Computing RW Supervisors: 20 odd years of battling complexity

Martin Fabian, Professor
Outline

- How did I get here?
- What is a hard problem?
- How do we battle computational complexity?
- Where are we now?
- Where to next?
PLC programming
Object oriented modeling

**Figure 1.1:** Object-oriented model of a flexible assembly cell. The PI’s represent product route specifications.

Operating concurrently, the internal resources constitute the plant to be controlled. This concurrent execution is modeled by *full synchronous composition* (FSC) of the internal resources, see Hoare (1985). This allows the internal resources to be coupled and thus able to synchronize, for instance, over mutual use of tools.

**Figure 1.2:** Each internal resource consists of a general and a specific part, thus separating the control of the assembly process from the control of the individual devices.

**Figure 1.3:** *M1* represents a general production unit, such as a lathe or a mill. *M4* represents a general assembly unit, and *R* represents a general transporting device.
ON THE SUPREMUM CONTROLLABLE SUBLANGUAGE OF A GIVEN LANGUAGE

R.M. Verbeek and P.J. Ramadge
Systems Control Group
Dept. of Electrical Engineering
University of Toronto
Toronto M5S 1A4, CANADA

Abstract

The concept of controllable language has been shown to play a basic role in the existence theory of supervisory controls for discrete event processes. In this paper the supremum controllable sublanguage 8 of a given language L is characterized as the largest fixpoint of a certain operator 2. In the case where the languages involved are regular it is shown that the fixpoint 8 can be computed as the limit of the (finite) sequence (8n) given by 81 = L, 8n+1 = 2(8n). An effective computational algorithm is developed, and three examples are provided for illustration.

Keywords: supervisory control, discrete event process, controllable language.

1. Introduction

The concept of controllable language was introduced in [Ramadge and Wonham, 1982], where it was shown to play a basic role in the existence theory of supervisory control for discrete event processes. In this paper we characterize the supremum controllable sublanguage 8 of a given language L as the largest fixpoint of a certain operator 2. In the case where the languages involved are regular we show that the fixpoint 8 can be computed as the limit of the (finite) sequence of languages 8n given by

\[ 8_1 = L, \quad 8_{n+1} = 2(8_n) \]

Three computational examples are provided for illustration.

2. A Fixpoint Characterization of \( \text{sup} \{8_n\} \)

Let \( I \) be the set of all languages over \( I \) (i.e., \( I = \text{power set of } I \)) and define the operator

\[ 8^+ : I \rightarrow I \]

according to

\[ 8^+ (X) = \{ \lambda \in I : T^+ \subseteq T, \quad T = T_{\lambda} \cap \lambda(X) \} \]

where \( T^+ \) is the positive cone of \( T \subseteq I \).

It is easy to check (cf. the proof of Prop. 7.1 in [Ramadge and Wonham, 1982]) that the condition in (2.3) is closed under arbitrary unions, so \( 8^+ \) is well defined. We now have

Proposition 2.1

Let \( S = \text{sup} \{8_n\} \). Then \( S = 8^+(S) \) and \( S \subseteq 8 \) for every \( X \) such that \( X \subseteq 8 \). A language \( X \) such that \( X = 8(X) \) is a fixpoint of \( 8 \). This prop. 2.1 describes \( \text{sup} \{8_n\} \) as the largest fixpoint of \( 8 \).

Conclusions

This work is concerned, in one way or another, with the problem of how to achieve or verify the orderly flow of events, and to this end how to bring together ideas from logic, language and automaton theory. However, while control problems are implicit in much of the work.

Key words: discrete event systems, control, automata.

AMS(MOS) subject classifications. 93C10, 93B50, 93C30

1. Introduction. In this paper we study the control of a class of systems broadly known as discrete event processes. The principal features of such processes are that they are discrete, asynchronous and (possibly) nondeterministic. Typical instances include computer networks, flexible manufacturing systems, and the start-up and shut-down procedures of industrial plants.

