Timed Automata Tools Grow Up: Seven Case Studies from Nijmegen

Frits Vaandrager Biniam Gebremichael

Institute for Computing and Information Sciences
Radboud University, Nijmegen
http://www.cs.ru.nl/ita

February 14, 2005, University of Kent

Outline

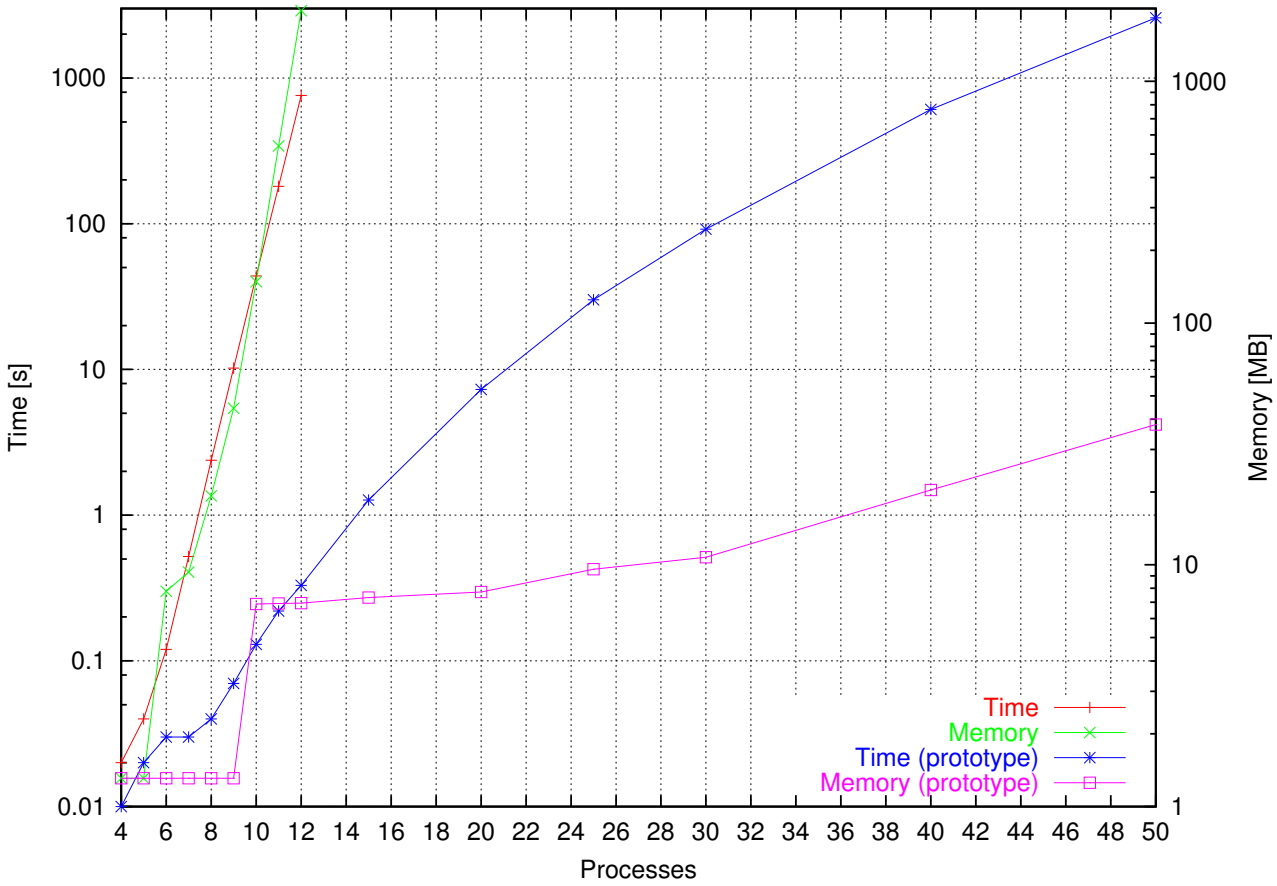
Timed Automata

- Model of finite automata enriched with real-values clock variables proposed by Rajeev Alur and David Dill in 1990
- Model checking tools under development since then; enormous progress has been made!
- Especially UPPAAL has become quite mature (for an academic prototype)
- Dozens of industrial applications: embedded controllers, distributed algorithms and protocols, scheduling problems,...

Some Recent improvements to Uppaal

- new techniques for static analysis and abstraction
- symmetry reduction
- heuristics for state space storage
- methods for cycle acceleration
- extension with cost functions and branch & bound techniques (Uppaal Cora)
- extension of syntax and user interface

Fischer's MutEx Protocol and Symmetry Reduction



The EU IST Project AMETIST

Mission:

- Improve timed automata model checking tools
- Investigate the applicability of TA tools, esp. to resource allocation problems
- Link to dedicated tools when appropriate

Approach

- Model as dynamical system with *state space* and well-defined *dynamics*: model generates behavior (the semantics)
- Design activities (verification, synthesis) explore and modify system structure so that behavior is correct, optimal, etc
- Do not let modeling suffer from tools

Timed automaton model as mathematical carrier

A Car Periphery Supervision System from Bosch

Presentation by Biniam

A Controller for a Wafer Scanner from ASML

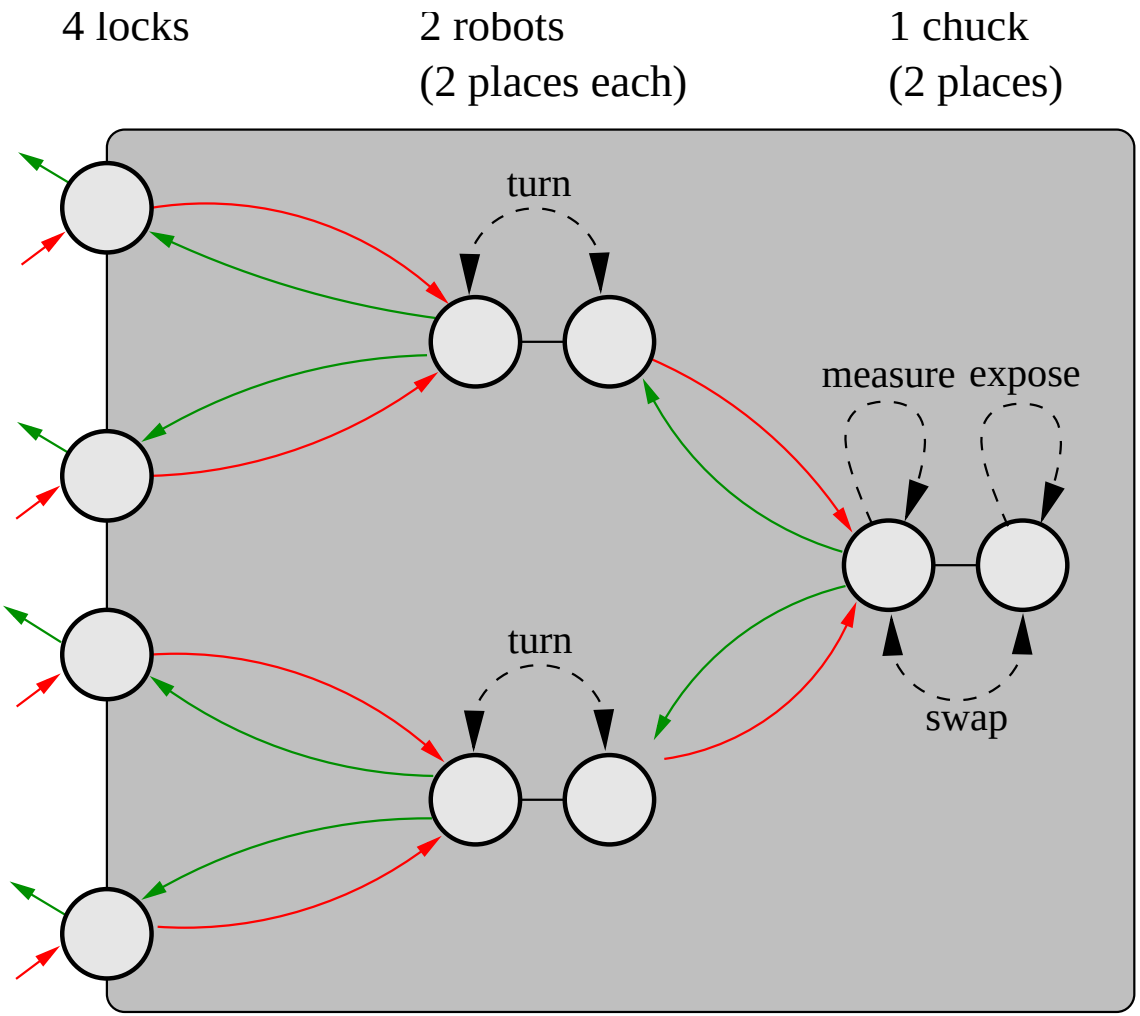
ASML builds wafer scanners

- Very complex lithographic machines used in the semiconductor manufacturing process
  - Machine is regarded as Task-Resource system (flexibility)
  - Scheduling in real-time (many things can go wrong)
  - Throughput is one of the main performance characteristics
  - Deadlock should be avoided at all costs

