SAL
SILICON AUSTRIA LABS

REAL TIME MODEL-CHECKING AND UPPAAL

FLORIAN LORBER

Bundesministerium Klimaschutz, Umwelt, Energie, Mobilität, Innovation und Technologie
FFG
Das Land Steiermark
SFG
LAND KÄRNTEN
ober österreich
Member of UAR INNOVATION NETWORK

REAL TIME MODEL-CHECKING AND UPPAAL

Florian Lorber
Silicon Austria Labs

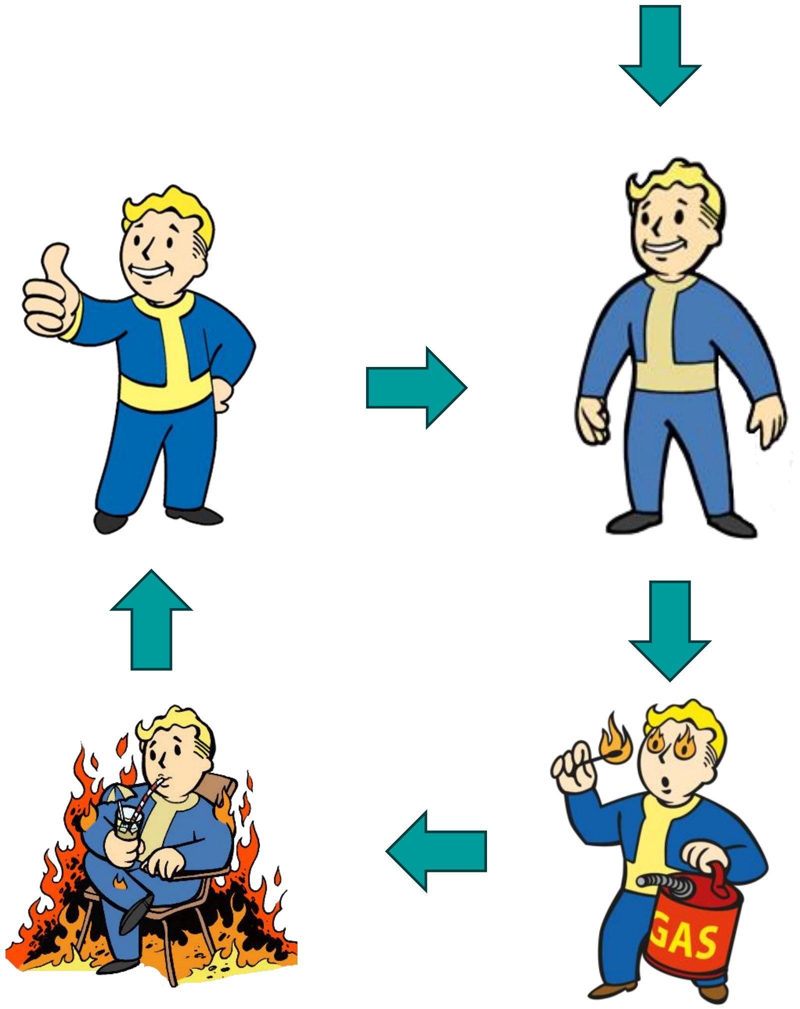
(Slides from Aalborg University)

MODEL CHECKING

- Bring me up to speed about what you know
- Check whether a model fulfills certain properties
  - Does our robot behave like a human?
- What kind of properties can you check?
- What are the two biggest problems with model-checking?

PROPERTIES

- Functional correctness
  - Does the system do what it is supposed to?
- Reachability
  - Is it possible to end up in a certain state?
  - Can the robot set itself on fire?
- Safety
  - Something bad can never happen
  - Will the robot never die?
- Liveness
  - Something good will eventually happen
  - Will the robot recover?
- Fairness
  - In cert. conditions, can an event occur repeatedly
  - Will the robot always recover?



STRENGTHS OF MODEL CHECKING

SAL SILICON AUSTRIA LABS

- General verification technique
- Partial verification is possible
- Covers all traces
- Sound and mathematical foundation
- My highlights about model-checking from papers:
  - “No high degree of expertise needed”
  - “Learning curve is not steep”

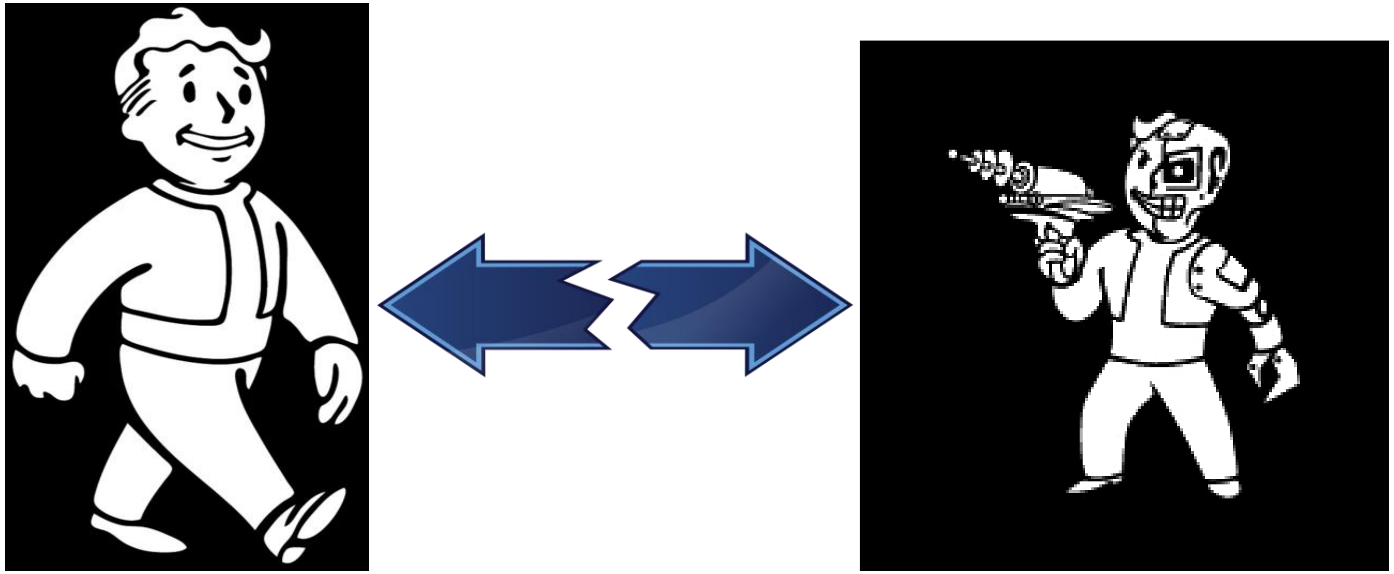
WEAKNESSES

SAL
SILICON AUSTRIA LABS

- Not for data intensive applications
- Decidability issues
- Only the model is verified
- State space explosion

MODELLING GAP

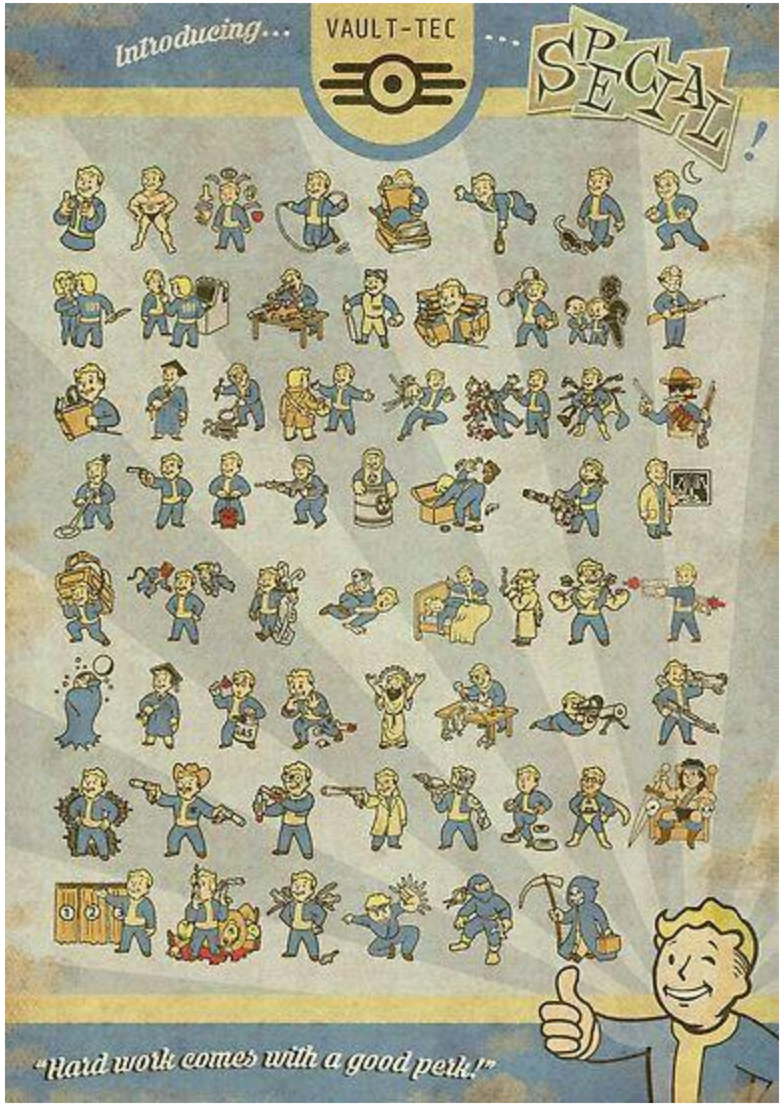
- Any verification using model-based techniques is only as good as the model of the system.



STATE SPACE EXPLOSION

SAL SILICON AUSTRIA LABS

- Too many states to complete the verification
- Concurrency, Data Variables, Complexity, …
- Consequences:
  - Memory consumption
  - Computation time
  - Scalability



REAL TIME SYSTEMS

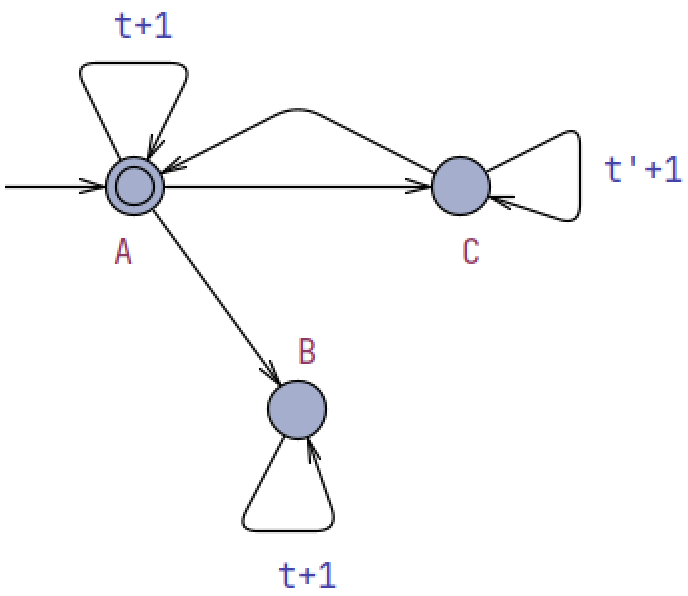
- Systems with Soft and Hard Deadlines

- Soft Deadline:
  - Some degree for flexibility
  - Missed deadline leads to degraded performance

- Hard Deadline:
  - No exceptions
  - Missed deadline leads to catastrophic failures
  - E.g. Pacemaker, traffic control, etc.