While numerous practical examples are described in the literature on simulation (see especially Fishman [1978] and Zeigler [1984]), there is at the present time apparently no unifying theory for the control of discrete event processes. Nor is it entirely clear what such a theory ought to encompass. Numerous approaches to the modeling of discrete event processes have appeared in the literature. A general sampling of these could include Boolean models (Aveyard [1974]); Petri nets (Peterson [1981]); formal languages (Beauquier and Nivat [1980], Park [1981]); temporal logic (Pnueli [1979], Halpern and Owicki [1983]); and port automata and flow networks (Milner [1979], Steenstrup, Arbib and Manes [1981]). All of this work is concerned, in one way or another, with the problem of how to achieve or verify the orderly flow of events, and to this end how to bring together ideas from logic, language and automaton theory. However, while control problems are implicit in much of the work.
Looking the R&W papers over...
And over...

Proposition 4.2. Let $S$ and $T$ be nonblocking supervisors for $G$. Then the supervisor $S \land T$ is nonblocking if and only if the languages $L(S/G)$ and $L(T/G)$ are nonconflicting.

Proof:

$S \land T$ is nonblocking if and only if $L(S \land T) = L(S) \cap L(T)$.

If and only if $L(S/G) \cap L(T/G) = L(S/G) \cap L(T/G)$.

If and only if $L(S/G) \cap L(T/G) = L(S/G) \cap L(T/G)$.

If and only if $L(S/G)$ and $L(T/G)$ are nonconflicting.

5. Modular Supervision

The basic problem in supervisory control is to synthesise a supervisor $S$ to achieve a prescribed closed loop behavior, i.e., for a given $K \subseteq 2^*$ synthesise a supervisor $S$ such that $L(S) = K$.

Hmm...
Synchronous composition as interaction

• Kumar, Garg, Marcus (1991)
• Synchronous composition as model of control
• Marking as specification

On controllability and normality of discrete event dynamical systems

Ratnesh Kumar, Vijay Garg, Steven I. Marcus
Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX 78712-1088, USA
Received 3 November 1990
Revised 8 March 1991

Abstract: This paper studies the supervisor synthesis problem of Discrete Event Dynamical Systems (DEDS) through the use of synchronous composition of the plant and the supervisor and discusses some of the simplifications achieved by using the synchronous composition to model supervisory control. An alternate definition of controllability is presented. This definition of controllability is then used to derive an algorithm that is computationally more efficient than previously existing ones for the construction of the supremal controllable sublanguage; the algorithm is also shown to be optimal. The observability and normality issues arising due to the partial observation of the system dynamics under an arbitrary mask are then investigated. Closed form representations of the supremal normal, prescribed behavior. Supervisors designed for closed loop control, so that none of the uncontrollable events that can occur in the plant behavior are prevented from occurring in the closed loop system, are called complete.

Since under the closed-loop control of a complete supervisor, all uncontrollable events that can occur in the plant, can also occur in the closed loop system, there may not always exist a complete supervisor for a given plant such that the closed loop system has a prespecified desired behavior. The question then arises as to how to construct a complete supervisor which is minimally restrictive, so that the closed loop system can engage in some maximal behavior and still maintain the prescribed behavioral constraint. Another problem related to the design of such supervisors is to reduce the computational complexity of finding the minimally restrictive supervisor.
Synchronous composition as interaction

- Silvano Balemi (1992)
- Synchronous composition as model of control
- BDD-based synthesis
DESCO — Discrete Event Systems Controller

- Command line tool
- Precursor of Supremica
- Used in course
- The students HATED it!
- Experiments showed
  - Some problems were easy
  - Others were hard
  - But which was what?
What is a hard problem?