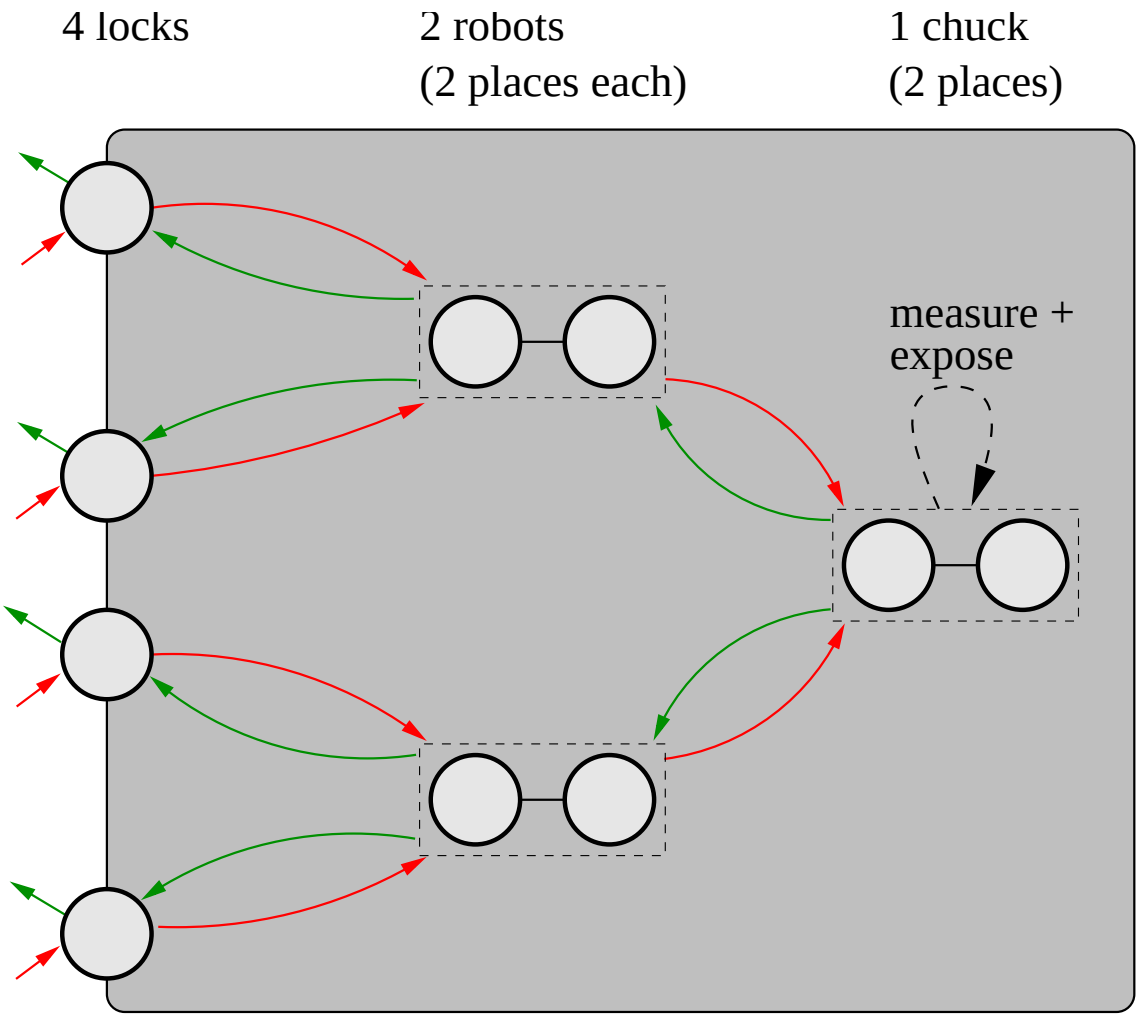
What is this case-study about?

- Material flow in Extreme Ultra Violet (EUV) machine
  - Compute a (least restrictive) deadlock avoidance policy
  - Compute schedules (optimal wrt throughput)