REAL TIME SYSTEMS

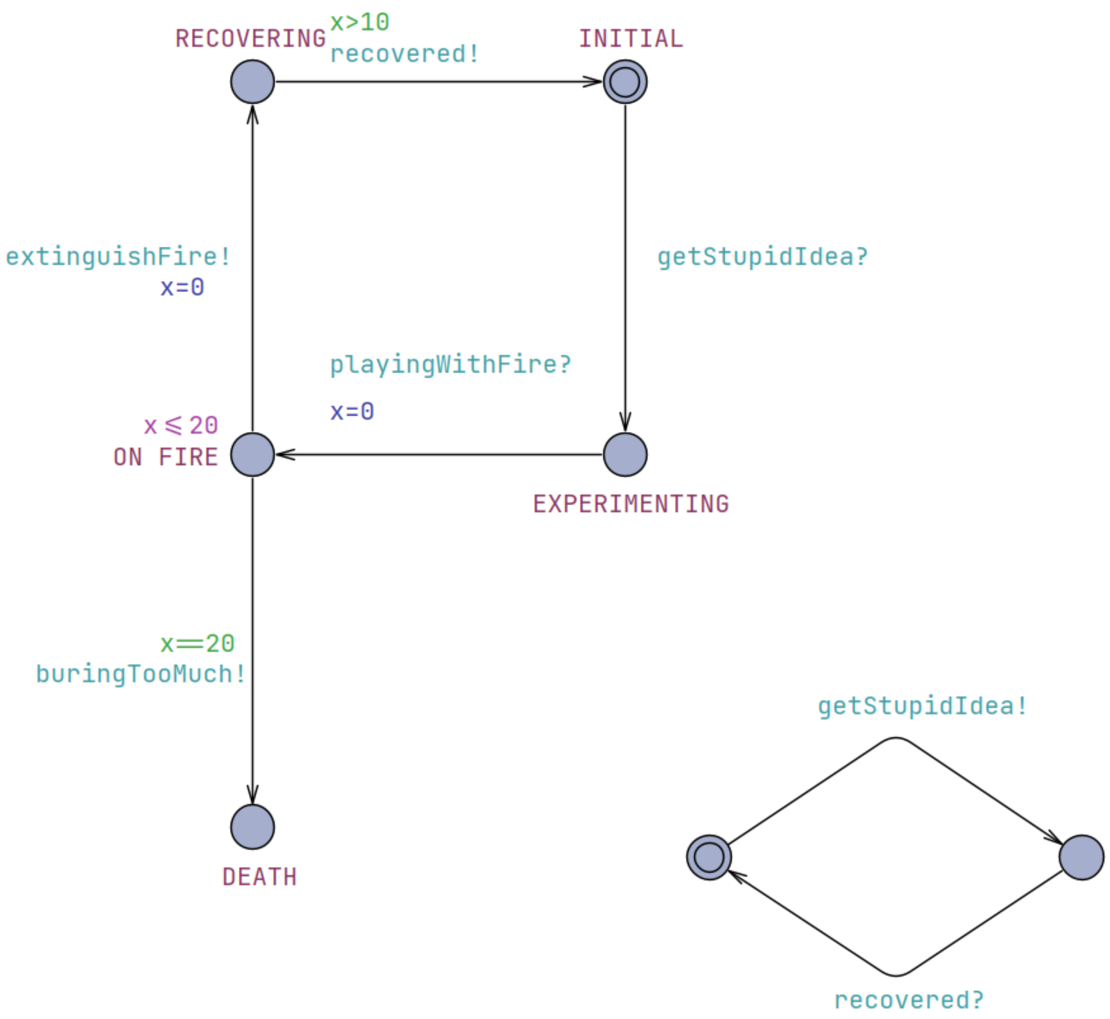
- Why does time warrant for each own lecture?
- I want to enter state B 5 seconds after I enter state A
  - How many variables needed to keep track of the timing?
  - What if time I spend in state C does not count to the 5 seconds?
- Consider discrete time
  - Time can count up in each state
- Now imagine the state space with real variables



TIMED AUTOMATA

SAL SILICON AUSTRIA LABS

- Extended final state machine
  - Labeled transitions
  - Clock variables
  - Measures continuous time
- Time progresses in locations
  - There might be a time limit
- Actions are instantaneous
  - Might only be enabled in certain times
  - Can reset clocks
- Networks of timed automata
  - ?!
- Statespace?



TIMED AUTOMATA - FORMAL

- Set of clocks C
  - B(C) is the set of junctions of simple conditions
    - $x \{<, \leq, =, \geq, >\}\ c$
    - $x - y \{<, \leq, =, \geq, >\}\ c$
    - $x, y \in C, c \in \mathbb{N}$
  - Set of clock valuations v
  - Valuations map clocks to real values
  - $v(x) \rightarrow \mathbb{R}$

- Timed Automaton: $\text{TA} = (L, l_0, C, A, E, I)$
  - L: set of locations
  - l_0: initial location
  - C: set of clocks
  - A: set of actions
  - E: set of edges
    - $E \subseteq L \times A \times B(C) \times 2^C \times L$
  - $I : L \rightarrow B(C)$

- Semantics:

- Clock valuations v
  - Map clocks to real values
  - $v(x) \rightarrow \mathbb{R}$
  - $v_0(x) \rightarrow 0\ \forall x \in C$

Definition 2 (Semantics of TA). *Let* (L, l_0, C, A, E, I) *be a timed automaton. The semantics is defined as a labelled transition system* $\langle S, s_0, \rightarrow \rangle$, *where* $S \subseteq L \times \mathbb{R}^C$ *is the set of states,* $s_0 = (l_0, u_0)$ *is the initial state, and* $\rightarrow \subseteq S \times \{\mathbb{R}_{\geq 0} \cup A\} \times S$ *is the transition relation such that:*

- $(l, u) \xrightarrow{d} (l, u + d)$ *if* $\forall d' : 0 \leq d' \leq d \implies u + d' \in I(l)$, *and*
- $(l, u) \xrightarrow{a} (l', u')$ *if there exists* $e = (l, a, g, r, l') \in E$ *s.t.* $u \in g$, $u' = [r \mapsto 0]u$, *and* $u' \in I(l)$,

UPPAAL - OUTLINE

SAL SILICON AUSTRIA LABS

- GUI
- Modelling language
- Simulator
- Formal semantics
- Query language
- Reachability algorithm

SAL SILICON AUSTRIA LABS

GUI - DEMO

Automaton, Location, Edge, Synchronization, Guard, Update, Select, Clocks, Channels, Environment, System Declarations, Simulator

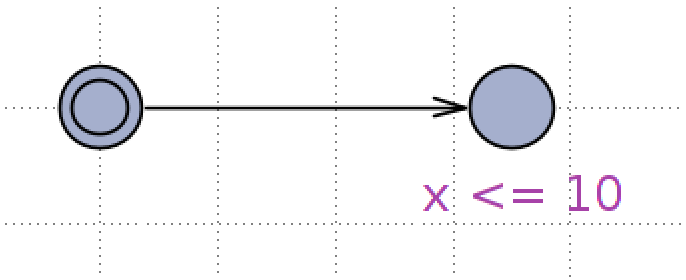
NOTATION

- Location – a place in a single template or process
- State – the state of the complete system including clock valuations and variable values

- Edge – a step between two locations
- Transition – a change of the global state of the system

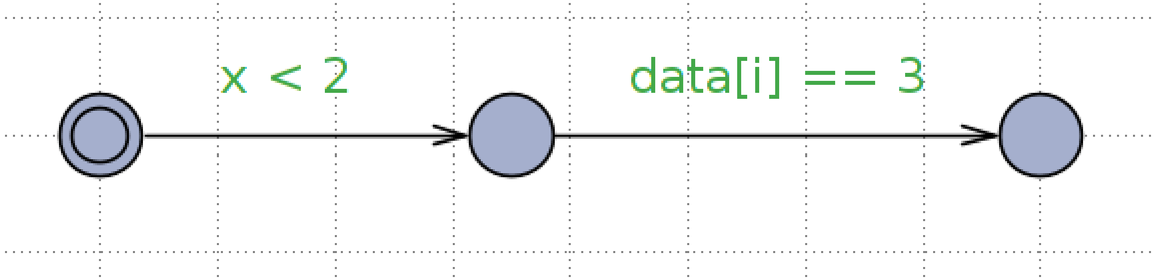
INVARIANT

- Something that must be true in a given location
  - If it is not true we must leave or else we deadlock
  - If it is not true we cannot enter the location



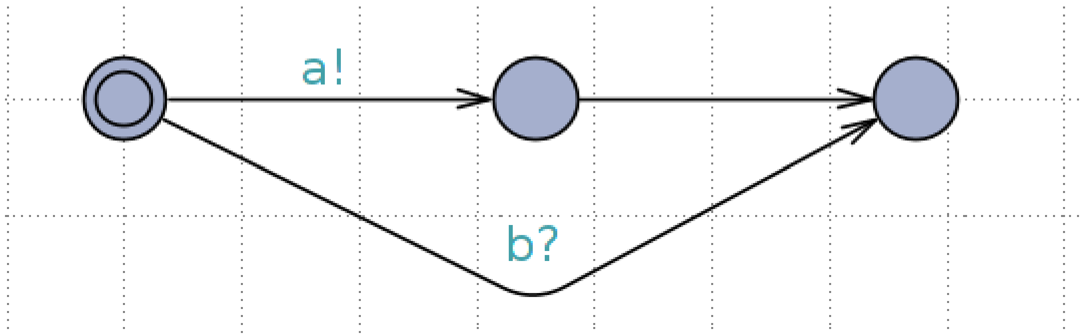
GUARD

A condition that must be true in order for a edge to be enabled



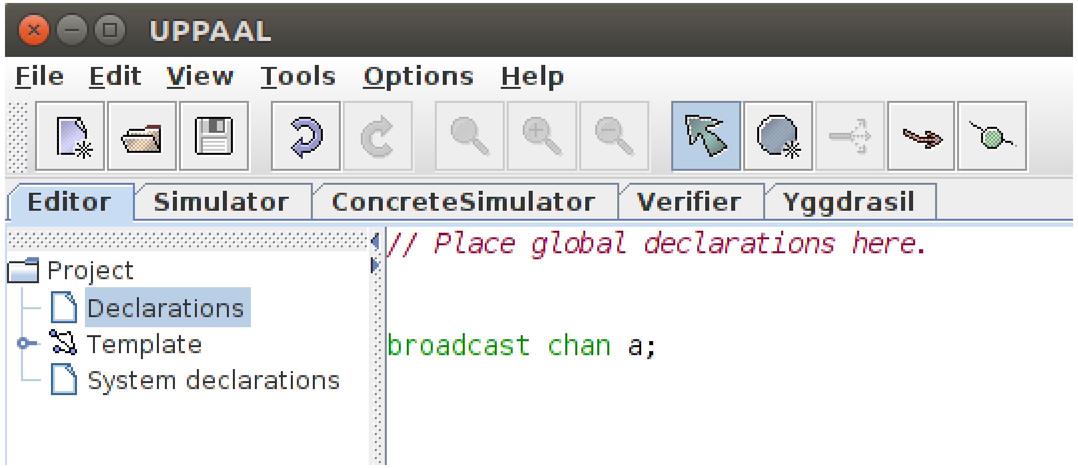
SYNCHRONIZATION

- The label on which the edge synchronizes with another edge
- If nothing is present
  - We call it a Tau τ / silent / epsilon ε transition
  - Can be taken alone



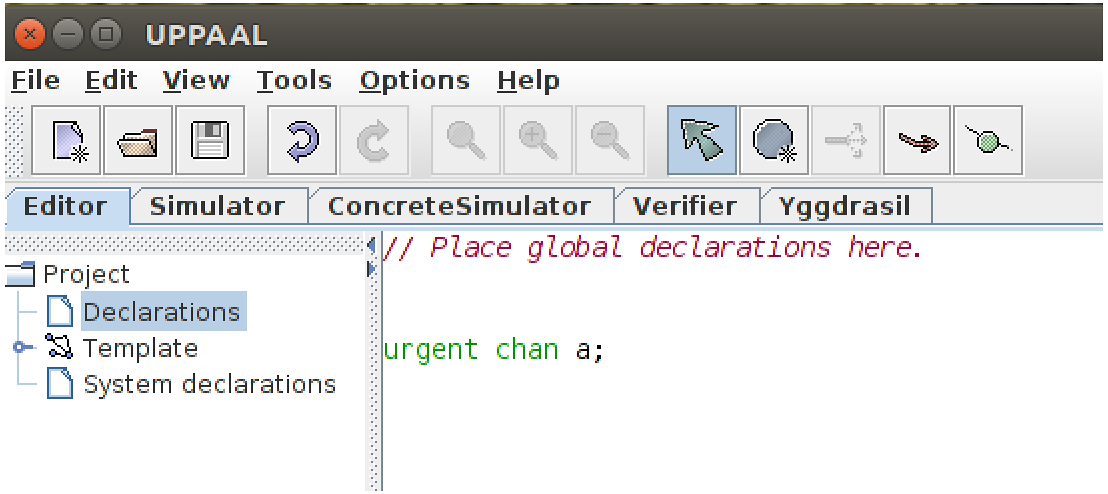
BROADCAST CHANNELS

- One sender
- Multiple receivers
  - All that can participate must participate
  - Note: Invariants after the input can block the execution of the complete broadcast



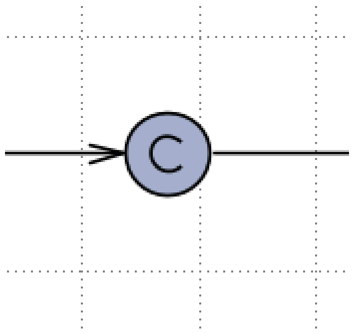
URGENT CHANNELS

- Must synchronize on an urgent channel as soon as it is possible
  - Does not allow clock guards on edges that synchronize on urgent channels
  - Data guards on the receiver can be a problem



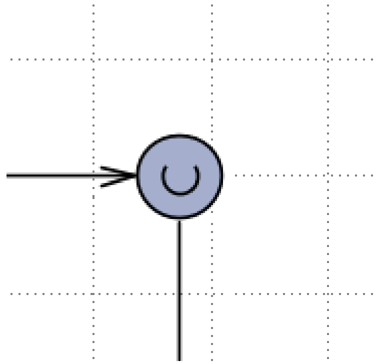
COMMITTED LOCATION

- Time must not pass while this location is part of the global state
- If there is any committed location among the locations in the global state then the next transition must involve at least one committed location