• Theoretical state-space
  • Does not tell us much
  • If anything at all
• Even actual state-space tells us very little
Transfer line – model

- 100 and 1000 serially connected transfer lines
- Actual global state-spaces
Transfer line – computation

- Linear growth in total number of supervisor states
- Quadratic growth in computation time
Model hardness measures

**SIZE** Number of automata in the model. The general idea is that larger sets of automata represent larger models, hence are they more complex. This assumption will shown to be incorrect.

**R** Number of reachable states in the model:

\[ R = |Q^{A1}| - |A^n| \]

Like in SIZE, it is assumed that a larger model is more complex. This assumption turns out to be incorrect.

**R^T** Number of theoretically reachable states in the model. This number is equal to the product of the number of states in each automata:

\[ R^T = \prod_{i=1}^{n} |Q_i| \]

- **Dependency set sizes**
  - Level-1, one step coupling
  - Level-2, two step coupling
- **LD-n**
  - Size of level-n dependency set divided by SIZE
- **Transfer line**
  - LD-1 is ~0.01
  - LD-2 is ~0.02
Pigeon hole problem

- LD-2 equal to 1.0 means all automata are coupled to all other in two steps.

Figure 4.6: PHP: (a) maximum level-1 dependency compared to (b) minimum level-2 dependency.
Decoupling is good!

- Let’s try and exploit it!
Modular/incremental synthesis

- Regard only the uncontrollable events
  - Lowers the dependency
- For each spec
  - Select the LDu-1 plant components
  - Synthesize controllable supervisor
- To guarantee minimally restrictive supervisor
  - Select also the LDu-n plants of the initially selected ones
- To avoid monolithic synthesis
  - Select LDu-n not all at once, but only as needed, "incrementally"
AGV Example

• Automatically guided vehicles
  • Each AGV has its own path
  • Pick and leave at stations
• Specifications
  • At most one AGV at each crossing
  • A crossing is a (mutex) zone
• ZoneX specifies
  • At most one AGV at the input station
• Plant has 30 million states
Compositional non-blocking verification

**Setting** A system of *(deterministic)* finite state automata (plants and supervisors/specifications)

\[ \{G_1, G_2, \cdots, G_n\} = \{P_1, P_2, \cdots, P_i, S_1, S_2, \cdots, S_j\}. \]

**Goal** Verify whether

\[ G = G_1 \parallel G_2 \parallel \cdots \parallel G_n \]

is nonblocking.

- Distinction between plant and spec irrelevant

Monolithic approach won’t work!
Compositional non-blocking verification

**Abstraction** Sometimes we can replace a component, say $G_1$, by a simpler component $G_1'$ without changing the result of the verification.

\[ G_1 \parallel G_2 \parallel \cdots \parallel G_n \text{ nonblocking} \iff G_1' \parallel G_2' \parallel \cdots \parallel G_n' \text{ nonblocking} \]

**Compositional approach** If we can apply these simplifications to the components *independently*, then we might as well use

\[ G_1' \parallel G_2' \parallel \cdots \parallel G_n'. \]

Next we can compose $G_1'$ and $G_2'$, and simplify $G_1' \parallel G_2'$ etc.
Conflict equivalence

**Definition 6** Two automata $G$ and $H$ are said to be conflict equivalent, $G \simeq_{\text{conf}} H$, if, for any automaton $T$, $G \parallel T$ is nonblocking if and only if $H \parallel T$ is nonblocking.

By definition, conflict equivalence is preserved under synchronous composition. Thus

$$G_A \simeq_{\text{conf}} H_A \quad \text{and} \quad G_B \simeq_{\text{conf}} H_B \implies G_A \parallel G_B \simeq_{\text{conf}} H_A \parallel H_B$$

Yes! Components can be abstracted independently, then composed and abstracted again.
Conflict equivalent abstractions

Local event  An event that is used solely by one automaton in the system

Hiding  Such events can be hidden, meaning that we ignore their identity and replace them by the silent event $\tau$

$\tau$  This event does not synchronize with anything else (including itself and other silent events) just like the original event