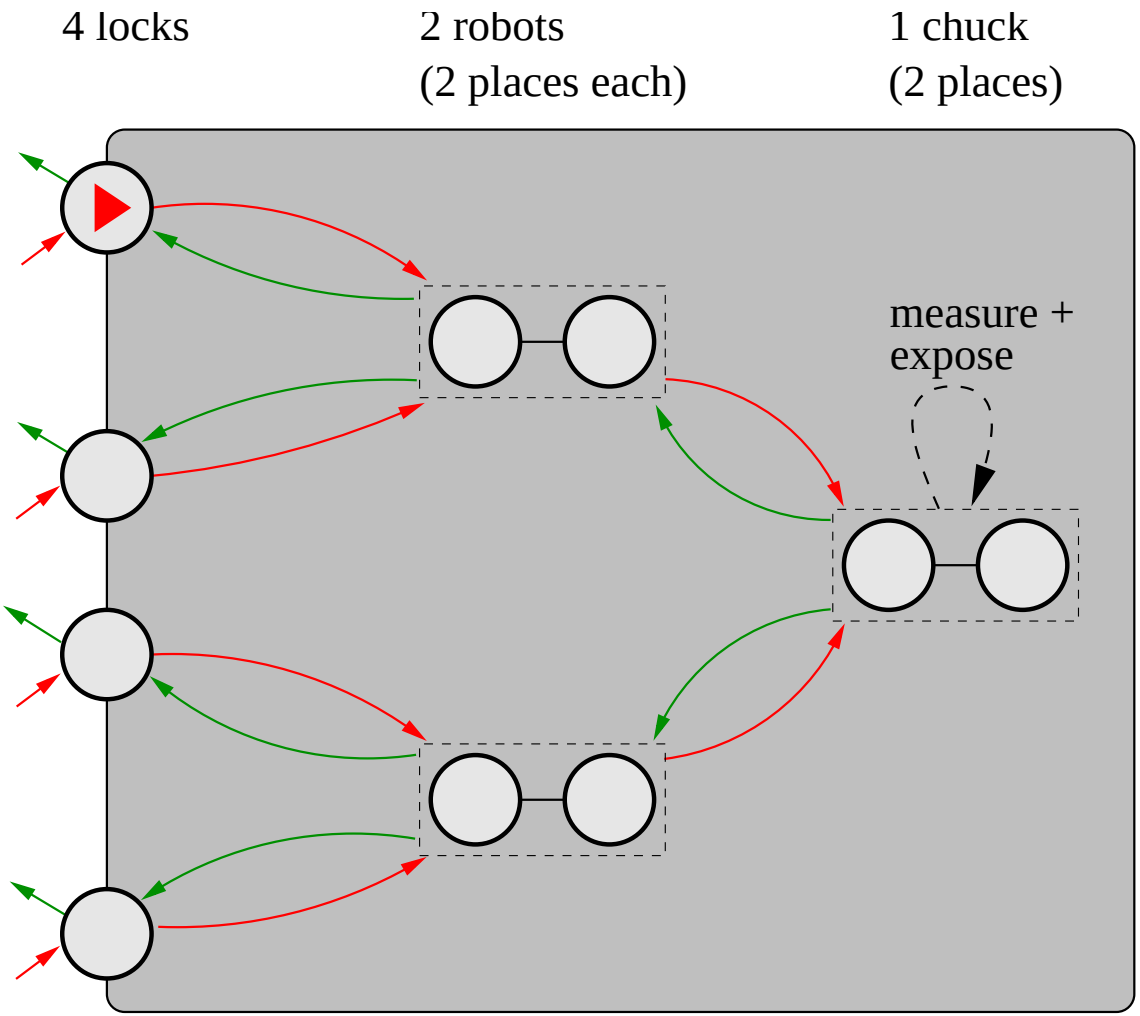
Material flow in EUV machine



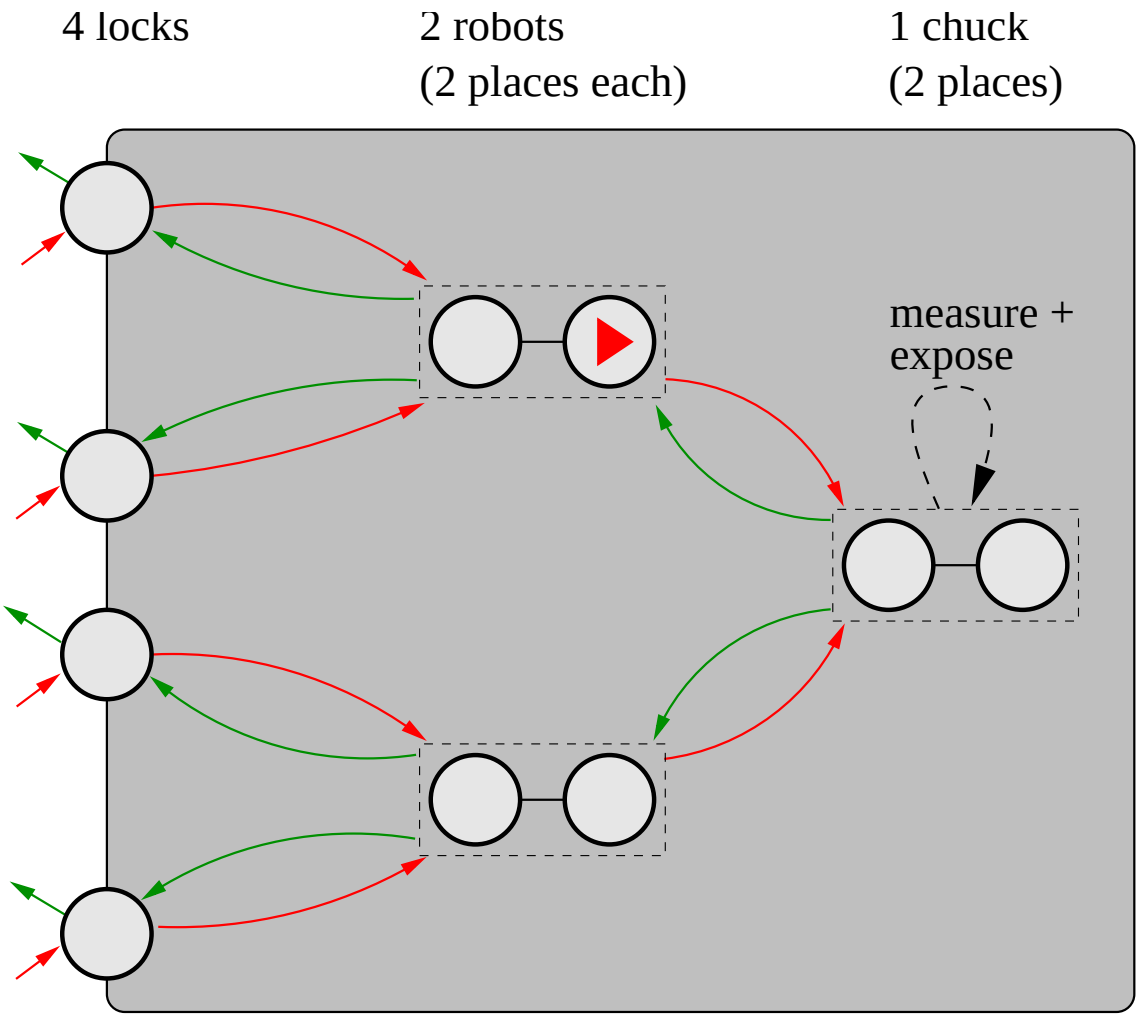
Material flow in EUV machine



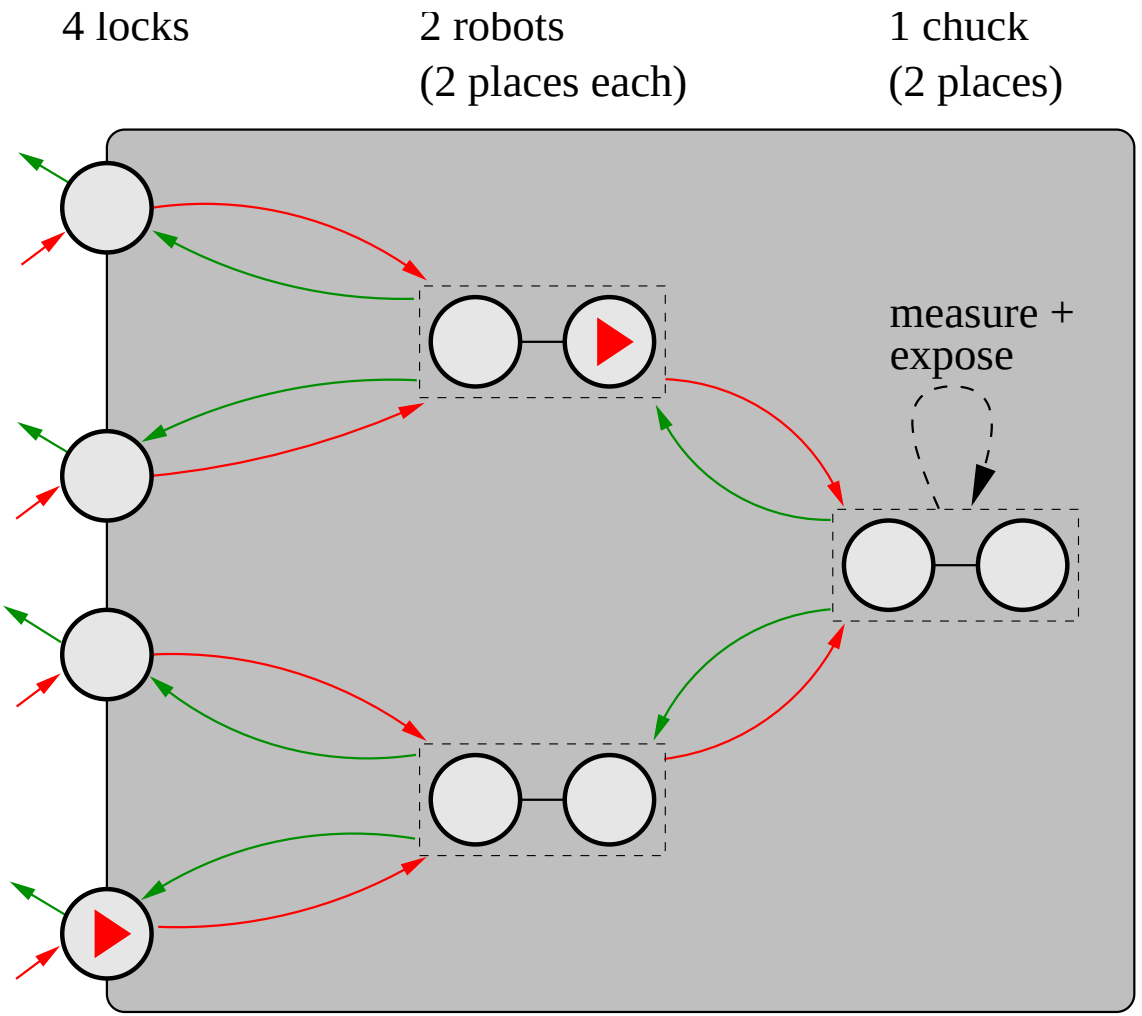
Material flow in EUV machine



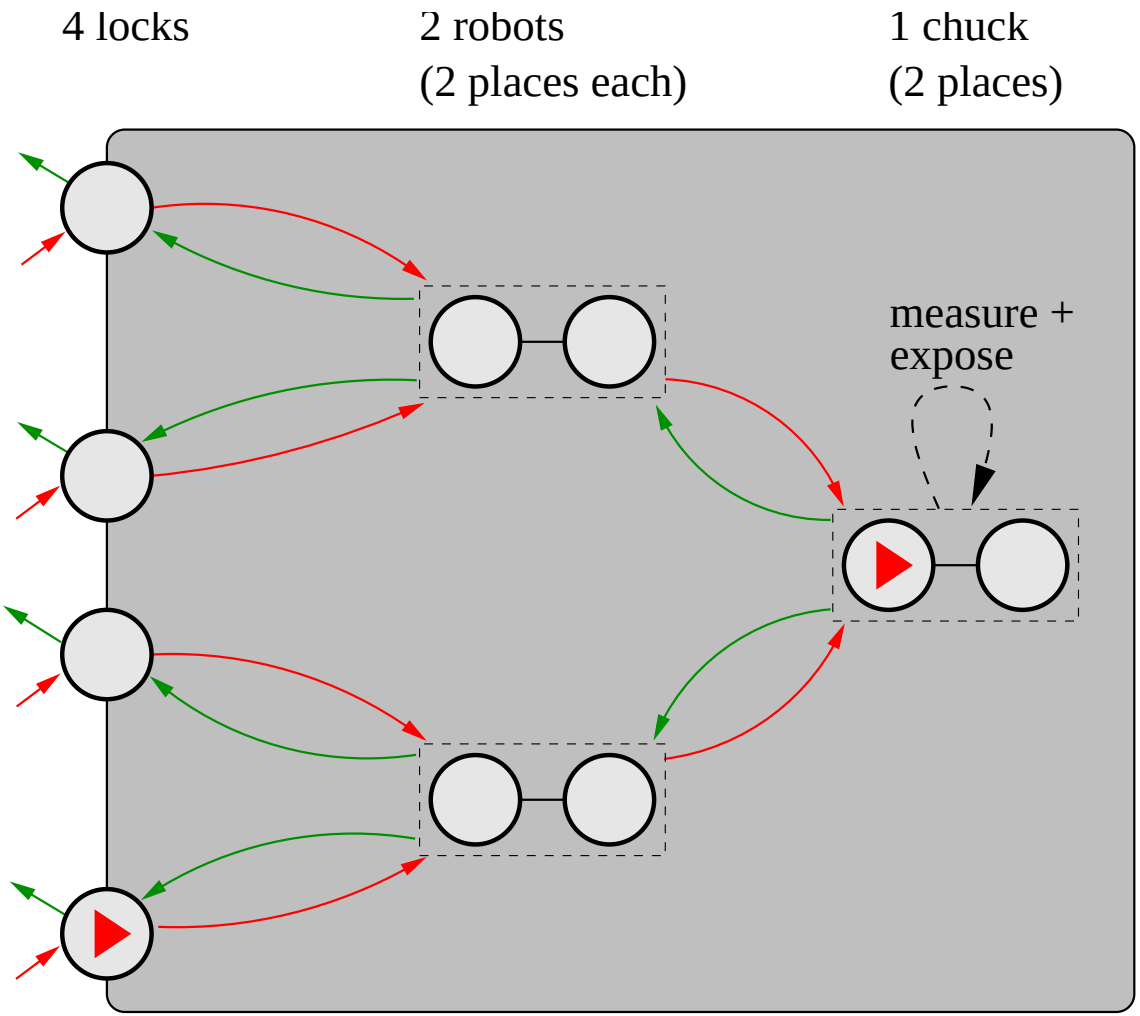
Material flow in EUV machine



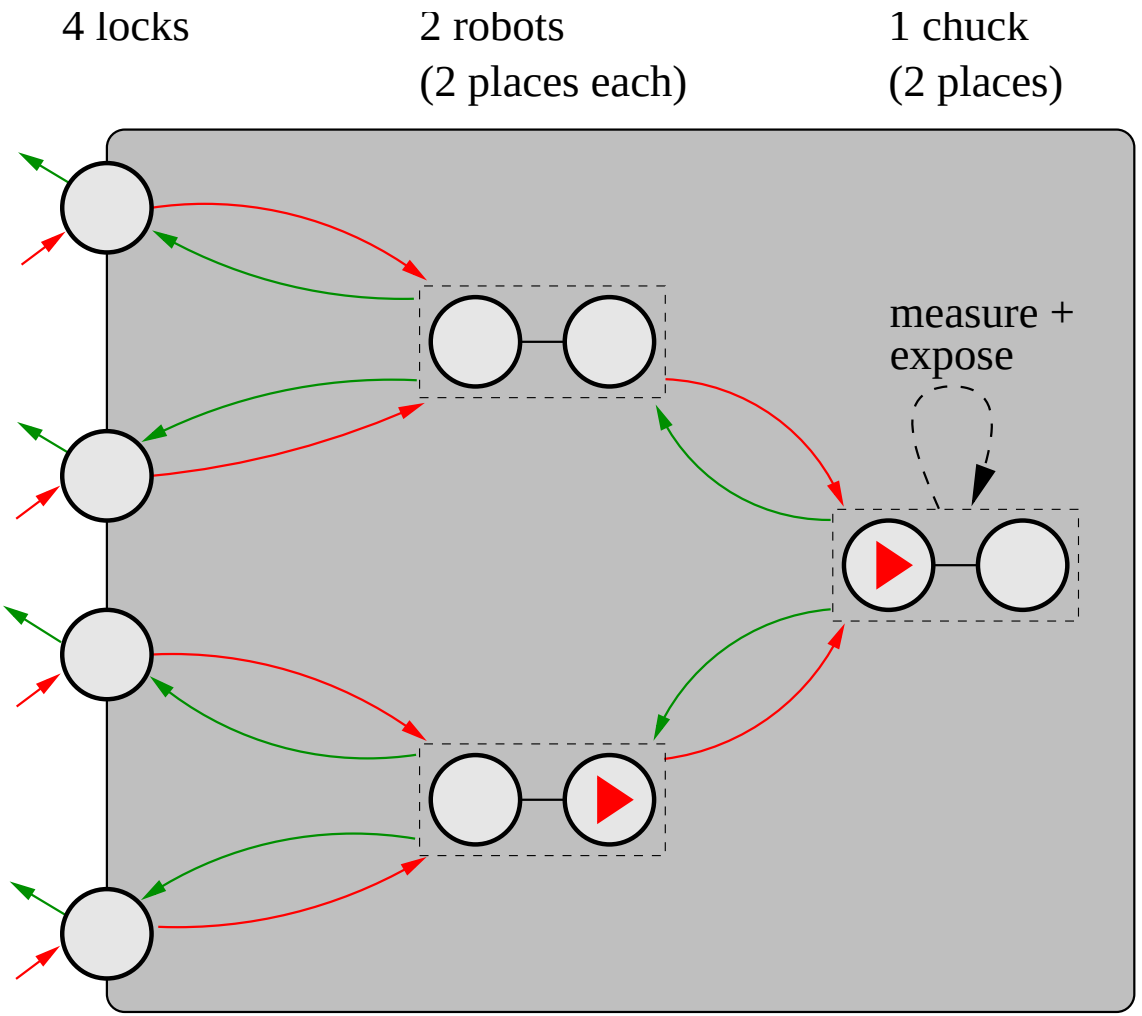
Material flow in EUV machine



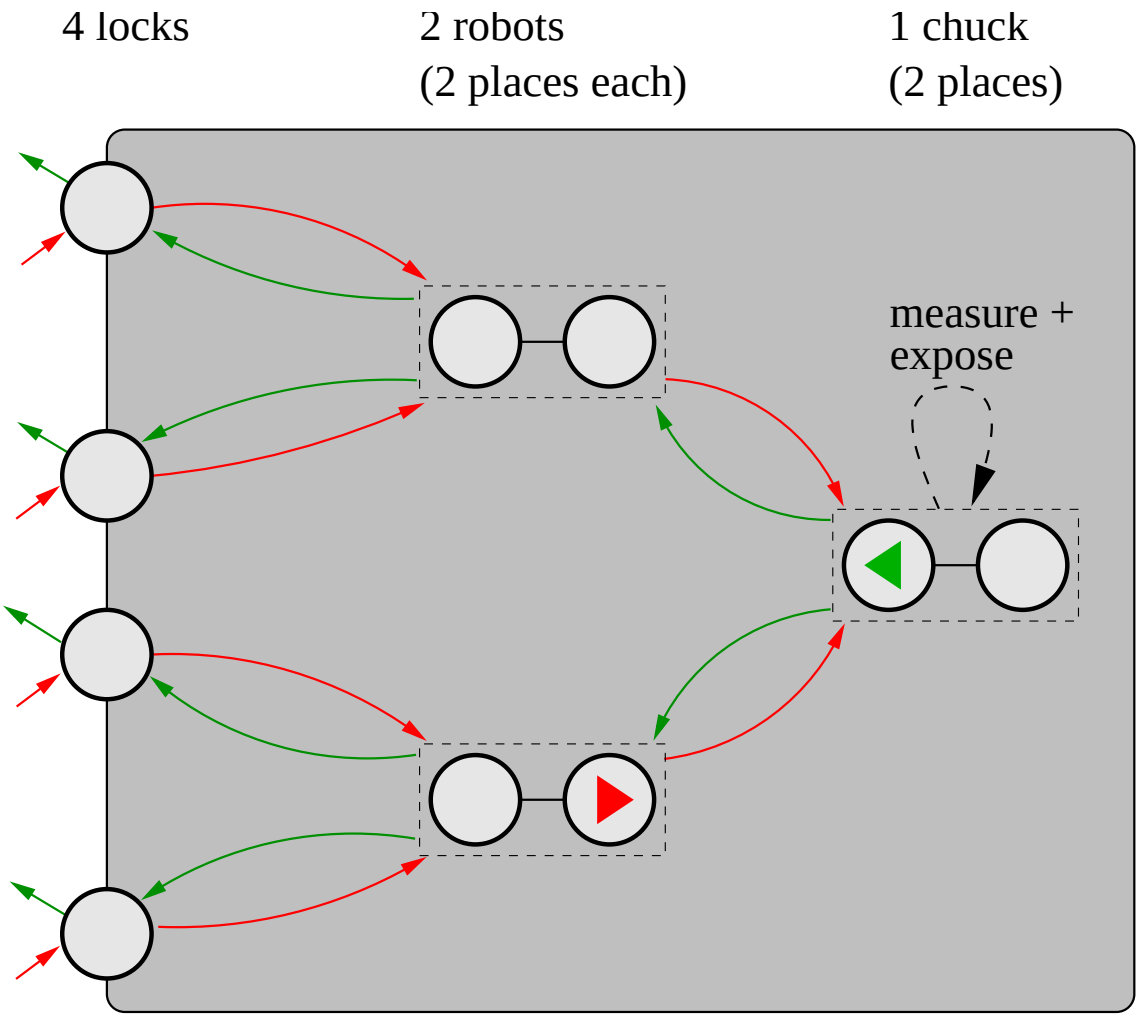
Material flow in EUV machine



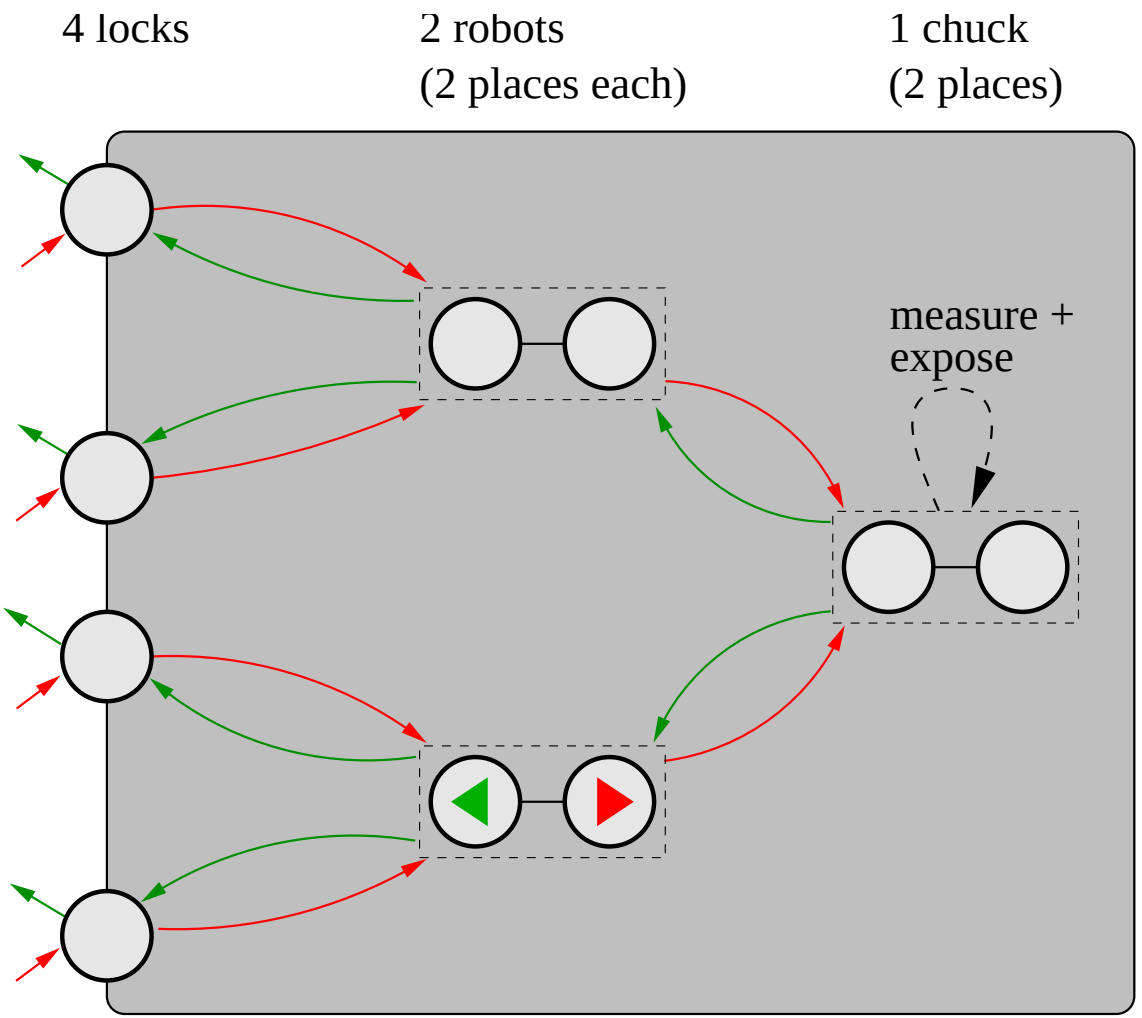
Material flow in EUV machine



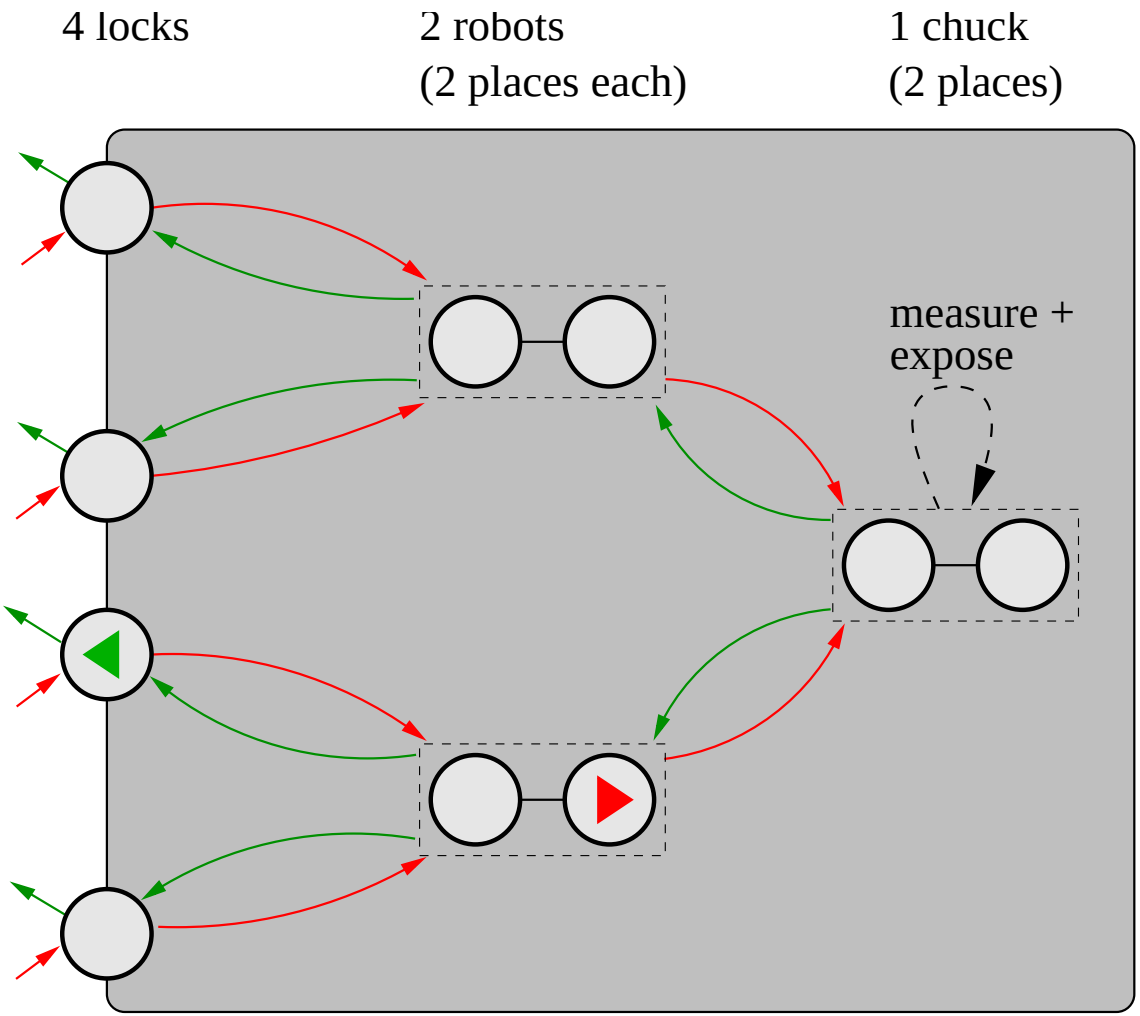
Material flow in EUV machine



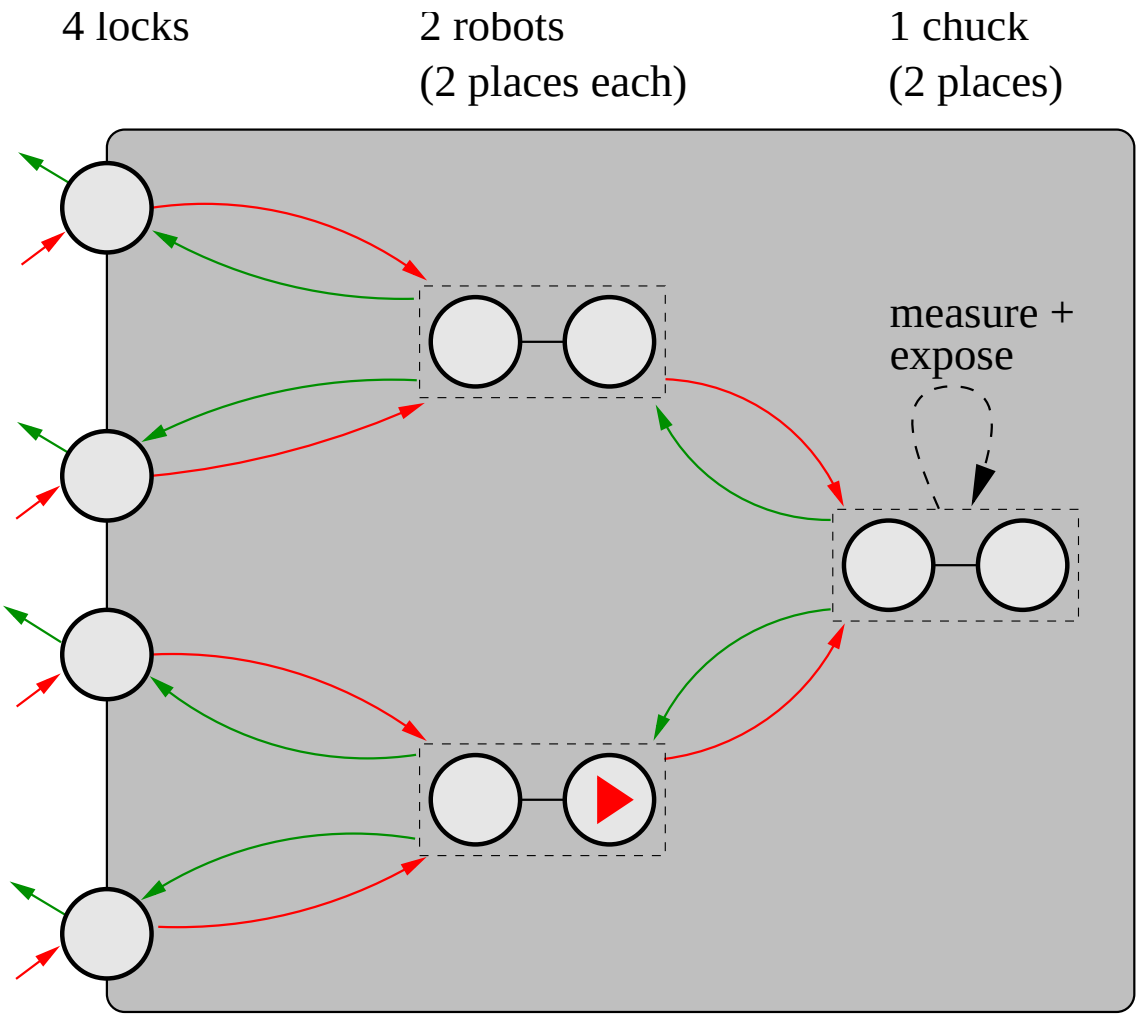
Material flow in EUV machine



Material flow in EUV machine

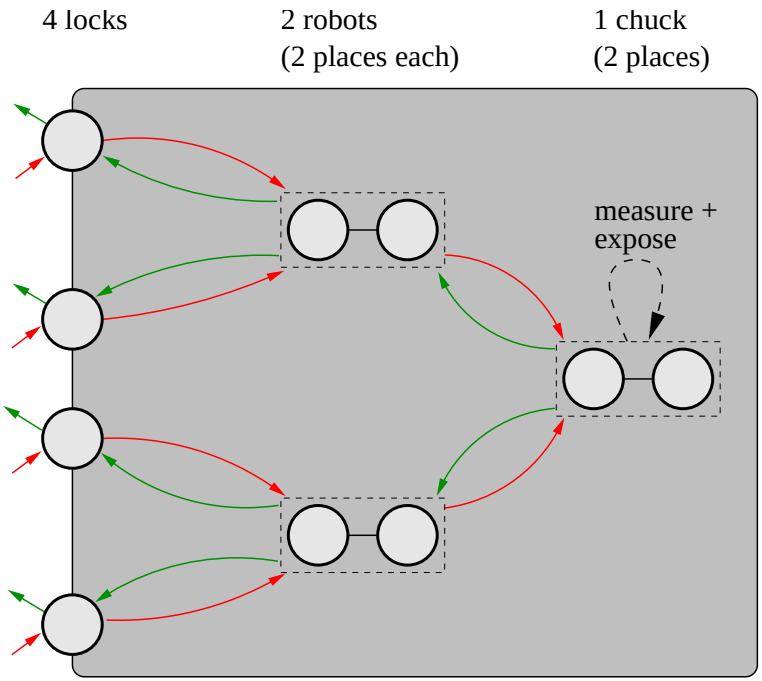


Material flow in EUV machine



Straightforward modeling using SMV model checker

- Every place is modeled by a state variable which can be empty (e), red (r), or green (g)
- Every pair of arrows is modeled by an asynchronous process



```
module main ()
{
  -- the places in the machine:
  l : array 0..3 of {e,r,g};
  c : array 0..1 of {e,r,g};
  rb: array 0..1 of
      array 0..1 of {e,r,g};

  -- initialization:
  for (i=0; i<4; i=i+1)
    init(l[i]):=e;
