \) and \( N_1 = x < 7 \)

Bad state conditions:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Location 0</th>
<th>Location 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>false</td>
<td>( x \geq 7 )</td>
</tr>
<tr>
<td>1</td>
<td>false</td>
<td>( x \geq 7 )</td>
</tr>
</tbody>
</table>
Phase 3: Adapts guards

- use computed bad state conditions; these express which combinations of values of variables should be avoided in a specific location, taking into account the limitation that guards of uncontrollable events may not be altered.

- guard $g$ of an edge from $i$ to $j$ with a controllable event and update $u$ is adapted:

$$i \xrightarrow{g \land \neg B_j[u]} j := i \xrightarrow{g \land B_j[u]} e \xrightarrow{u} j$$
Adapted guards, solution

\[ B_0 = x > 2 \quad B_1 = x > 4 \quad B_2 = false \]

Adapt guards:

\[ i \xrightarrow{g \land \neg B_j[u]} j := i \xrightarrow{g} j \]

\[ x := x + 2 \quad x \leq 4 \quad x := x + 1 \]

\[ x \leq 3 \quad x := x - 1 \]
Exercise, adapt guards

\[ x < 8 \]
\[ e_1 \]
\[ x := x + 2 \]

\[ x = 0 \rightarrow 0 \rightarrow 1 \]
\[ x < 7 \]
\[ e_2 \]
\[ x := x + 1 \]

Bad state conditions: \( B_0 = false \) and \( B_1 = x \geq 7 \)
Exercise, adapt guards

$0 \xrightarrow{x < 8} e_1 \xrightarrow{x := x + 2} 1$

$x < 7 \xrightarrow{e_2} x := x + 1$

Bad state conditions: $B_0 = \text{false}$ and $B_1 = x \geq 7$

for event $e_1$: $x < 8 \land \neg (x \geq 7)[x := x + 2] = x < 8 \land \neg (x + 2 \geq 7) = x < 8 \land \neg (x \geq 5) = x < 8 \land x < 5 = x < 5$
Guided self-study

Data-based synthesis (Section 4.3)
- Notation: Exercise 4.23
- Nonblocking conditions: Exercises 4.24 - 4.28
- Bad state conditions: Exercise 4.29
- Adapt guards: Exercise 4.30
- Synthesis: Exercises 4.31 - 4.35