Nondeterministic  This makes an automaton nondeterministic

- Observation equivalence implies conflict equivalence
  - We can minimize with respect to observation equivalence, known algorithms
- Many other conflict equivalent abstractions are known
  - Implemented in Supremica
A quick view at some experimental data

<table>
<thead>
<tr>
<th>Name</th>
<th>Example</th>
<th>Aut</th>
<th>Size</th>
<th>Max</th>
<th>States</th>
<th>Trans</th>
<th>Time</th>
<th>Block</th>
<th>Block</th>
<th>Block</th>
<th>Block</th>
<th>Block</th>
<th>Block</th>
<th>Block</th>
<th>Block</th>
<th>Block</th>
<th>Heuristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGV</td>
<td>16</td>
<td>2.6 \times 10^7</td>
<td>413</td>
<td>1272</td>
<td>0.50s</td>
<td>false</td>
<td>39</td>
<td>19</td>
<td>0</td>
<td>131</td>
<td>91</td>
<td>131</td>
<td>minT \ maxL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AGVb</td>
<td>17</td>
<td>2.3 \times 10^7</td>
<td>255</td>
<td>778</td>
<td>0.35s</td>
<td>true</td>
<td>41</td>
<td>36</td>
<td>236</td>
<td>71</td>
<td>21</td>
<td>695</td>
<td>minT \ maxL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>veeriege13</td>
<td>53</td>
<td>9.7 \times 10^8</td>
<td>1239</td>
<td>5261</td>
<td>3.4s</td>
<td>false</td>
<td>88</td>
<td>80</td>
<td>0</td>
<td>115</td>
<td>581</td>
<td>2153</td>
<td>minT \ maxL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>veeriege13b</td>
<td>52</td>
<td>1.3 \times 10^9</td>
<td>11</td>
<td>46</td>
<td>0.12s</td>
<td>true</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>minT \ maxL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>veeriege14</td>
<td>65</td>
<td>4.5 \times 10^{10}</td>
<td>1897</td>
<td>8353</td>
<td>5.0s</td>
<td>false</td>
<td>130</td>
<td>125</td>
<td>0</td>
<td>201</td>
<td>963</td>
<td>3754</td>
<td>minT \ maxL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>veeriege14b</td>
<td>64</td>
<td>6.3 \times 10^{10}</td>
<td>11</td>
<td>46</td>
<td>0.13s</td>
<td>true</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>minT \ maxL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>big_bwt</td>
<td>31</td>
<td>3.1 \times 10^7</td>
<td>187</td>
<td>1474</td>
<td>0.91s</td>
<td>false</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>246</td>
<td>172</td>
<td>minT \ maxL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SNS</td>
<td>11</td>
<td>312</td>
<td>17</td>
<td>22</td>
<td>0.24s</td>
<td>false</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>3</td>
<td>minT \ maxL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PM3</td>
<td>38</td>
<td>5.7 \times 10^6</td>
<td>35</td>
<td>362</td>
<td>0.39s</td>
<td>false</td>
<td>18</td>
<td>8</td>
<td>0</td>
<td>63</td>
<td>6</td>
<td>470</td>
<td>minT \ maxL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPC</td>
<td>12</td>
<td>20592</td>
<td>256</td>
<td>985</td>
<td>0.41s</td>
<td>true</td>
<td>26</td>
<td>17</td>
<td>65</td>
<td>39</td>
<td>165</td>
<td>608</td>
<td>minT \ maxL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ftech2nlk</td>
<td>36</td>
<td>1.2 \times 10^8</td>
<td>66</td>
<td>698</td>
<td>0.92s</td>
<td>true</td>
<td>4</td>
<td>11</td>
<td>78</td>
<td>2</td>
<td>173</td>
<td>117</td>
<td>minT \ maxL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ftech2nlk</td>
<td>42</td>
<td>1.2 \times 10^8</td>
<td>135</td>
<td>2434</td>