  for (i=0; i<2; i=i+1)
    for (j=0; j<2; j=j+1)
      init(rb[i][j]):=e;
  for (i=0; i<2; i=i+1)
    init(c[i]):=e;

  -- system dynamics:
  for (i=0; i<4; i=i+1)
    t2l[i]: process entry_exit(l[i]);

  for (i=0; i<4; i=i+1)
    for (j=0; j<2; j=j+1)
      l2r[i][j]: process move(l[i],rb[(i<2?0:1)][j]);

  for (i=0; i<2; i=i+1)
    for (j=0; j<2; j=j+1)
      for (k=0; k<2; k=k+1)
        r2c[i][j][k]: process move(rb[i][j],c[k]);

  for (i=0; i<2; i=i+1)
    exp[i]: process expose(c[i]);
}
```

```
module entry_exit (p)
{
   if (p=e)
     next(p):=r;
   else if (p=g)
     next(p):=e;
}
```

```
module move (lft,rgt)
{
   if (lft=r && rgt=e)
   {
      next(lft):=e;
      next(rgt):=r;
   }
   else if (lft=e && rgt=g)
   {
      next(lft):=g;
      next(rgt):=e;
   }
}
```

```
module expose (p)
{
   if (p=r)
       next(p):=g;
}
```

Handling Deadlock

Three ways of handling deadlock

- Deadlock prevention: restrict system such that deadlock is a priori impossible
- Deadlock detection: detect and resolve deadlocks at runtime
- Deadlock avoidance: dynamically choose control actions to avoid deadlock

Deadlock avoidance: keep the system in a set of safe states (Dijkstra, 1965)