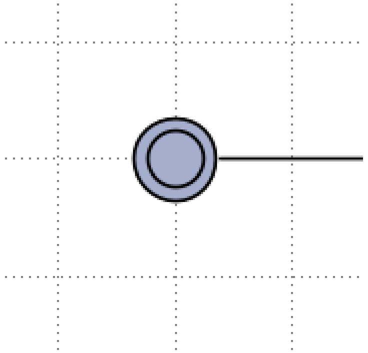
URGENT LOCATION

- Time must not pass while this location is part of the global state

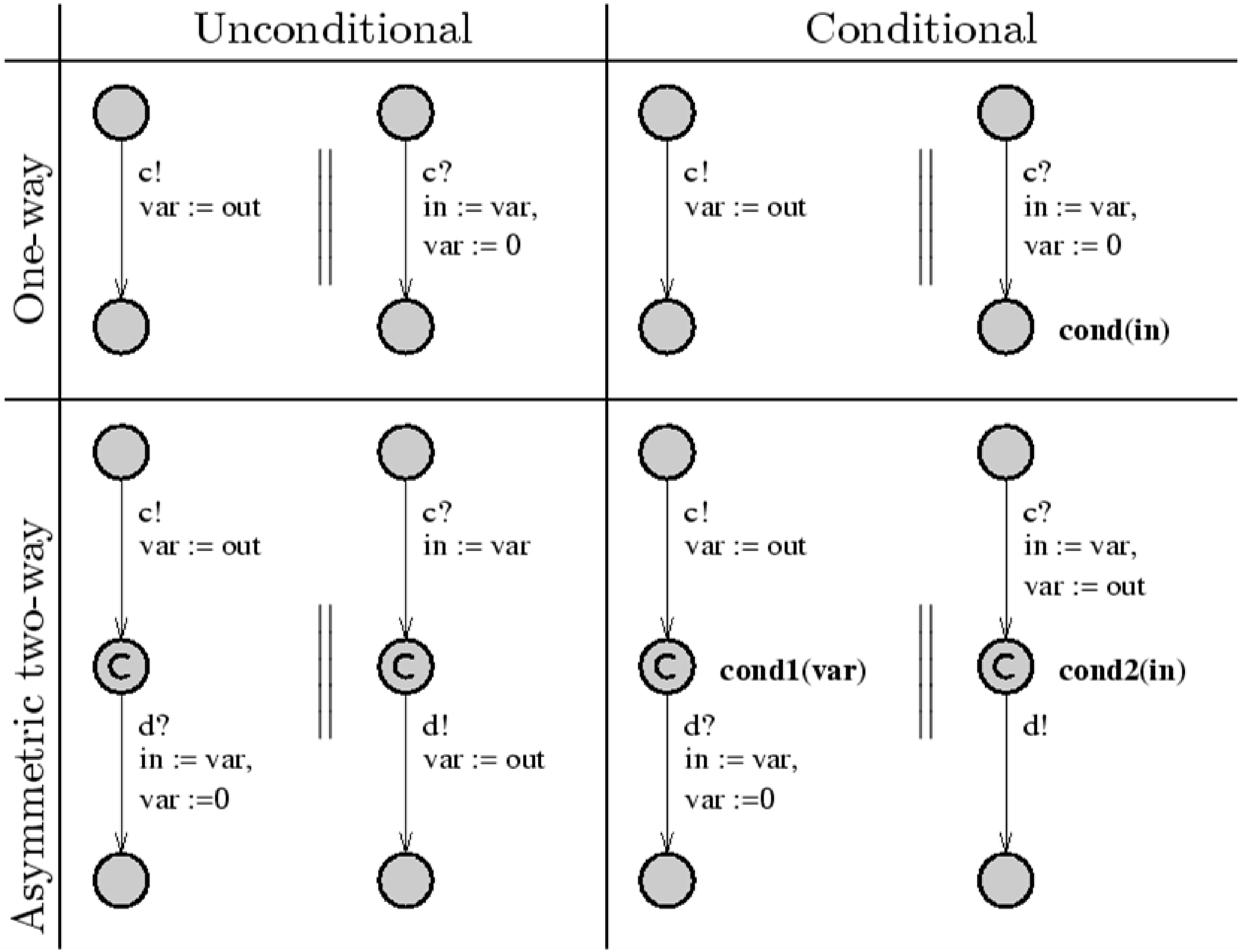


INITIAL LOCATION

- The location in which a given process starts



SYNCHRONOUS VALUE PASSING



MODELLING LANGUAGE

MODELING LANGUAGE

SAL SILICON AUSTRIA LABS

- Global and local definitions, and system declaration

- Types
  - built-in types: *int*, *int[min,max]*, *bool*, arrays
  - *typedef struct { … } name*
  - *typedef built-in-type name*
- Functions
  - C-style syntax, no pointer, can load C libraries
- Select
  - *name : type*
- Network of TA = instances of templates
  - argument *const type expression*
  - argument *type& name*

EXAMPLE: FREE PIZZA STOPWATCH

- Hit the stop button at exactly 10 seconds for pizza
- Two systems: watch and user
- Signals: Start, stop, tooLate, tooEarly, reward
- After the reward, the user shouts “freePizzzzza” into the world
- Global variable for coins, 10 coins as reward
  - Change model so that a pizza costs 20 coins
    - You need to hit the button twice
- Use concrete and symbolic simulation

Youtube screenshot

SPECIFICATION LANGUAGE

LOGICAL SPECIFICATIONS

- Validation Properties
  - Possibly: E<> *P*
- Safety Properties
  - Invariant: A[] *P*
  - Pos. Inv.: E[] *P*
- Liveness Properties
  - Eventually: A<> *P*
  - Leadsto: P → *Q*
- Bounded Liveness
  - Leads to within: $P \rightarrow_{\leq t} Q$

The expressions *P* and *Q* must be type safe, **side effect free**, and evaluate to a boolean.

Only references to integer variables, constants, clocks, and locations are allowed (and arrays of these).

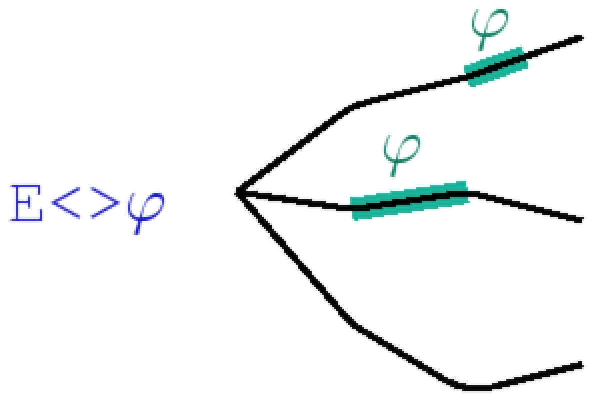
SYMBOLS

SAL SILICON AUSTRIA LABS

- ∃ = exists = there is one path
- ∀ = forall = for all paths
- □ = Always = The whole path
- ◊ = Eventually = At some point along the path

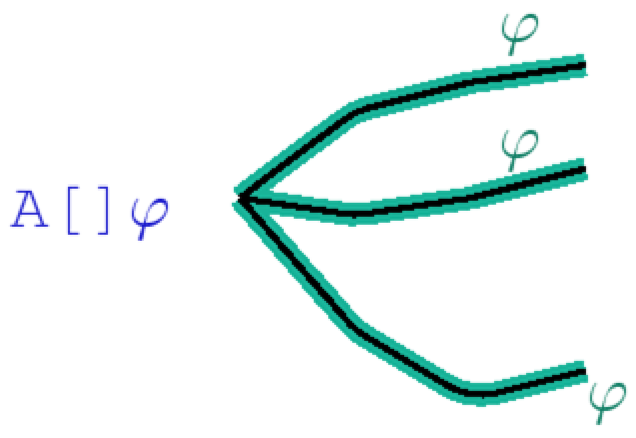
LOGICAL SPECIFICATIONS

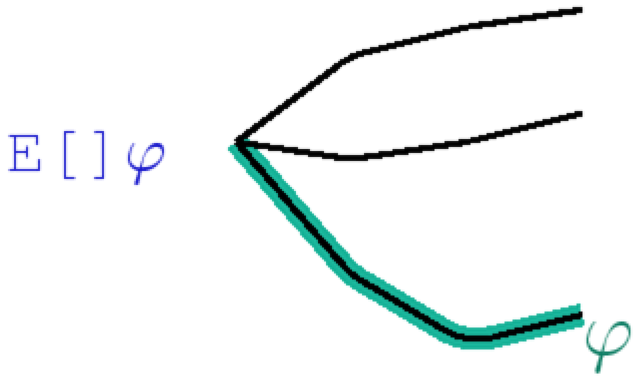
- Validation/Reachability Properties
  - Possibly: E<> *P*
- Safety Properties
  - Invariant: A[] *P*
  - Pos. Inv.: E[] *P*
- Liveness Properties
  - Eventually: A<> *P*
  - Leadsto: P → *Q*
- Bounded Liveness
  - Leads to within: *P* $\rightarrow_{\leq t}$ *Q*



LOGICAL SPECIFICATIONS

- Validation Properties
  - Possibly: E<> *P*
- Safety Properties
  - Invariant: A[] *P*
  - Pos. Inv.: E[] *P*
- Liveness Properties
  - Eventually: A<> *P*
  - Leadsto: P → *Q*
- Bounded Liveness
  - Leads to within: *P* $\rightarrow_{\leq t}$ *Q*





LOGICAL SPECIFICATIONS

SAL SILICON AUSTRIA LABS

- Validation Properties

–Possibly: E<> *P*

- Safety Properties

–Invariant: A[] *P*

–Pos. Inv.: E[] *P*

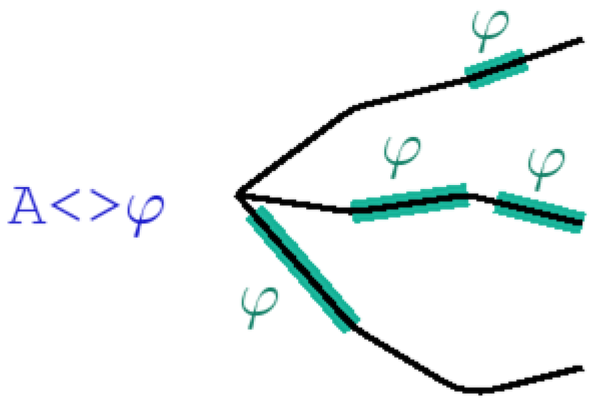
- Liveness Properties

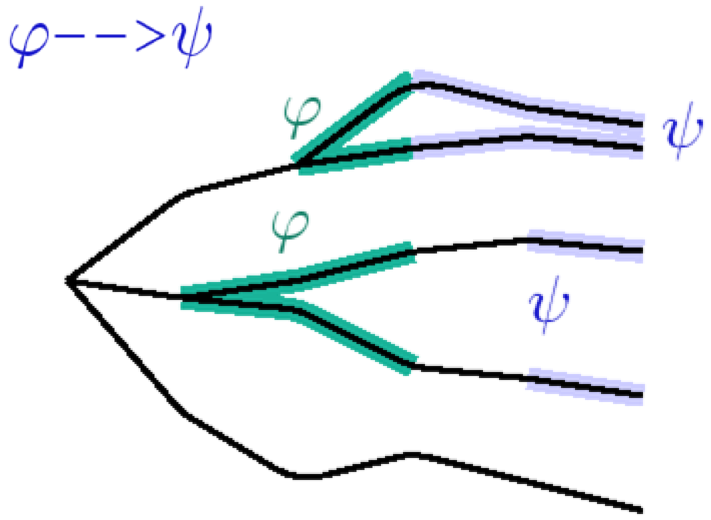
–Eventually: A<> *P*

–Leadsto: P → *Q*

- Bounded Liveness

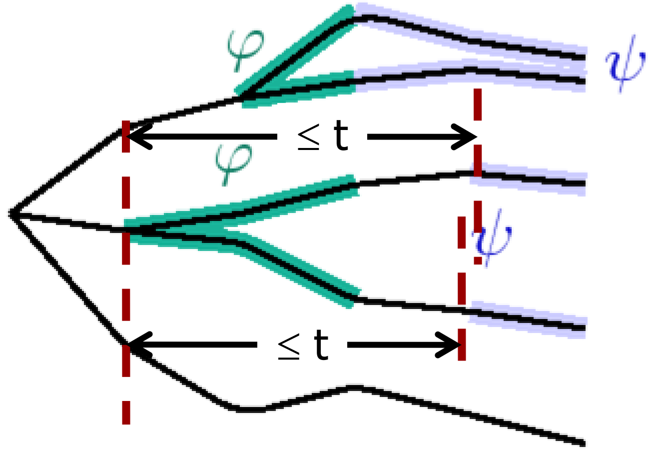
–Leads to within: $P \rightarrow_{\leq t} Q$





LOGICAL SPECIFICATIONS

- Validation Properties
  - Possibly: E<> *P*
- Safety Properties
  - Invariant: A[] *P*
  - Pos. Inv.: E[] *P*
- Liveness Properties
  - Eventually: A<> *P*
  - Leadsto: P → *Q*
- Bounded Liveness
  - Leads to within: $P \rightarrow_{\leq t} Q$



STOPWATCH EXAMPLE

SAL
SILICON AUSTRIA LABS

- Safety: Do debt allowed
  - A[] coins >= 0
- Validation/Reachability: We do not cheat
  - E<> coins =>10
- Try it out:
  - Can you think of more queries?
  - Make some queries that (should) fail

QUESTIONS

- What is the difference between an urgent location and an urgent channel?
- What is the difference between a committed and an urgent location?
- What is the difference between location and a state? And why do we care?
- How can I check if a model never reaches a certain state?
- How to check for deadlock freeness?