<td>0.35s</td>
<td>true</td>
<td>10</td>
<td>15</td>
<td>129</td>
<td>4</td>
<td>97</td>
<td>91</td>
<td>mustL \ minS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tbed</td>
<td>84</td>
<td>13680</td>
<td>1800</td>
<td>9313</td>
<td>12s</td>
<td>true</td>
<td>0</td>
<td>6</td>
<td>2219</td>
<td>43</td>
<td>1018</td>
<td>457</td>
<td>mustL \ minS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fzei1e</td>
<td>67</td>
<td>3.0 \times 10^{11}</td>
<td>810</td>
<td>3169</td>
<td>5.4s</td>
<td>false</td>
<td>106</td>
<td>176</td>
<td>4</td>
<td>280</td>
<td>847</td>
<td>3500</td>
<td>mustL \ minS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AIT</td>
<td>117</td>
<td>1.0 \times 10^9</td>
<td>1050</td>
<td>4200</td>
<td>6.2s</td>
<td>false</td>
<td>135</td>
<td>150</td>
<td>10</td>
<td>159</td>
<td>1090</td>
<td>4296</td>
<td>mustL \ minS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>256philo</td>
<td>512</td>
<td>5.4 \times 10^{16}</td>
<td>35</td>
<td>86</td>
<td>6.6s</td>
<td>true</td>
<td>512</td>
<td>1</td>
<td>0</td>
<td>1535</td>
<td>3548</td>
<td>9769</td>
<td>maxS \ maxL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>812philo</td>
<td>1024</td>
<td>2.9 \times 10^{137}</td>
<td>35</td>
<td>86</td>
<td>17s</td>
<td>true</td>
<td>1024</td>
<td>1</td>
<td>0</td>
<td>3071</td>
<td>7132</td>
<td>19599</td>
<td>maxT \ maxL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1024philo</td>
<td>2048</td>
<td>8.5 \times 10^{574}</td>
<td>35</td>
<td>86</td>
<td>414s</td>
<td>true</td>
<td>2048</td>
<td>1</td>
<td>0</td>
<td>6143</td>
<td>14390</td>
<td>39260</td>
<td>maxS \ maxL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>126transfer</td>
<td>640</td>
<td>1.6 \times 10^{231}</td>
<td>24</td>
<td>67</td>
<td>5.7s</td>
<td>false</td>
<td>381</td>
<td>3</td>
<td>0</td>
<td>894</td>
<td>254</td>
<td>2055</td>
<td>minT \ maxL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>256transfer</td>
<td>1280</td>
<td>2.4 \times 10^{462}</td>
<td>24</td>
<td>67</td>
<td>29s</td>
<td>false</td>
<td>765</td>
<td>3</td>
<td>0</td>
<td>1790</td>
<td>510</td>
<td>4103</td>
<td>minT \ maxL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>812transfer</td>
<td>2560</td>
<td>5.8 \times 10^{2424}</td>
<td>24</td>
<td>67</td>
<td>3.3m</td>
<td>false</td>
<td>1533</td>
<td>3</td>
<td>0</td>
<td>3582</td>
<td>1022</td>
<td>8199</td>
<td>minT \ maxL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128aribiter</td>
<td>637</td>
<td>3.9 \times 10^{158}</td>
<td>318</td>
<td>941</td>
<td>14s</td>
<td>false</td>
<td>1205</td>
<td>1273</td>
<td>1482</td>
<td>1478</td>
<td>6226</td>
<td>6588</td>
<td>minT \ maxL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>256aribiter</td>
<td>1277</td>
<td>3.8 \times 10^{1684}</td>
<td>318</td>
<td>941</td>
<td>32s</td>
<td>false</td>
<td>2420</td>
<td>2593</td>
<td>3083</td>
<td>2987</td>
<td>13097</td>
<td>14243</td>
<td>minT \ maxL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>812aribiter</td>
<td>2557</td>
<td>5.7 \times 10^{4539}</td>
<td>318</td>
<td>941</td>
<td>69s</td>
<td>false</td>
<td>4848</td>
<td>5217</td>
<td>6211</td>
<td>5996</td>
<td>26581</td>
<td>28671</td>
<td>minT \ maxL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Never considers more than 2000 states, 10000 transitions.
What if verification says “No!”