- What is deadlock and what are safe states?
- How to express deadlock and safety in CTL?
- How to characterize the set of safe states?

Informal Definitions

Deadlock

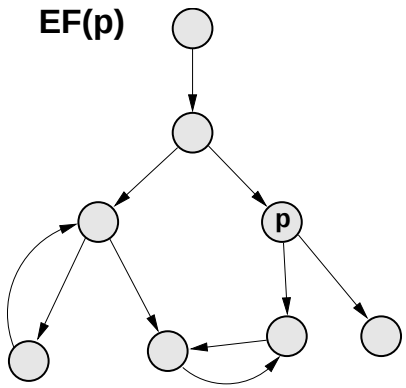
- A state is *deadlocked* iff there is a circular wait
- In our model, a state is a deadlock iff there exists a wafer that cannot move anymore

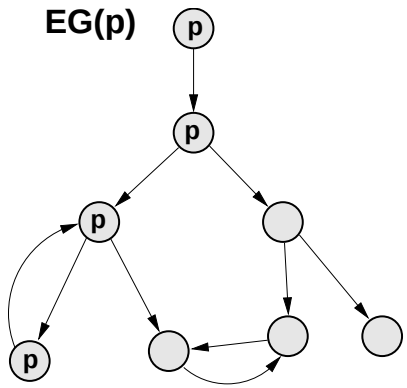
Safety

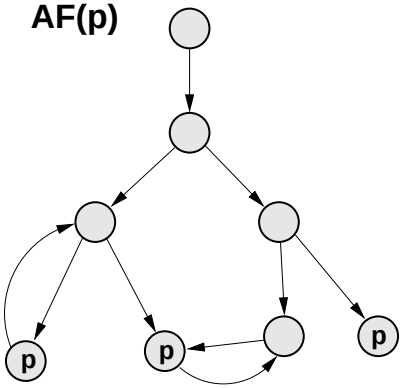
- A state is *safe* iff all processes (wafers in our case) can be run to completion.
- In our model, a wafer is run to completion when it exits the machine

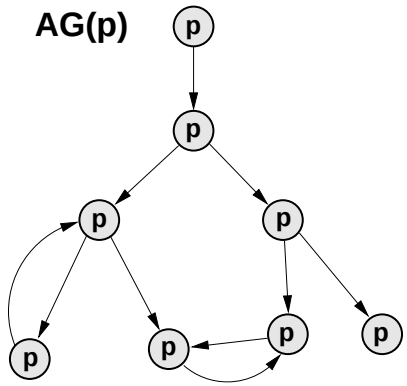
CTL Interlude

SMV builds a transition system over which it interprets CTL









CTL Definitions

$$\textbf{deadlock} \equiv \bigvee_{p \in P} \textbf{AG}(p \text{ is not empty})$$

$$\textbf{safe} \equiv \textbf{EF} \left(\bigwedge_{p \in P} (p \text{ is empty}) \right)$$

where P is the set of places of the EUV machine

Note: **deadlock** $\rightarrow$ $\neg$**safe** but in general not: $\neg$**safe** $\rightarrow$ **deadlock**

Avoiding Deadlock

What is the connection between safe and deadlock states?