UPPAAL VERIFICATION ENGINE

STATE-SPACE EXPLOSION PROBLEM

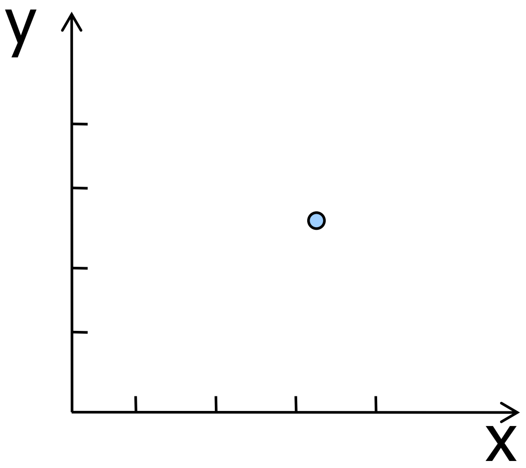
- 10 (11) components with 2 states each
  - 2^10 = 1024 states
  - 2^11 = 2048 states
- 2 (3/9) components with 10 states each
  - 10^2 = 100 states
  - 10^3 = 1000 states
  - 10^9 = 1000000000 states

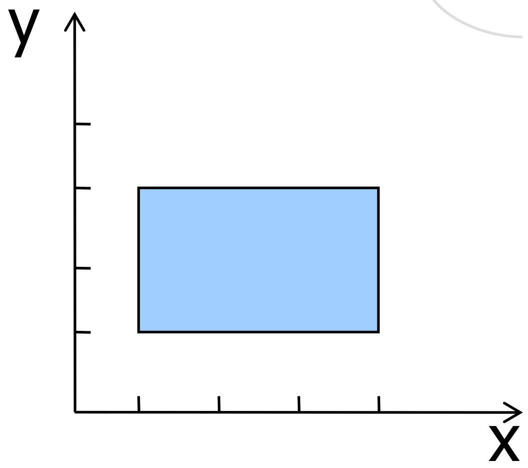
ZONES - FROM INFINITE TO FINITE

State
(n, x=3.2, y=2.5)

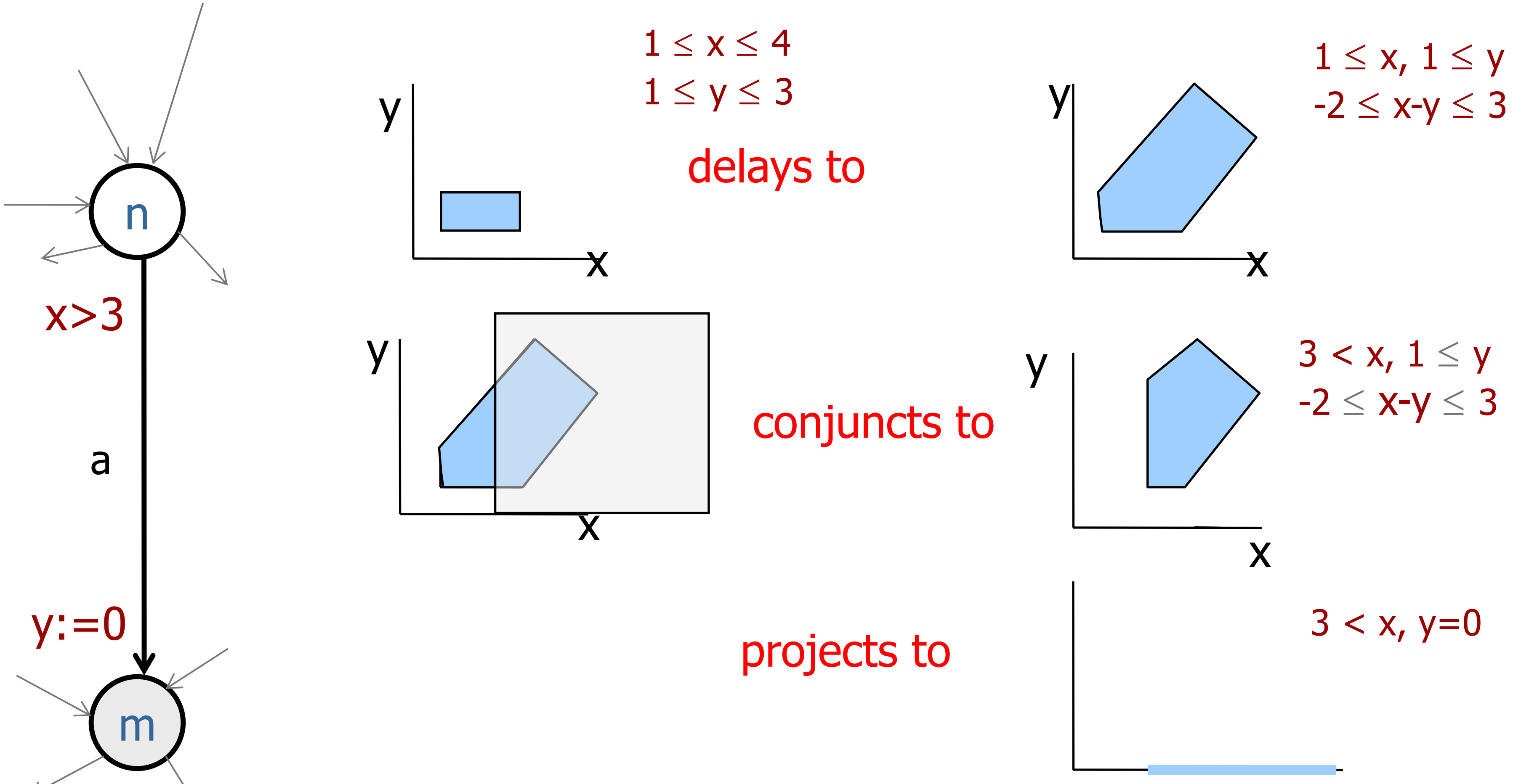
Symbolic state (set)
(n, $1 \leq x \leq 4$, $1 \leq y \leq 3$)

Zone:
conjunction of
x-y<=m,
x<=m,
x>=m





SYMBOLIC TRANSITIONS

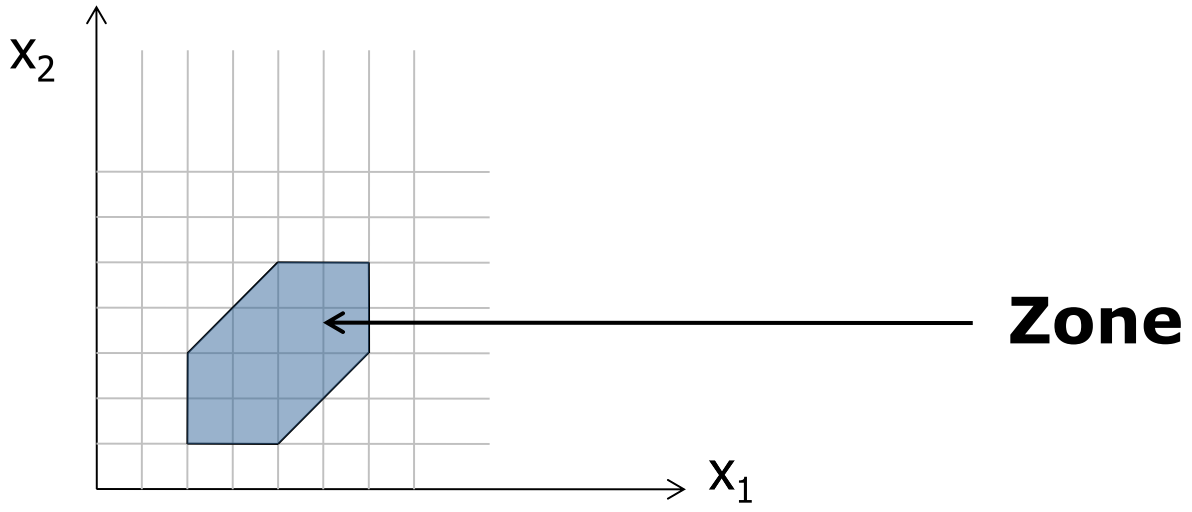


Thus (n, $1 \leq x \leq 4$, $1 \leq y \leq 3$) $\rightarrow^{a}$ (m, $3 < x$, $y=0$)

DIFFERENCE BOUND MATRICES

$x_0-x_0<=0$	$x_0-x_1<=-2$	$x_0-x_2<=-1$
$x_1-x_0<=6$	$x_1-x_1<=0$	$x_1-x_2<=3$
$x_2-x_0<=5$	$x_2-x_1<=1$	$x_2-x_2<=0$

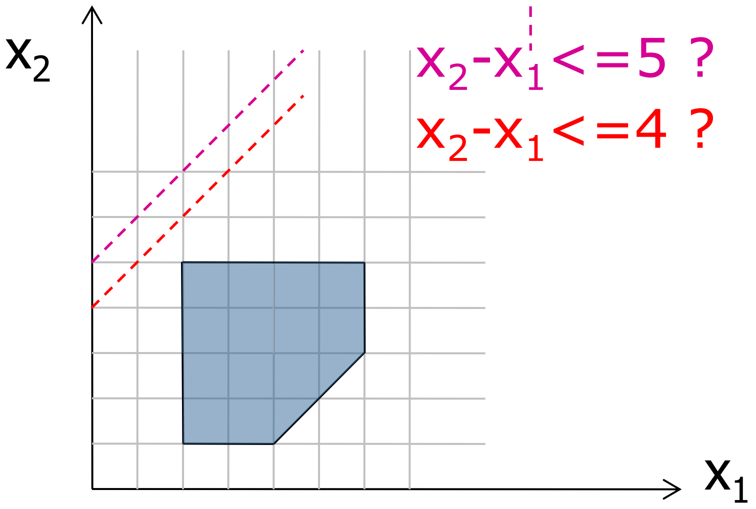
$$\mathbf{x_i-x_j<=c_{ij}}$$



DIFFERENCE BOUND MATRICES

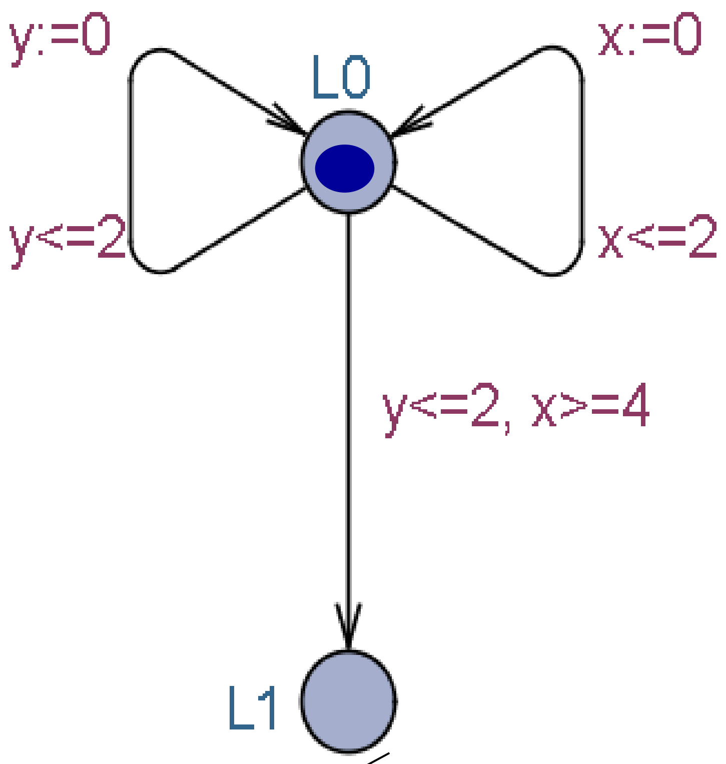
$x_0-x_0<=0$	$x_0-x_1<=-2$	$x_0-x_2<=-1$
$x_1-x_0<=6$	$x_1-x_1<=0$	$x_1-x_2<=3$
$x_2-x_0<=5$	$x_2-x_1<=$ **3**	$x_2-x_2<=0$

$$x_i-x_j<=c_{ij}$$

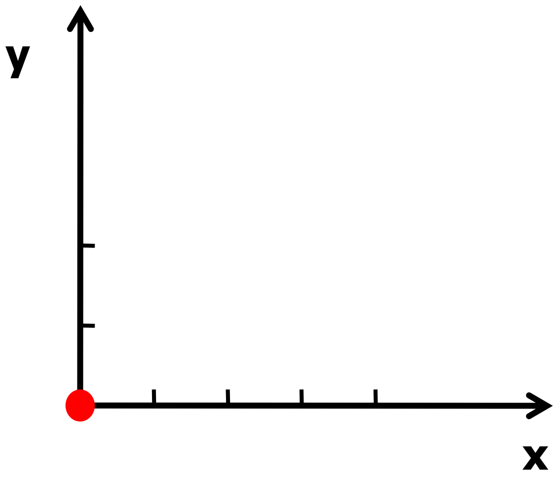


Canonical representation:
All constraints **as tight as possible**.
Needed for inclusion checking.
→ **Unique** DBM to represent a zone.