- Give counter example
  - Trace to blocking state
  - User has to fix the issue and verify again
  - And again... And again...
- Or we do synthesis
  - Get rid of all counter examples at once
Verification vs synthesis

- Verification
  - Japanese archer shooting down one counter example at a time
- Synthesis
  - Shotgun blast that takes out all counter examples at once
Compositional synthesis

• Works much the same way as compositional verification
  • Need to consider plants and specs
  • Different abstractions
• Different property to preserve, namely...
Synthesis abstraction

Definition 9: [13] Let $G$ and $\tilde{G}$ be two deterministic automata with alphabet $\Sigma$. Then $\tilde{G}$ is a synthesis abstraction of $G$ with respect to the local events $\mathcal{Y} \subseteq \Sigma$, written $G \preceq_{\text{synth,} \mathcal{Y}} \tilde{G}$, if for every deterministic automaton $T = \langle \Sigma_T, Q_T, \rightarrow_T, Q^0_T \rangle$ such that $\Sigma_T \cap \mathcal{Y} = \emptyset$ the following holds,

$$\mathcal{L}(G \parallel T \parallel \text{supCN}(\tilde{G} \parallel T)) = \mathcal{L}(G \parallel T \parallel \text{supCN}(G \parallel T))$$

The supervisor calculated with the abstracted model should, when controlling the plant, generate the same behavior as the supervisor calculated from the non-abstracted model. In all possible contexts $T$, including "the rest of the system".
Synthesis abstractions

- Observation equivalence is **not** a synthesis abstraction
- A supervisor can use local events for control
- Unless the local event is uncontrollable
- "Uncontrollable observation equivalence" **is** a synthesis abstraction