- We want to show that safe states really are safe, ie, it is always possible to avoid deadlock
- Furthermore, the set of safe states is the largest set from which deadlock can always be avoided

$$s_{\text{init}} \models \mathbf{AG}\,(\mathbf{safe} \iff \mathbf{EG}(\neg \mathbf{deadlock}))$$

Least restrictive deadlock avoidance policy for EUV machine

- Keep it within the set of safe states!

Characterizing the Safe States

Iterative approach

set $C = \mathit{true}$

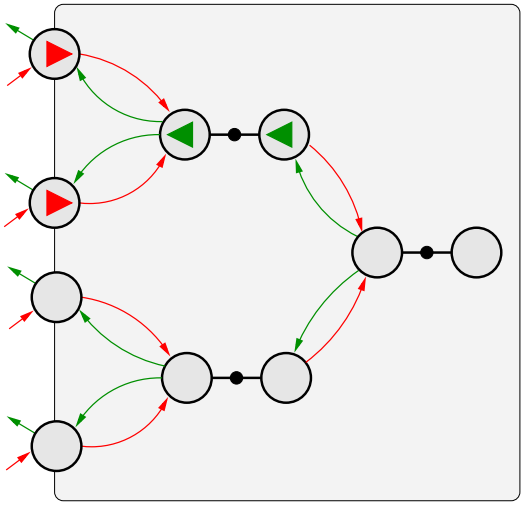
while $s_{\text{init}} \not\models \mathbf{AG}(\mathbf{safe} \iff C)$ do:

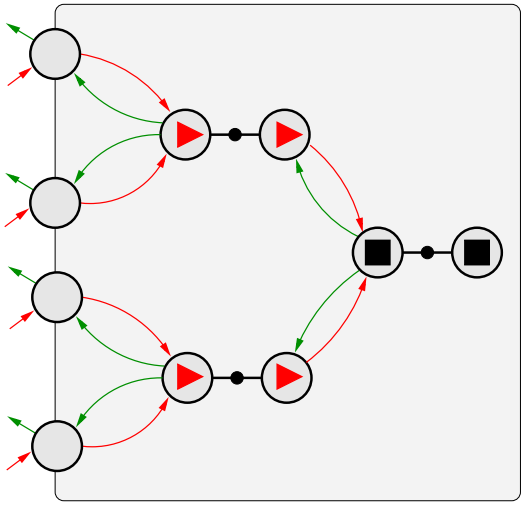
Update C to exclude counterexample (involves thinking)

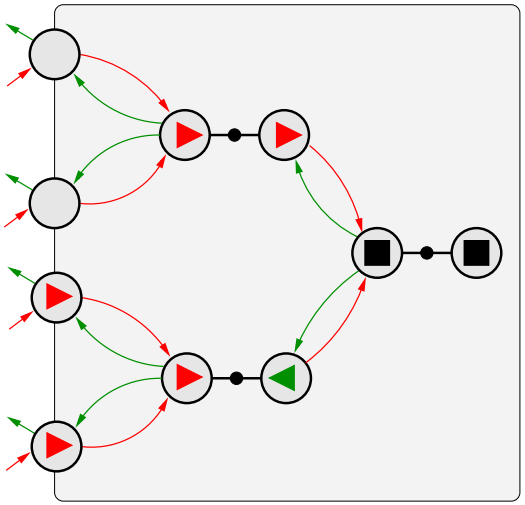
This case: 4 iterations to get 4 unsafe situations (mod symmetry)

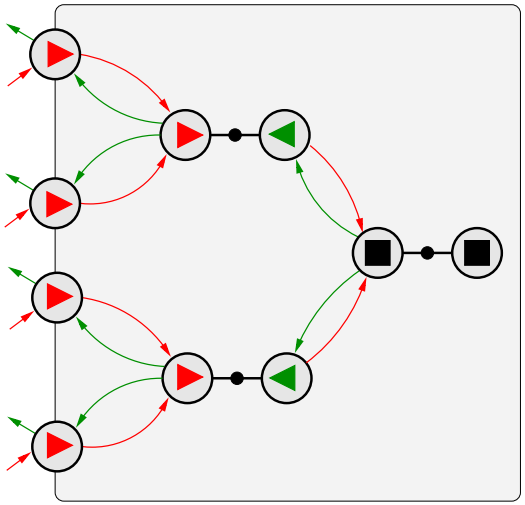
Note

- Creative step not needed: SMV internally builds a BDD representation of the set of safe states if you ask whether $s_{\text{init}} \models \mathbf{safe}$
- However, iterative approach gives *insight* (a BDD does not)









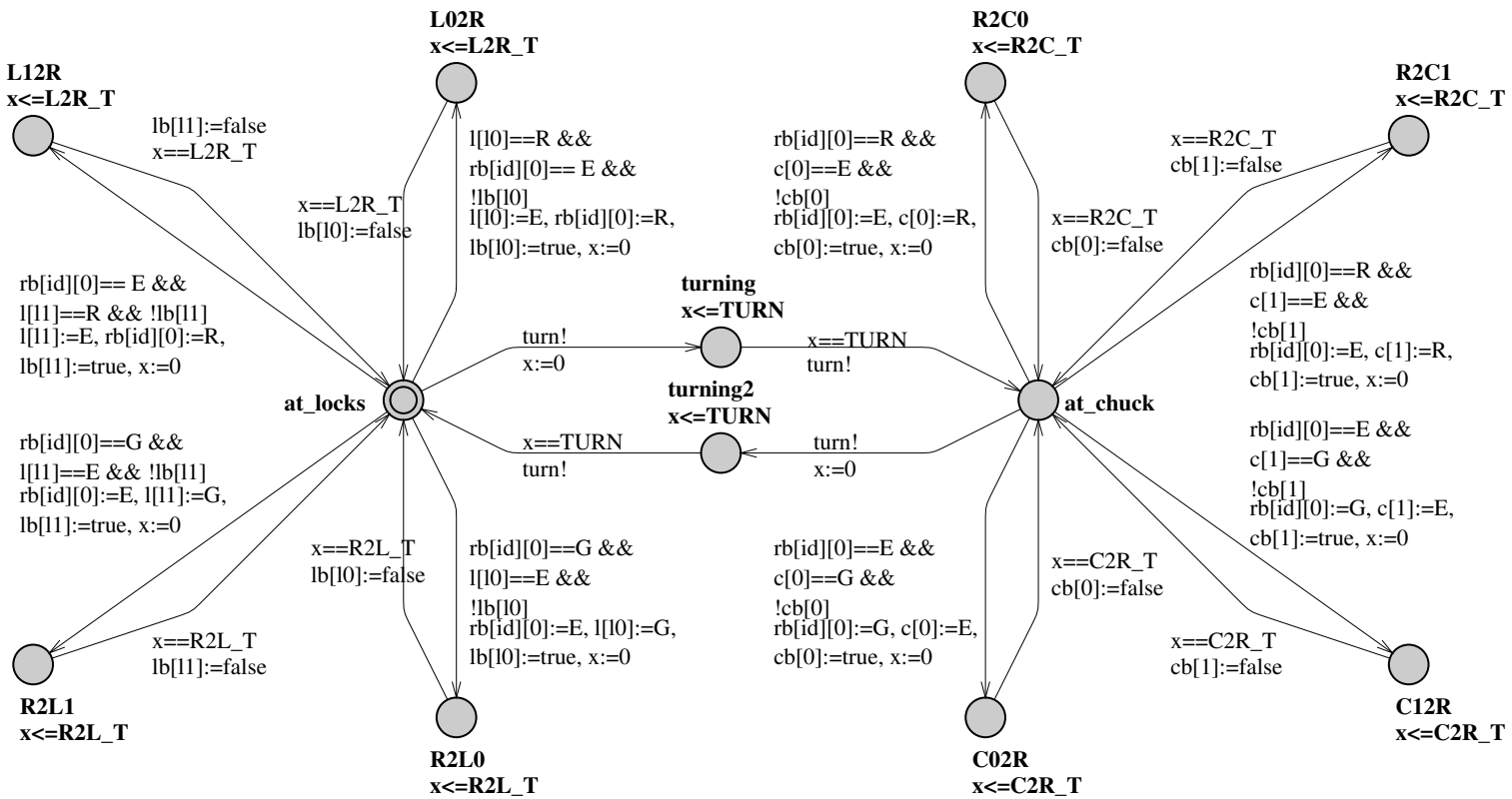
Predicate C that Characterizes Set of Safe States

```
~( (l[0]=r & l[1]=r & rb[0][0]=g & rb[0][1]=g)
   |
   (l[2]=r & l[3]=r & rb[1][0]=g & rb[1][1]=g)
   |
   (~c[0]=e & ~c[1]=e & rb[0][0]=r & rb[0][1]=r & rb[1][0]=r & rb[1][1]=r)
   |
   (~c[0]=e & ~c[1]=e & rb[0][0]=r & rb[0][1]=r &
   ((rb[1][0]=r & rb[1][1]=g) | (rb[1][0]=g & rb[1][1]=r)) & l[2]=r & l[3]=r)
   |
   (~c[0]=e & ~c[1]=e & rb[1][0]=r & rb[1][1]=r &
   ((rb[0][0]=r & rb[0][1]=g) | (rb[0][0]=g & rb[0][1]=r)) & l[0]=r & l[1]=r)
   |
   (~c[0]=e & ~c[1]=e & ((rb[0][0]=r & rb[0][1]=g) | (rb[0][0]=g & rb[0][1]=r)) &
   ((rb[1][0]=r & rb[1][1]=g) | (rb[1][0]=g & rb[1][1]=r)) &
   l[0]=r & l[1]=r & l[2]=r & l[3]=r)
)
```