SYMBOLIC EXPLORATION

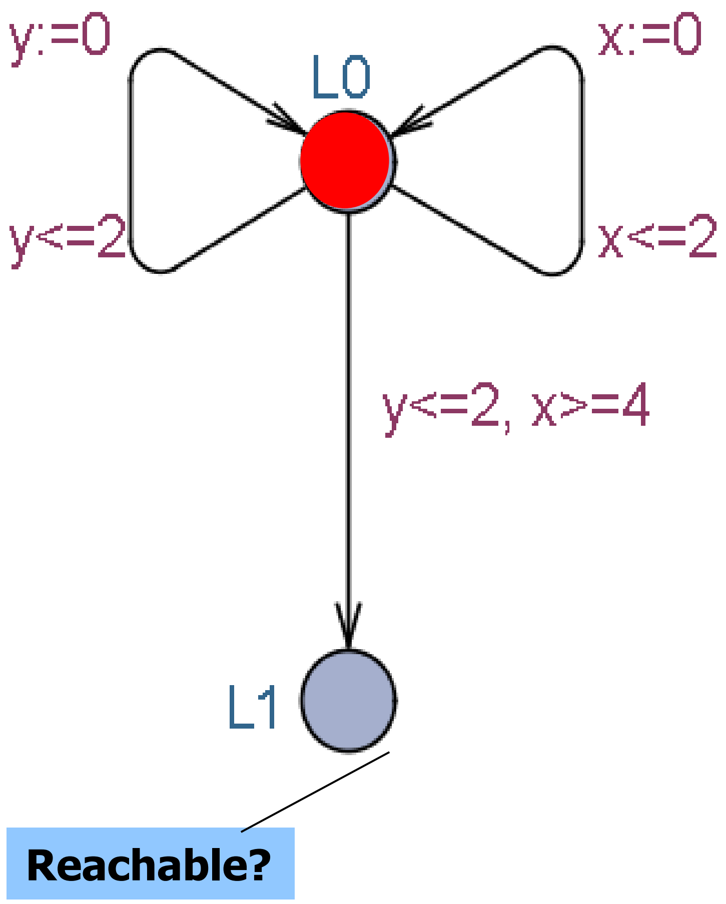


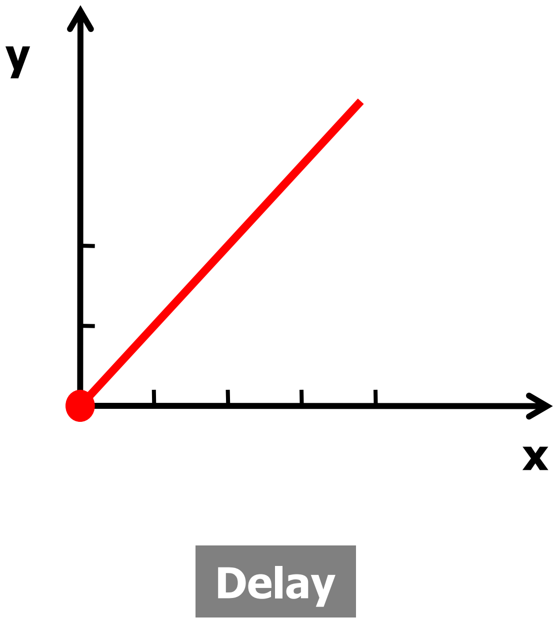
Reachable?



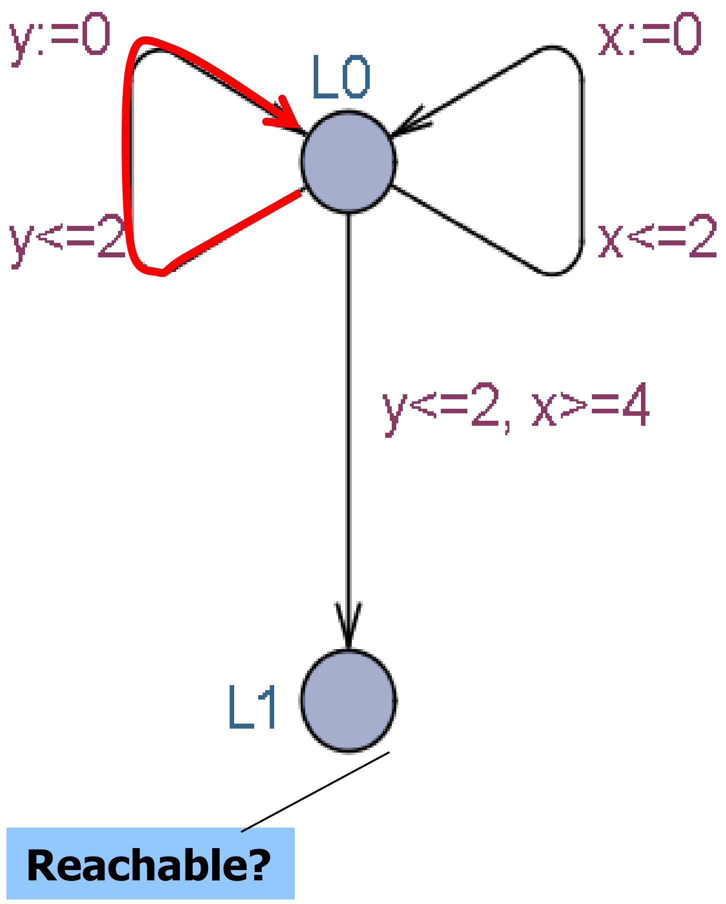
SYMBOLIC EXPLORATION

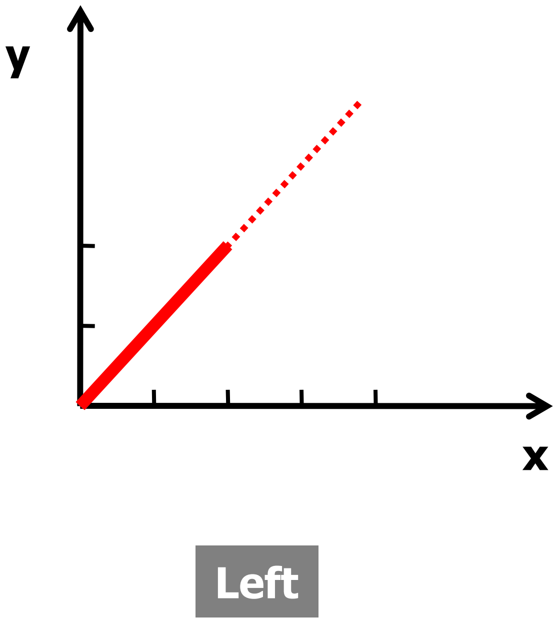
SAL SILICON AUSTRIA LABS





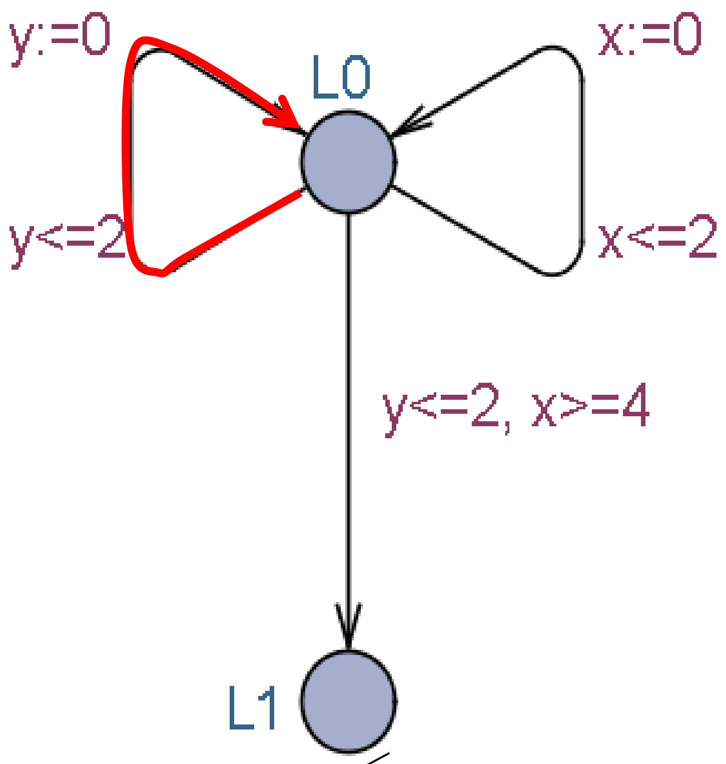
SYMBOLIC EXPLORATION



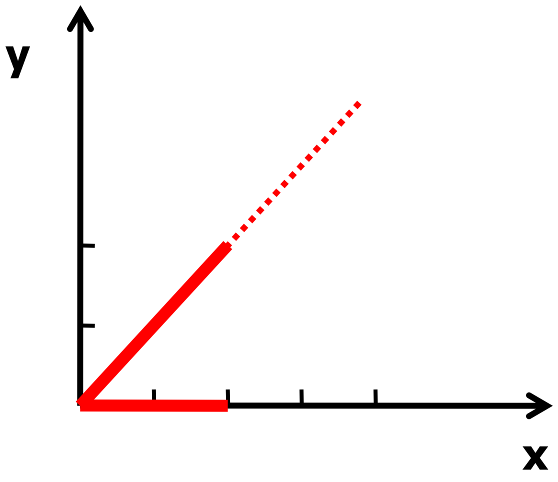


SYMBOLIC EXPLORATION

SAL SILICON AUSTRIA LABS

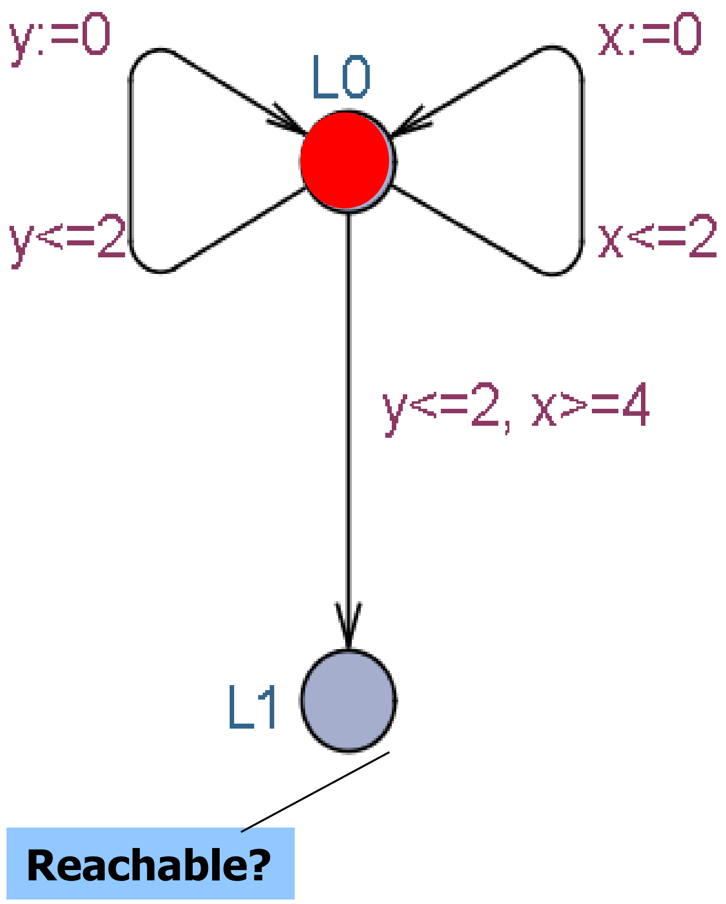


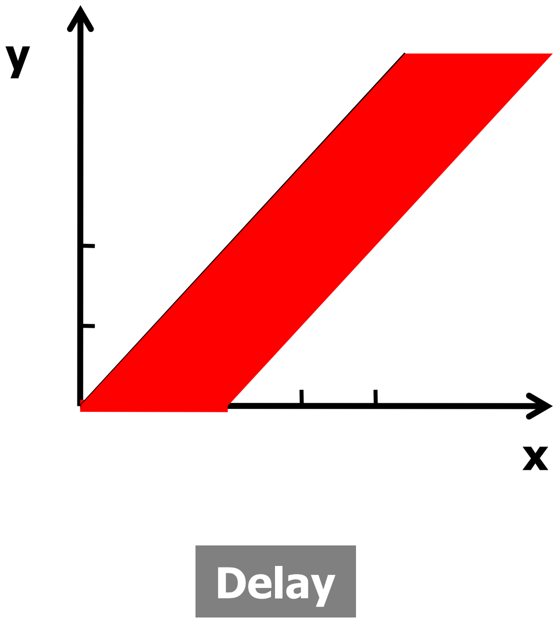
Reachable?