- Bisimulation is also a synthesis abstraction
- We have also "Synthesis observation equivalence" as a synthesis abstraction
Some more experimental results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>agv</td>
<td>16</td>
<td>2.6\times10^7</td>
<td>true false</td>
<td>856</td>
<td>3.11</td>
<td>27.9</td>
<td>6</td>
<td>12339</td>
<td>0</td>
<td>30</td>
<td>208</td>
<td></td>
</tr>
<tr>
<td>agvb</td>
<td>17</td>
<td>2.3\times10^7</td>
<td>false false</td>
<td>562</td>
<td>0.81</td>
<td>61.3</td>
<td>7</td>
<td>9380</td>
<td>0</td>
<td>30</td>
<td>187</td>
<td></td>
</tr>
<tr>
<td>aip0alps</td>
<td>35</td>
<td>3.0\times10^6</td>
<td>false true</td>
<td>502</td>
<td>0.43</td>
<td>84.3</td>
<td>3</td>
<td>17</td>
<td>2</td>
<td>53</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>fencaiwon09b</td>
<td>29</td>
<td>8.9\times10^7</td>
<td>false true</td>
<td>182</td>
<td>0.27</td>
<td>118.4</td>
<td>6</td>
<td>917</td>
<td>4</td>
<td>56</td>
<td>57</td>
<td>328</td>
</tr>
<tr>
<td>fencaiwon09s</td>
<td>29</td>
<td>2.9\times10^6</td>
<td>false false</td>
<td>525</td>
<td>0.44</td>
<td>150.2</td>
<td>11</td>
<td>436</td>
<td>5</td>
<td>59</td>
<td>186</td>
<td>2</td>
</tr>
<tr>
<td>tbed-noderailb</td>
<td>84</td>
<td>3.1\times10^12</td>
<td>true false</td>
<td>4989</td>
<td>6.22</td>
<td>265.2</td>
<td>17</td>
<td>4982</td>
<td>0</td>
<td>12</td>
<td>158</td>
<td>112</td>
</tr>
<tr>
<td>tbed-uncont</td>
<td>84</td>
<td>3.6\times10^12</td>
<td>true false</td>
<td>4479</td>
<td>5.34</td>
<td>491.6</td>
<td>10</td>
<td>19737</td>
<td>1</td>
<td>1</td>
<td>190</td>
<td>73</td>
</tr>
<tr>
<td>verriegel3b</td>
<td>52</td>
<td>1.3\times10^9</td>
<td>false true</td>
<td>1367</td>
<td>1.80</td>
<td>218.2</td>
<td>1</td>
<td>4</td>
<td>77</td>
<td>64</td>
<td>1390</td>
<td>1796</td>
</tr>
<tr>
<td>verriegel4b</td>
<td>64</td>
<td>6.2\times10^10</td>
<td>false false</td>
<td>1382</td>
<td>4.86</td>
<td>250.5</td>
<td>1</td>
<td>4</td>
<td>21</td>
<td>71</td>
<td>189</td>
<td>622</td>
</tr>
<tr>
<td>6linke</td>
<td>53</td>
<td>2.4\times10^{14}</td>
<td>false true</td>
<td>3614</td>
<td>19.52</td>
<td>515.3</td>
<td>13</td>
<td>2073</td>
<td>15</td>
<td>48</td>
<td>1754</td>
<td>2103</td>
</tr>
<tr>
<td>6linki</td>
<td>53</td>
<td>2.7\times10^{14}</td>
<td>false true</td>
<td>2925</td>
<td>13.72</td>
<td>635.4</td>
<td>12</td>
<td>4017</td>
<td>12</td>
<td>49</td>
<td>1205</td>
<td>1897</td>
</tr>
<tr>
<td>6linkp</td>
<td>48</td>
<td>4.2\times10^{14}</td>
<td>false true</td>
<td>3614</td>
<td>26.62</td>
<td>538.3</td>
<td>17</td>
<td>2073</td>
<td>25</td>
<td>45</td>
<td>1731</td>
<td>2107</td>
</tr>
<tr>
<td>6linkre</td>
<td>59</td>
<td>6.2\times10^{14}</td>
<td>false false</td>
<td>240</td>
<td>1.01</td>
<td>584.9</td>
<td>19</td>
<td>375</td>
<td>10</td>
<td>51</td>
<td>221</td>
<td>279</td>
</tr>
<tr>
<td>tline100</td>
<td>401</td>
<td>6.5\times10^{150}</td>
<td>true false</td>
<td>50</td>
<td>3.44</td>
<td>252.4</td>
<td>201</td>
<td>79</td>
<td>0</td>
<td>495</td>
<td>1192</td>
<td>4126</td>
</tr>
<tr>
<td>tline1000</td>
<td>4001</td>
<td>2.8\times10^{1505}</td>
<td>true false</td>
<td>50</td>
<td>336.46</td>
<td>864.1</td>
<td>2001</td>
<td>79</td>
<td>0</td>
<td>4995</td>
<td>11992</td>
<td>41926</td>
</tr>
</tbody>
</table>

Less than 30 seconds for synthesis of some large systems.
Where are we now?

- Modular/incremental synthesis for FSM (controllability only)
- Compositional non-blocking verification for FSM
- Compositional synthesis for FSM
- Compositional non-blocking verification for EFSM
- What about synthesis for EFSM...?
Where to next?

- R&W-SCT has been around for 30 years
- Still, little industrial acceptance
- Requirements on formal approaches to development are coming
- This is good for us, if we can deliver
- Where to get the models?
- Where to get the specifications?
People

Bengt Lennartson
Professor

Knut Åkesson
Associate Professor

Arash Vahidi
PhD

Hugo Flordal
PhD

Robi Malik
Professor

Sahar Mohajerani
Post-doc UMich