Refinement of the SMV model

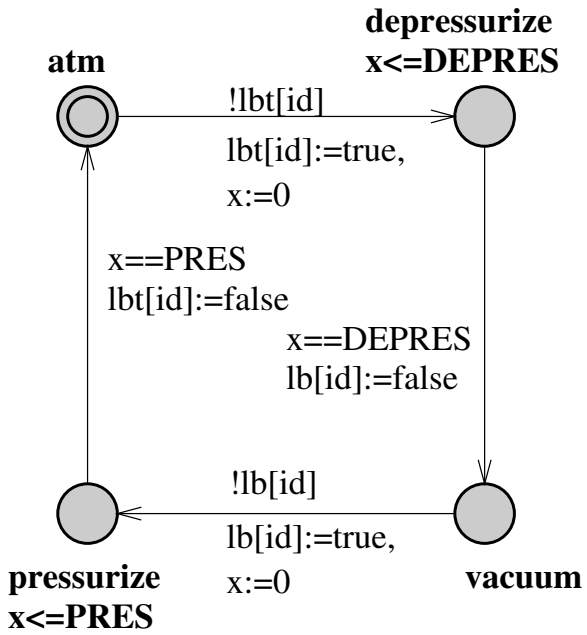
Refinement of the SMV model

Add detail and timing



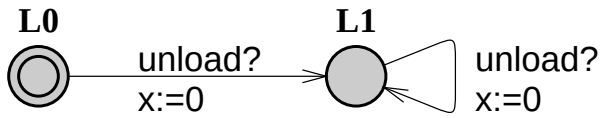
Refinement of the SMV model

Add constraints (locks for instance; also mutual exclusion)



Refinement of the SMV model

Add Observer process (for throughput optimization)

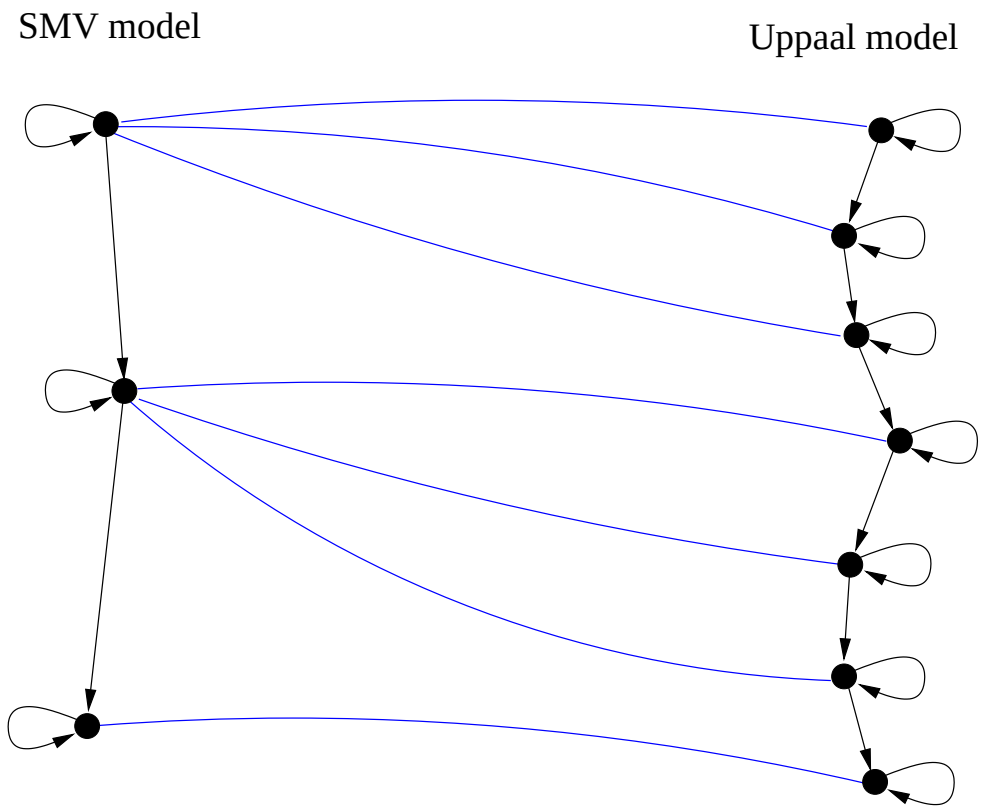


Ask Uppaal whether

$$s_{\text{init}} \models \mathbf{EG} \left(\begin{array}{c} \mathbf{Observer.L0} \Longrightarrow \mathbf{Observer.x} \leq H \\ \wedge \\ \mathbf{Observer.L1} \Longrightarrow \mathbf{Observer.x} \leq S \end{array} \right)$$

Relation with SMV Model

There is a *stuttering bisimulation* R between the Uppaal model and the SMV model Thus, CTL\X formulas are preserved (Browne, Clarke & Grümberg, 1988)



Adding Heuristics

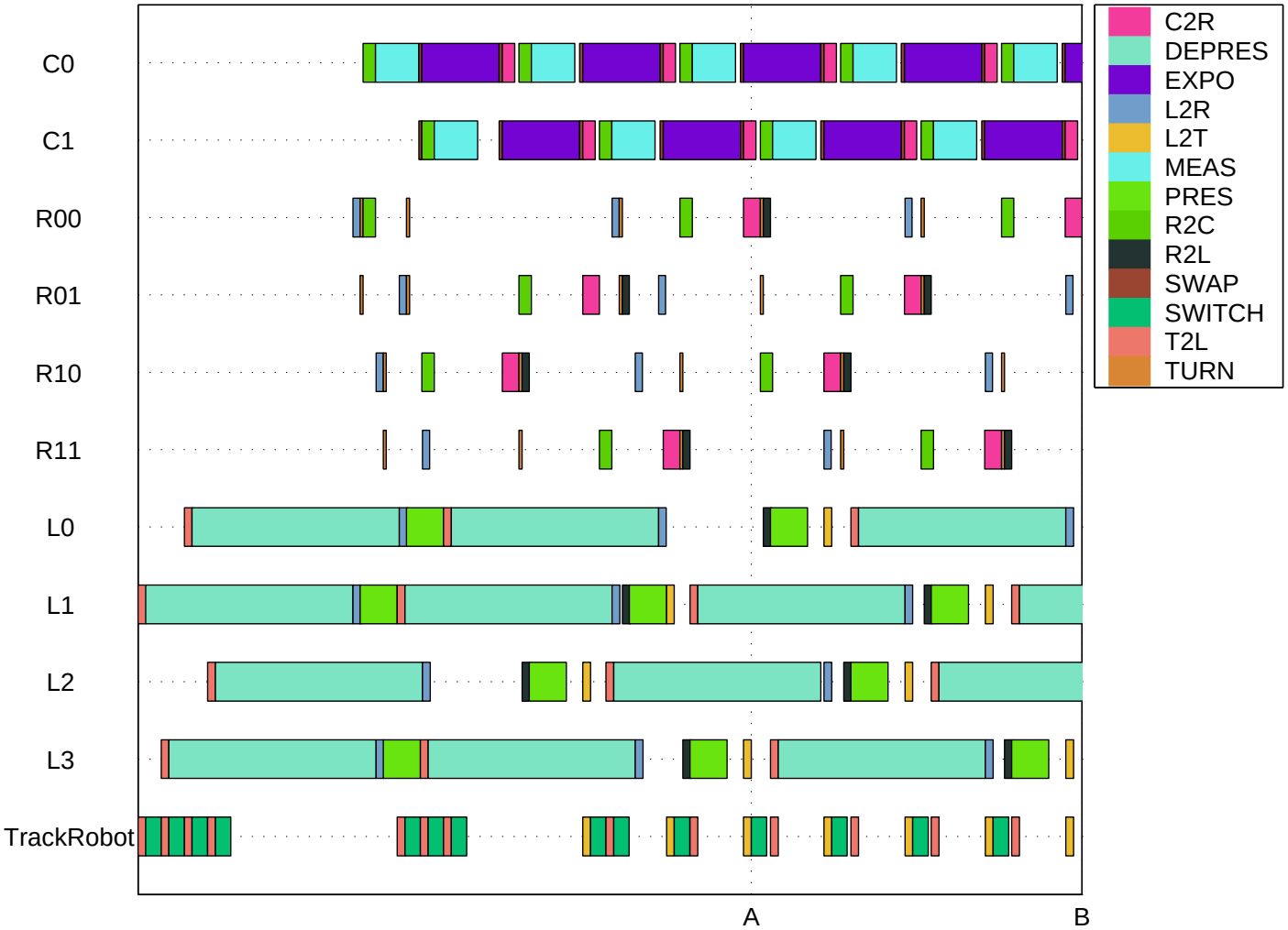
State space is too large

- Locks can depressurize or pressurize (almost) any time
- Internal robots can turn (almost) any time
- Chuck can swap (almost) any time
- Large differences in time scale: 670 (lock depres) vs 10 (turn)