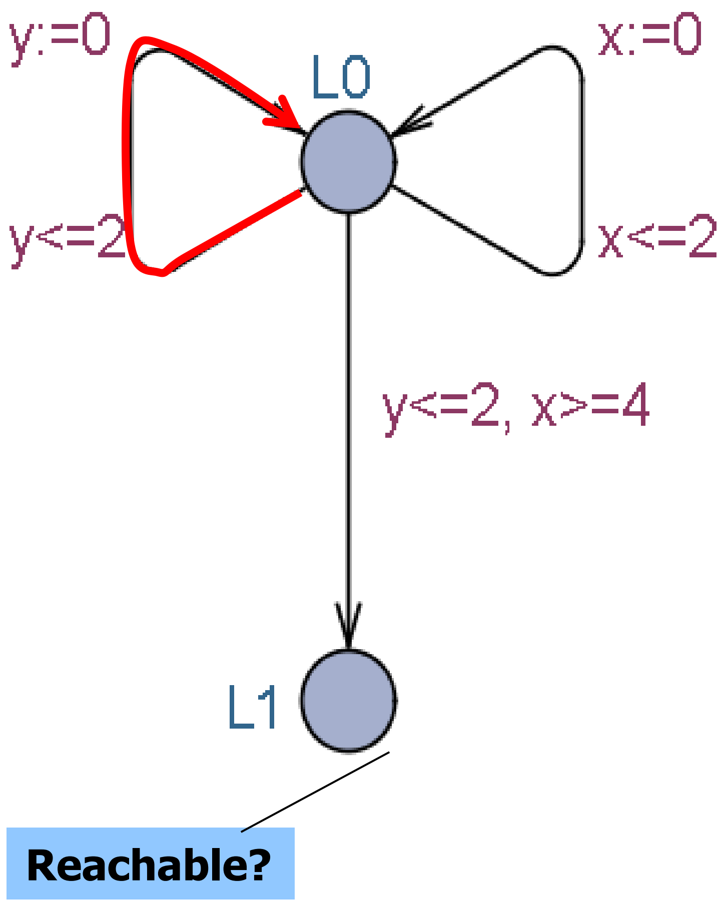
Left

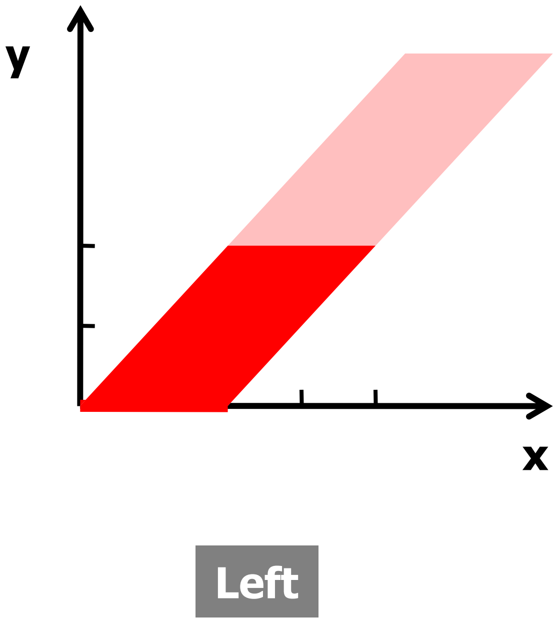
SYMBOLIC EXPLORATION



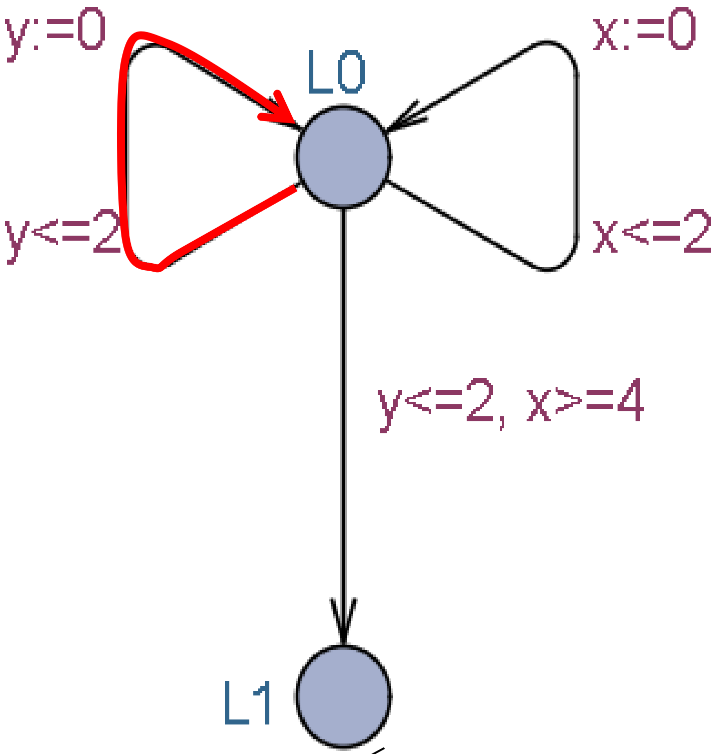


SYMBOLIC EXPLORATION

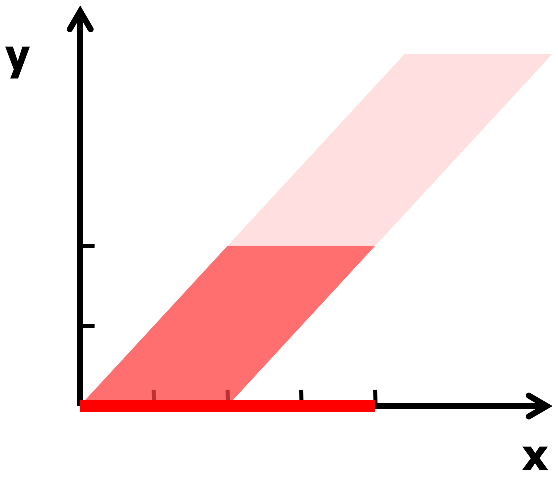




SYMBOLIC EXPLORATION

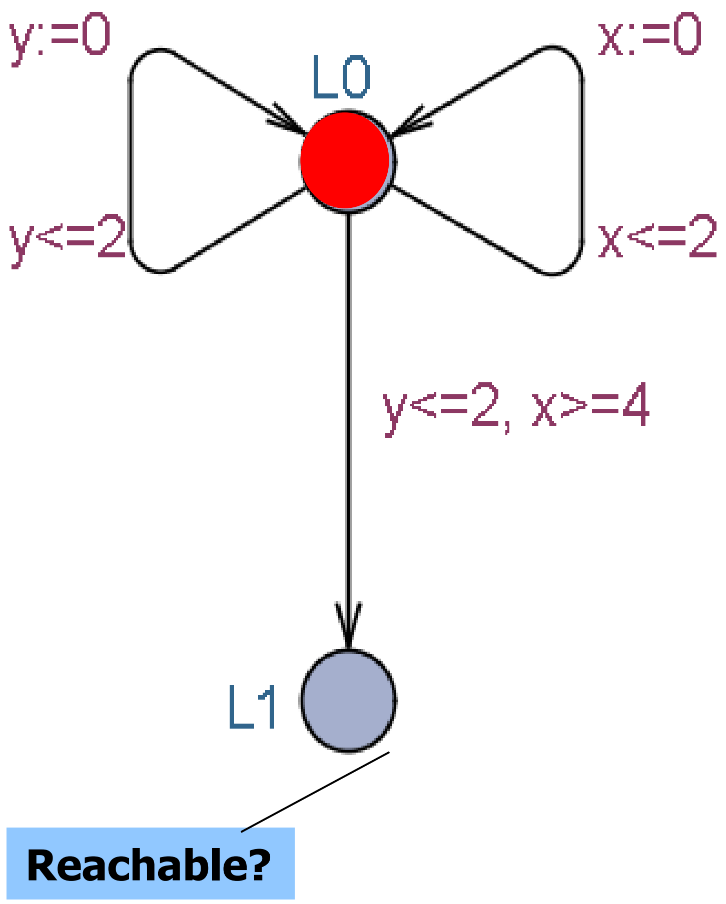


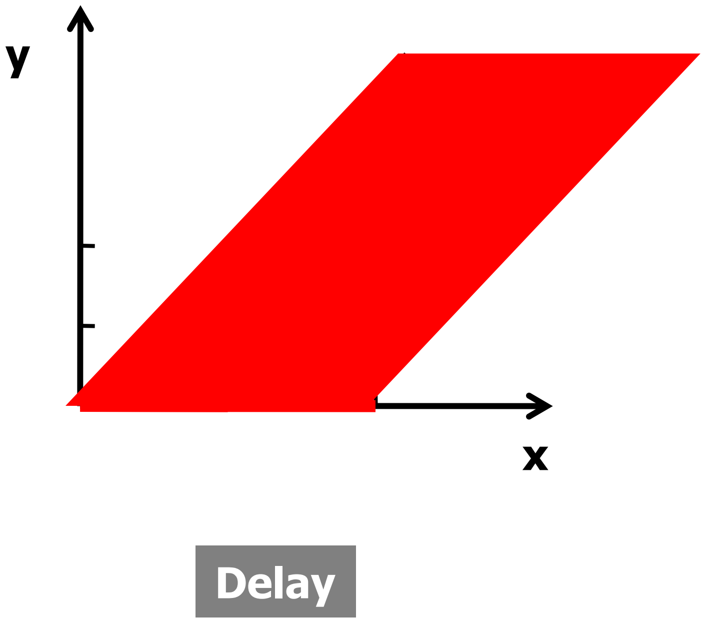
Reachable?



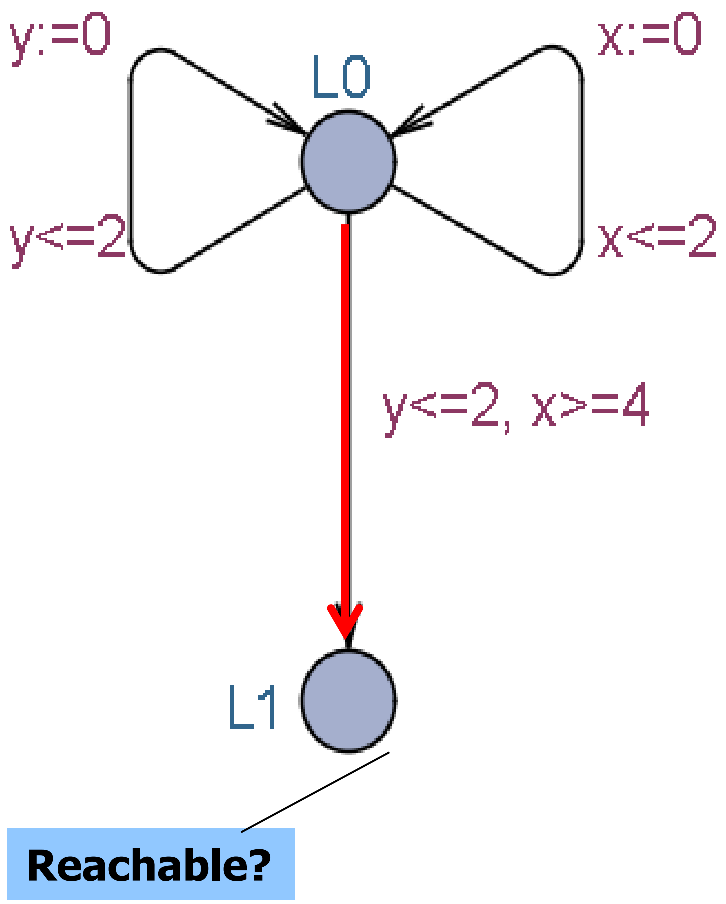
Left

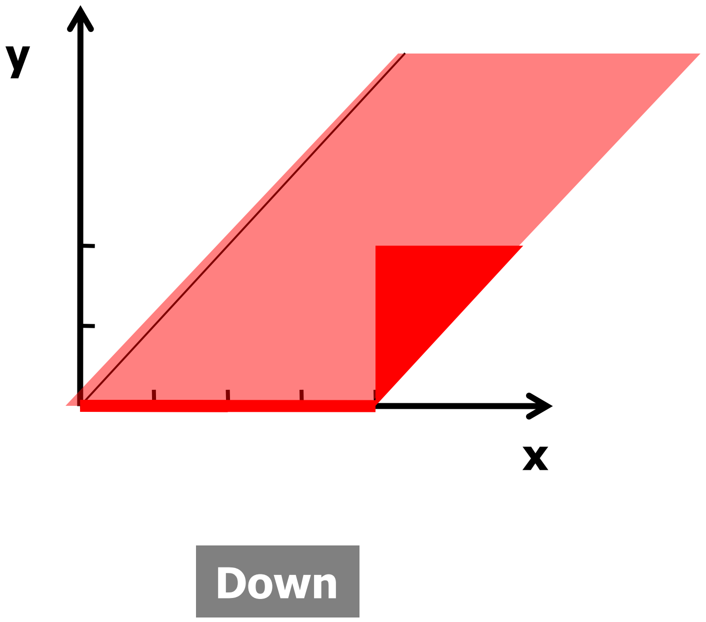
SYMBOLIC EXPLORATION





SYMBOLIC EXPLORATION



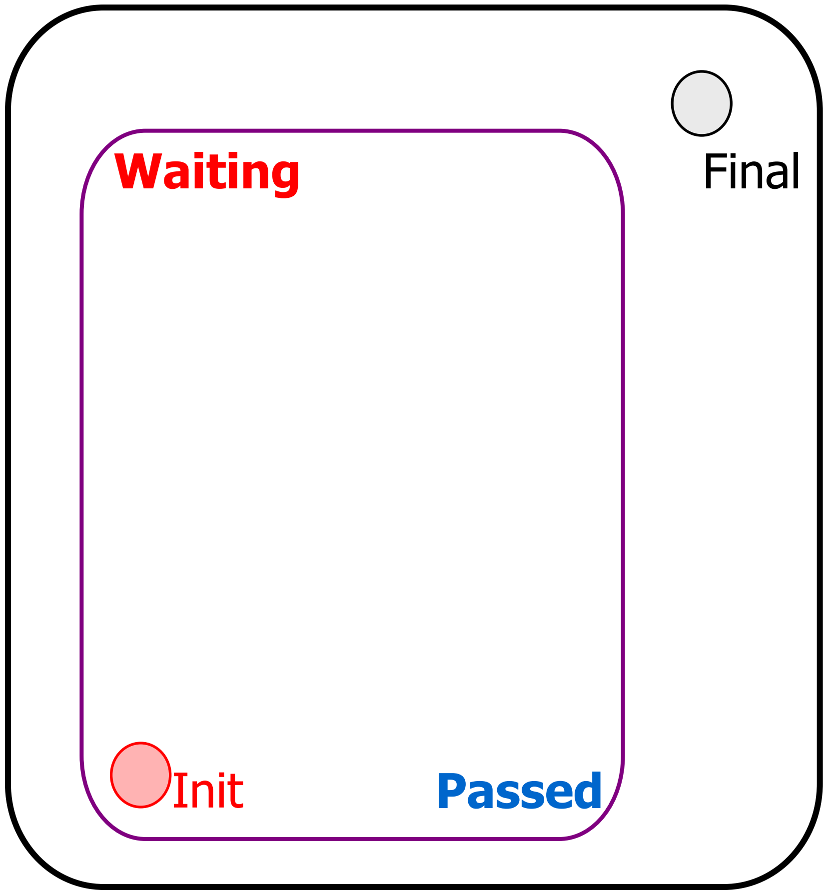


DEMO

Search order,
Clock constraints in simulator
Diagnostic trace

FORWARD REACHABILITY ALGORITHM

Init -> Final ?



INITIAL **Passed** := Ø;
Waiting := {(n_0,Z_0)}

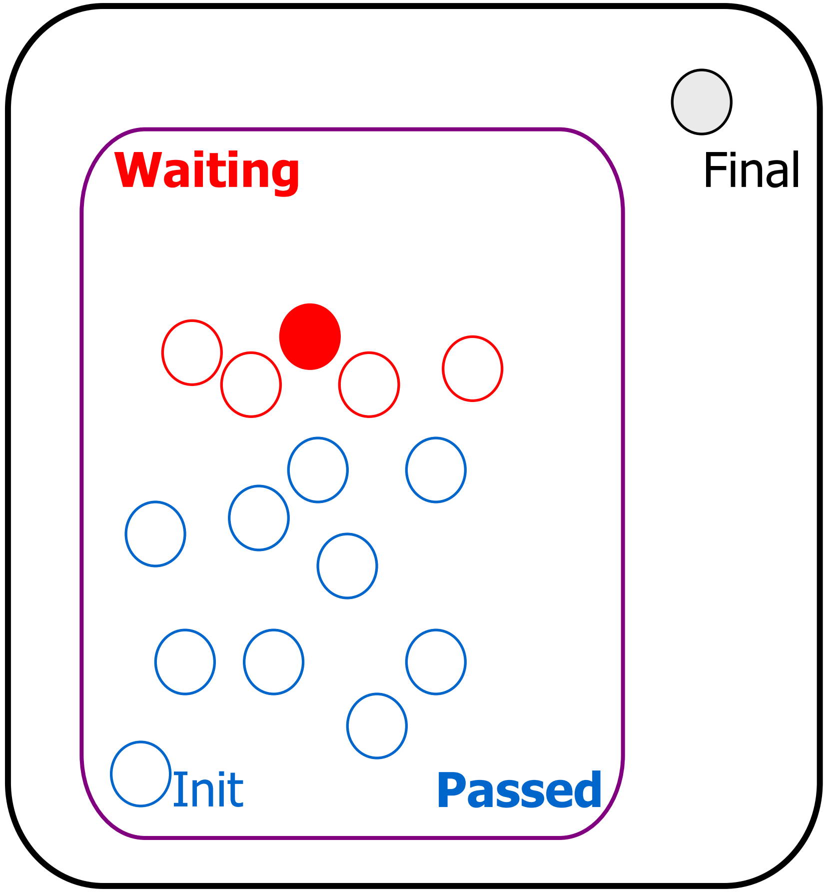
REPEAT

UNTIL **Waiting** = Ø
return false

FORWARD REACHABILITY ALGORITHM

SAL SILICON AUSTRIA LABS

Init -> Final ?



INITIAL **Passed** := Ø;
Waiting := {(n_0,Z_0)}

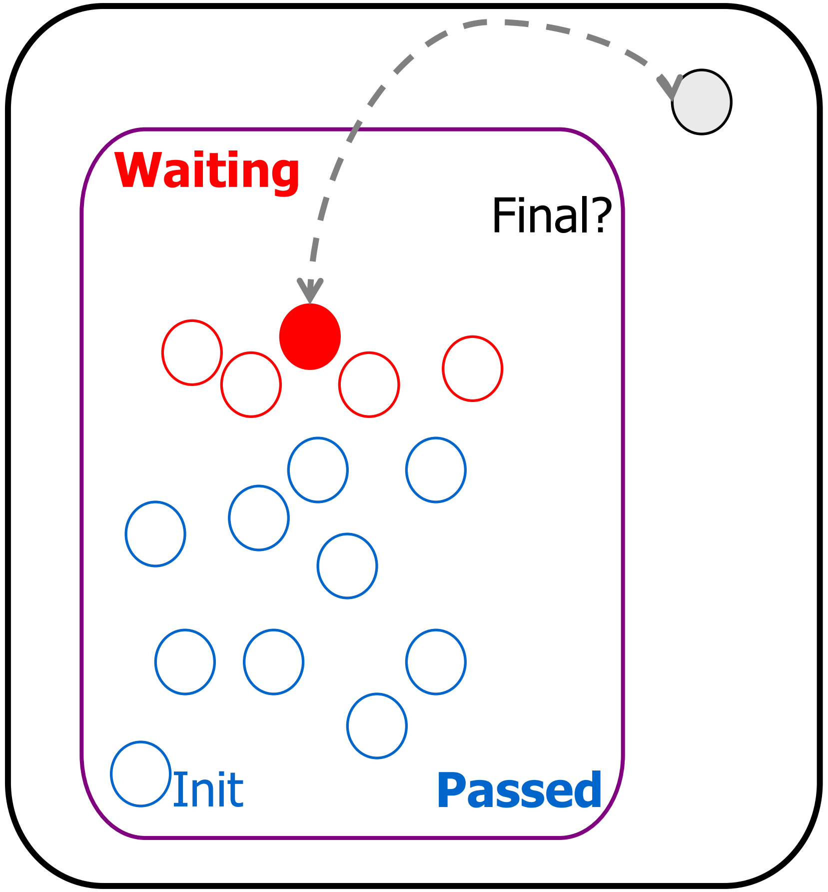
REPEAT
pick **(n,Z)** in **Waiting**

UNTIL **Waiting** = Ø
return false

FORWARD REACHABILITY ALGORITHM

SAL SILICON AUSTRIA LABS

Init -> Final ?



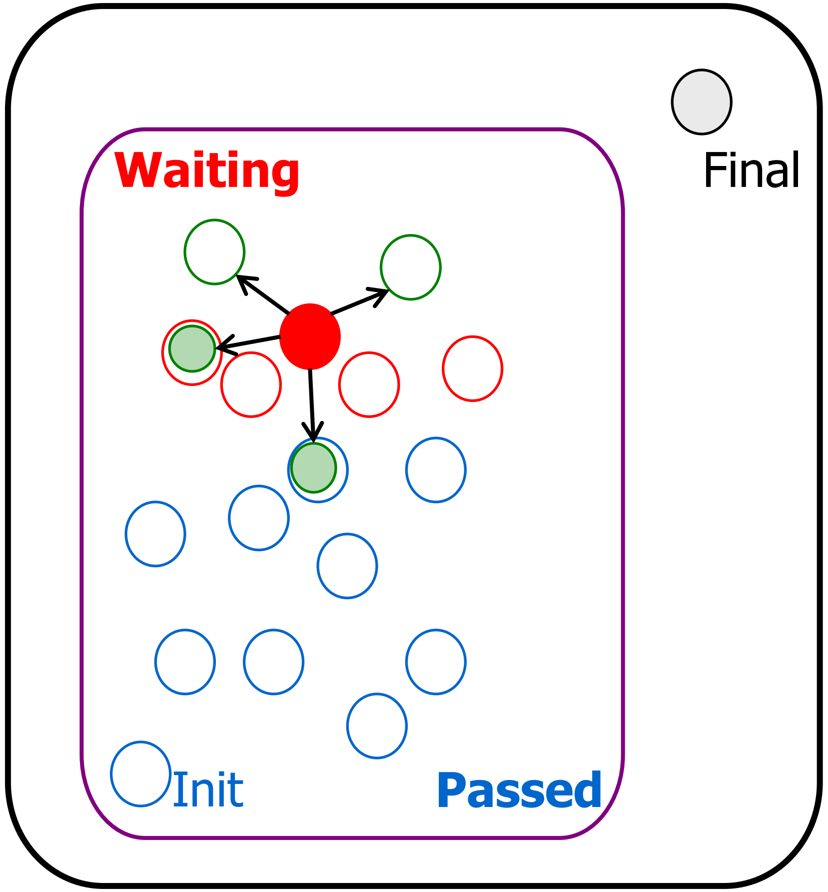
INITIAL **Passed** := Ø;
Waiting := {(n_0,Z_0)}

REPEAT
pick **(n,Z)** in **Waiting**
if (n,Z) = Final **return true**

UNTIL **Waiting** = Ø
return false

FORWARD REACHABILITY ALGORITHM

Init -> Final ?



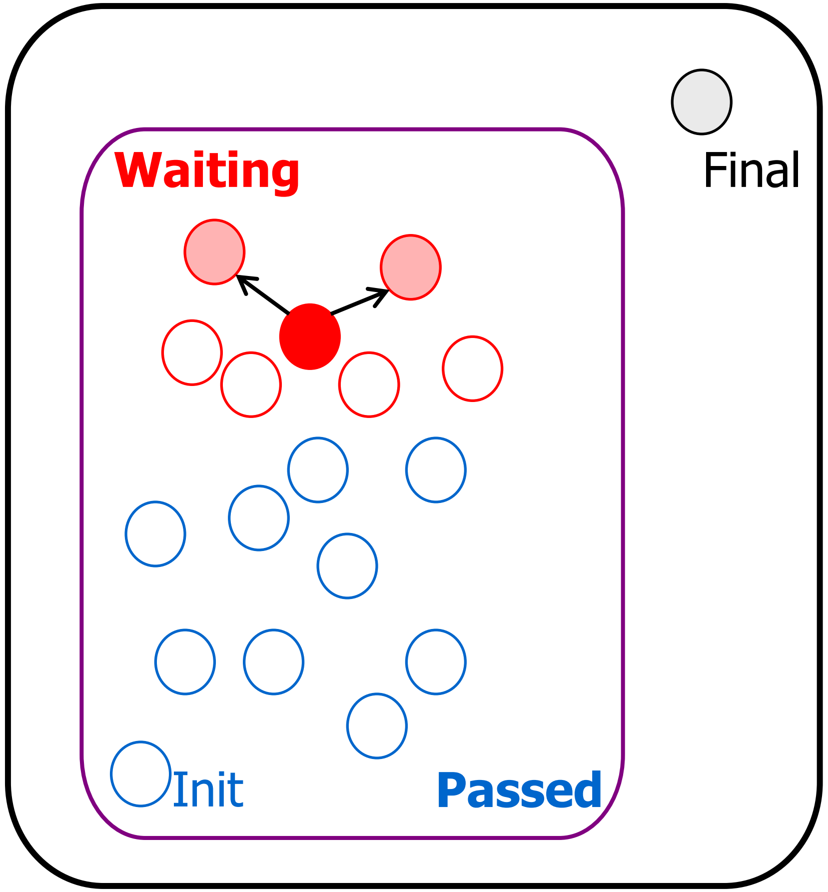
INITIAL **Passed** := Ø;
Waiting := {(n_0,Z_0)}

REPEAT
pick **(n,Z)** in **Waiting**
if (n,Z) = Final **return true**
for all (n,Z)→(**n'**,Z'):
if for some (**n'**,Z") **Z'** ⊆ **Z"** **continue**

UNTIL **Waiting** = Ø
return false

FORWARD REACHABILITY ALGORITHM

Init -> Final ?