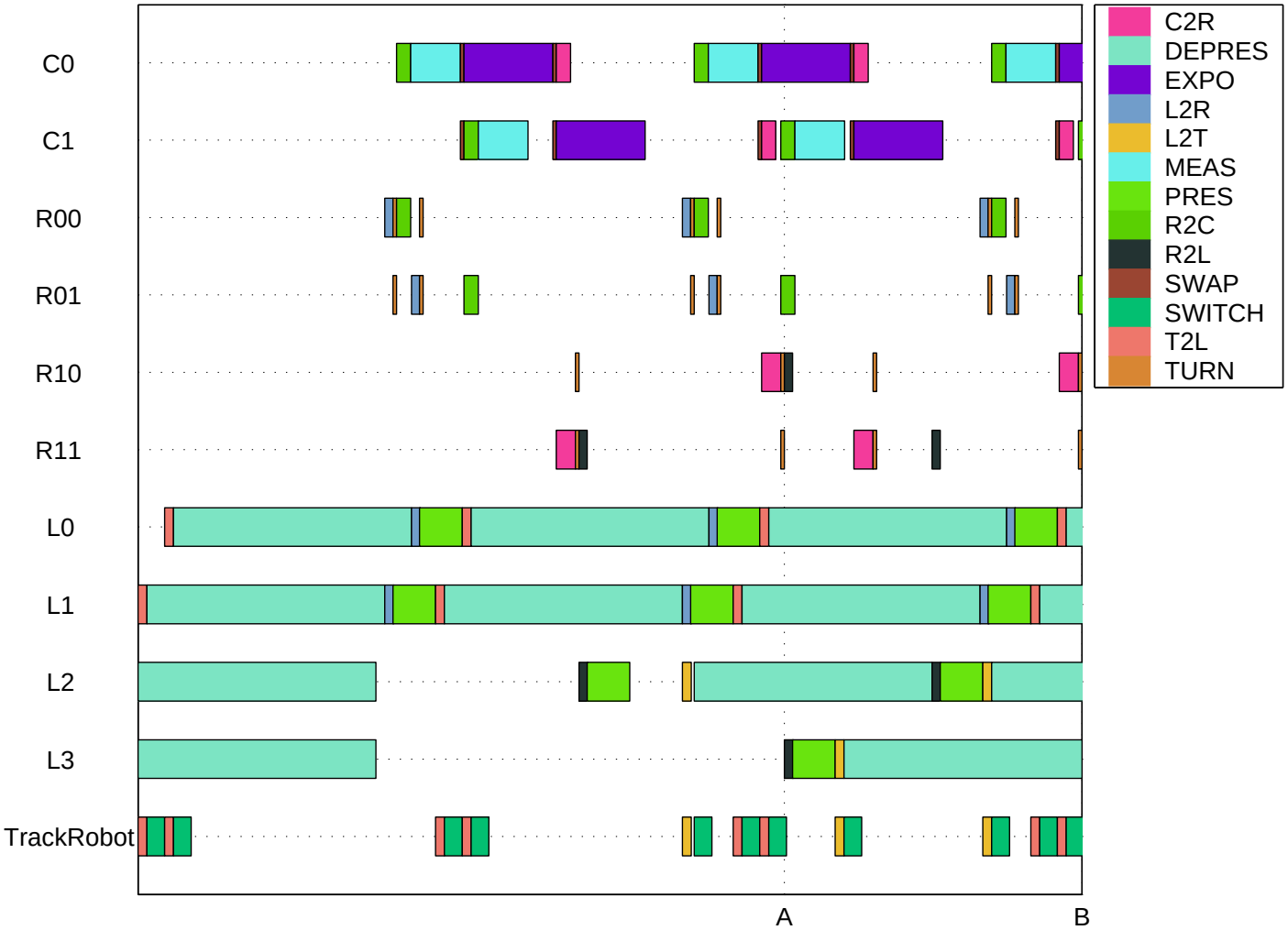
Solutions

- Avoid unsafe material configurations
- Avoid useless transitions (turns, swaps, etc)
- Make some transitions greedy/urgent

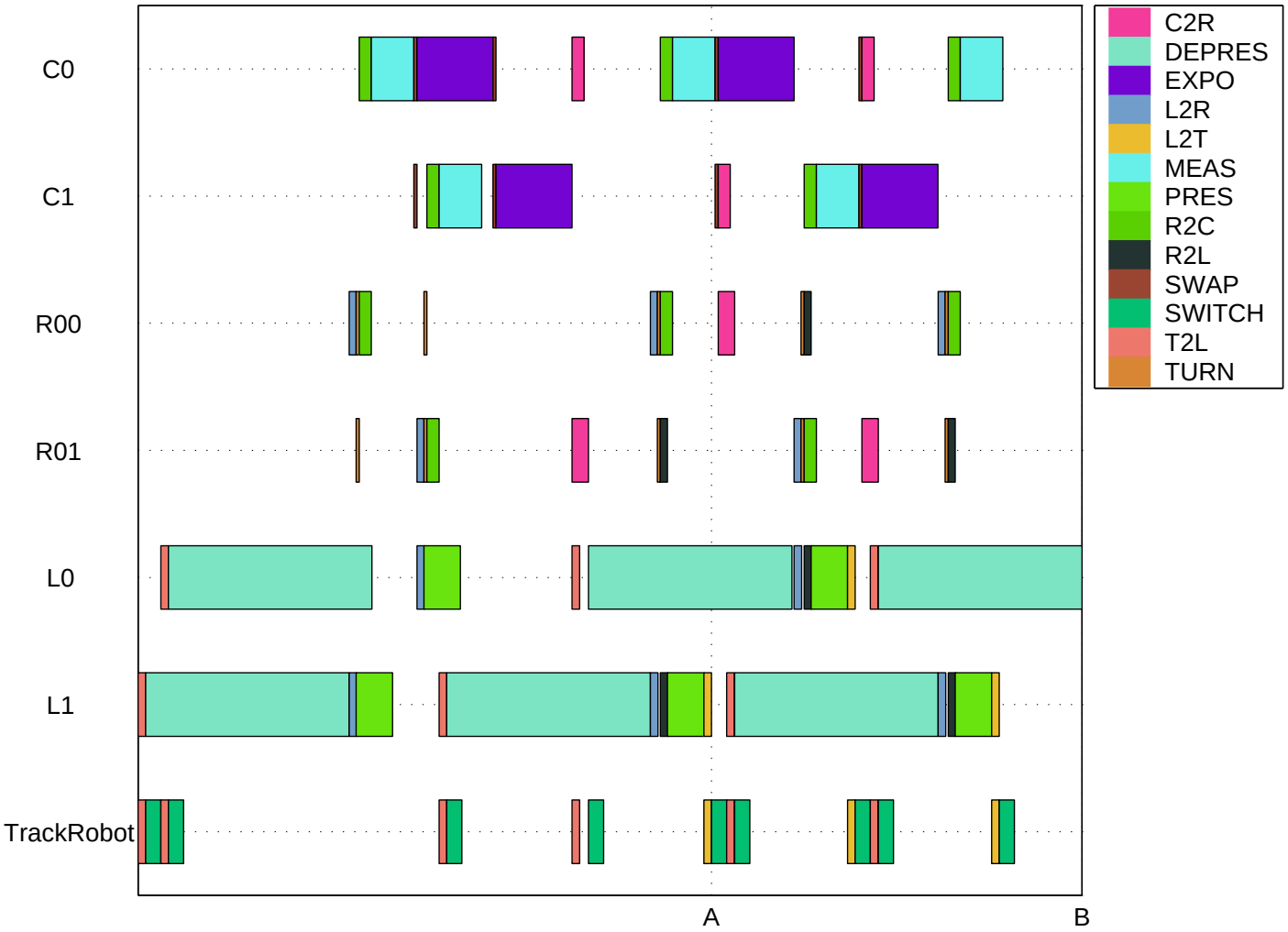
Optimal Schedule



Optimal Schedule without Crossing Wafer Paths



Schedule for 2 locks and 1 internal robot



Conclusions ASML Case Study

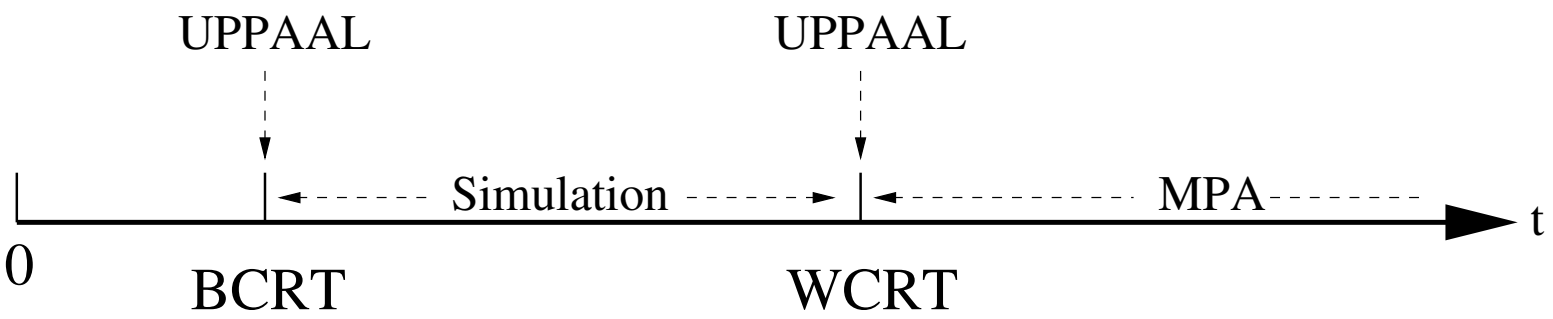
- *Short* and *exact* characterization of safe states (either by iterative process or by extracting a BDD from SMV)
- Synthesis of a schedule that optimizes throughput; analysis of an alternative configuration and control policy
- We have adjusted abstraction level for different goals and proved soundness
- It took us approx. 2 weeks to build the models and to obtain our results
- Our work confirms once more that formal modeling and analysis may help to improve the design process; our work is referred to in a patent application filed by ASML

Scheduling Laquer Production (Behrmann/Brinksma/Hendriks/Mader, '05)

- Case study proposed by Axxom to Ametist project
- First real challenge for UPPAAL CORA
- Schedules found by Uppaal CORA improve on those found by ORION-Pi tool of Axxom

System Architecture Evaluation Using MPA and Uppaal (Wandeler/Verhoef/Hendriks, WIP, '05)

- Use of MPA and real-time calculus of to assess architectures for embedded systems
- Use of max plus algebra to compute with arrival curves and service curves
- Application: a distributed in-car navigation system
- Response time and sensitivity analysis also done using Uppaal


UPPAAL
UPPAAL
Simulation
MPA
t
0
BCRT
WCRT

A Biphase Mark Protocol (Vaandrager/De Groot, '04)

- Physical level protocol implemented e.g. by Intel
- First formal model by Moore in 1994
- Uppaal model allowed us to quickly analyse many instances of protocol
- General (parametric) correctness verified using PVS
- Combined use of model checker and theorem prover useful!
- Analysis reveals instances with faster bit rates than instances that are commonly implemented

Agreement Protocol of Attiya, Dwork, Lynch & Stockmeyer (work by Hendriks, '05)

- Partially synchronous algorithms can be very difficult for timed automata: one clock per message!
- Still these protocols get within scope of Uppaal
- Modular verification approach
- Key improvement: use of C-like functions in Uppaal
- Uppaal can verify several non trivial instances
- Future improvements: symmetry reduction and sweep line method

The IPv4 Zeroconf Protocol(Zhang/Vaandrager, '03

- Protocol proposed by IETF to enable use of IP for local communication
- Trade of between reliability and response time analyzed using stochastic models
- Modeling and analysis of several scenarios using Uppaal
- We hope that full state space can be explored with Uppaal 4.0
- Include Uppaal models as part of standard?

Conclusions

- Enormous progress in TA technology recently!
- Expect continued progress for at least few more years
- Widespread industrial use?


AMETIST
advanced methods for timed systems