INITIAL **Passed** := Ø;
Waiting := {(n_0,Z_0)}

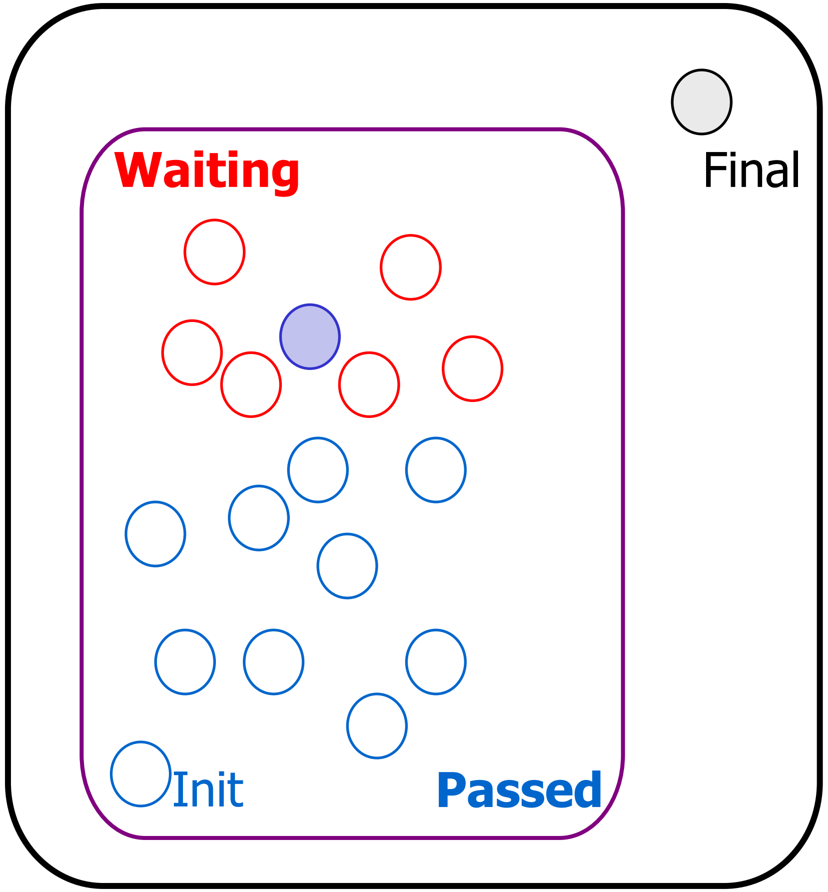
REPEAT
pick **(n,Z)** in **Waiting**
if (n,Z) = Final **return true**
for all (n,Z)→(**n′**,Z′):
if for some (**n′**,Z″) **Z′⊆ Z″ continue**
else add (n′,Z′) to **Waiting**

UNTIL **Waiting** = Ø
return false

FORWARD REACHABILITY ALGORITHM

SAL SILICON AUSTRIA LABS

Init -> Final ?



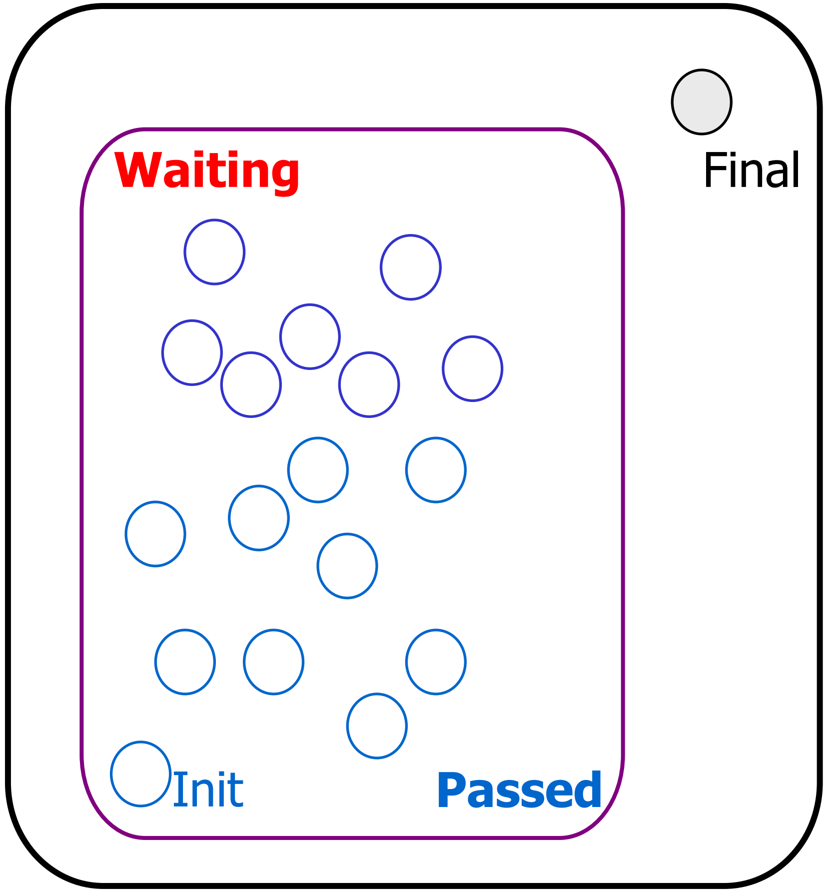
INITIAL **Passed** := Ø;
Waiting := {(n_0,Z_0)}

REPEAT
pick **(n,Z)** in **Waiting**
if (n,Z) = Final **return true**
for all (n,Z)→(**n'**,Z'):
if for some (**n'**,Z'') **Z'⊆ Z''** **continue**
else add (n',Z') to **Waiting**
move (n,Z) to **Passed**

UNTIL **Waiting** = Ø
return false

FORWARD REACHABILITY ALGORITHM

Init -> Final ?



INITIAL **Passed** := Ø;
Waiting := {(n_0,Z_0)}

REPEAT
pick **(n,Z)** in **Waiting**
if (n,Z) = Final **return true**
for all (n,Z)→**(n′**,Z′):
if for some (**n′**,Z″) **Z′⊆ Z″** **continue**
else add (n′,Z′) to **Waiting**
move (n,Z) to **Passed**

UNTIL **Waiting** = Ø
return false

ZENONESS

- Problem: UPPAAL does not check for zenoness directly.

– A model has "zeno" behavior if it can take an infinite amount of actions in finite time.

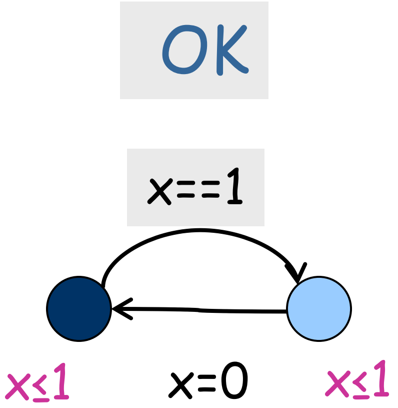
– That is usually not a desirable behavior in practice.

– Zeno models may wrongly conclude that some properties hold though they logically should not.

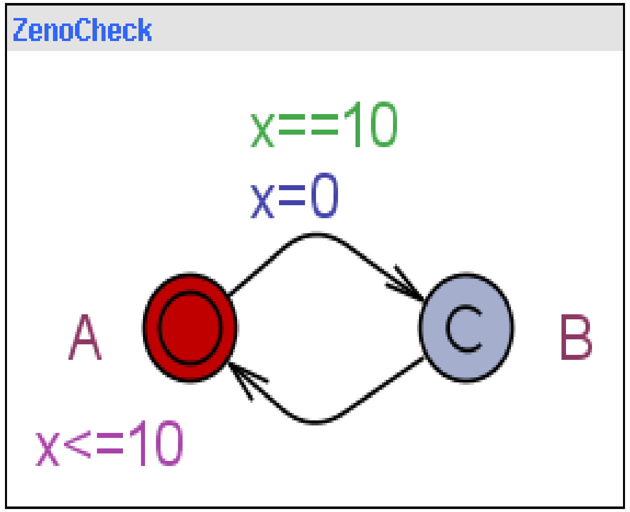
– Rarely taken into account.

- Solution: Add an observer automata and check for non-zenoness, i.e., that time will always pass.

ZENONESS



Detect by
•adding the observer:

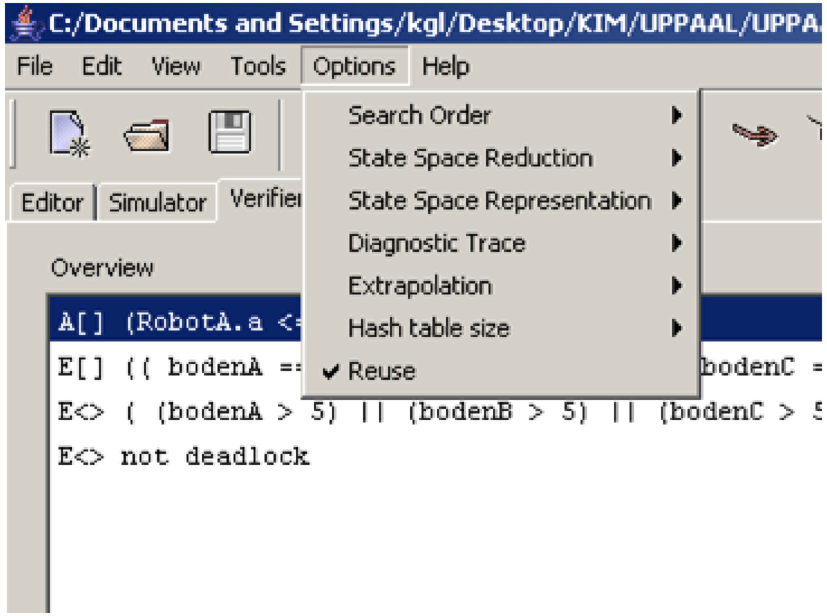


Constant (10) can be anything (>0), but choose it well w.r.t. your model for efficiency. Clocks 'x' are local.

•and check the property
ZenoCheck.A --> ZenoCheck.B

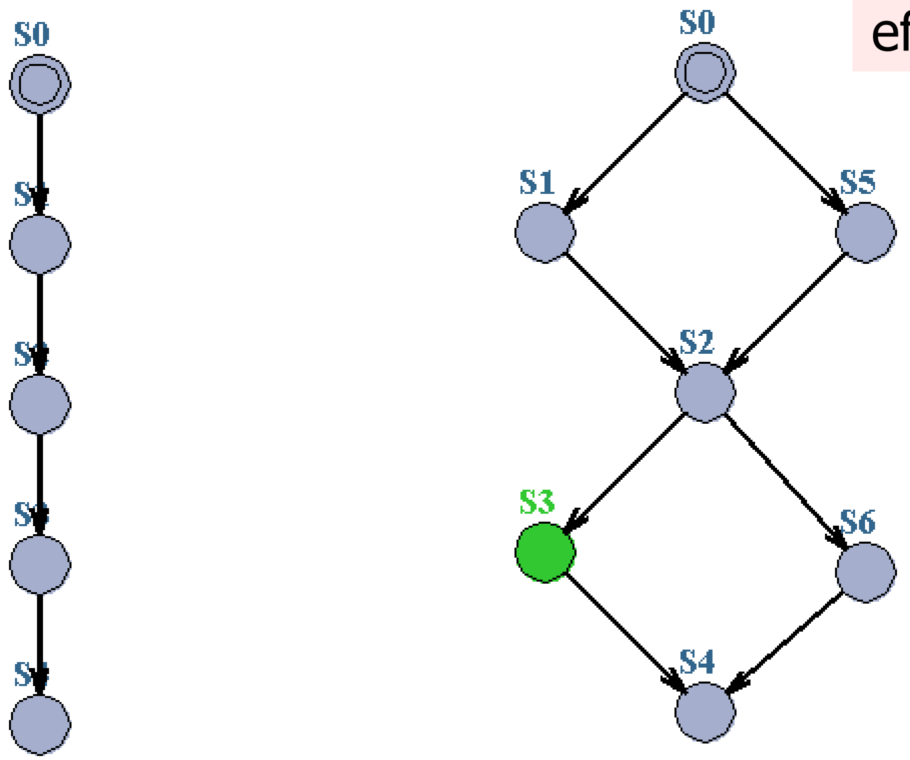
VERIFICATION OPTIONS

VERIFICATION OPTIONS



Search Order
Depth First
Breadth First
State Space Reduction
None
Conservative
Aggressive
State Space Representation
DBM
Compact Form
Under Approximation
Over Approximation
Diagnostic Trace
Some
Shortest
Fastest

STATE SPACE REDUCTION

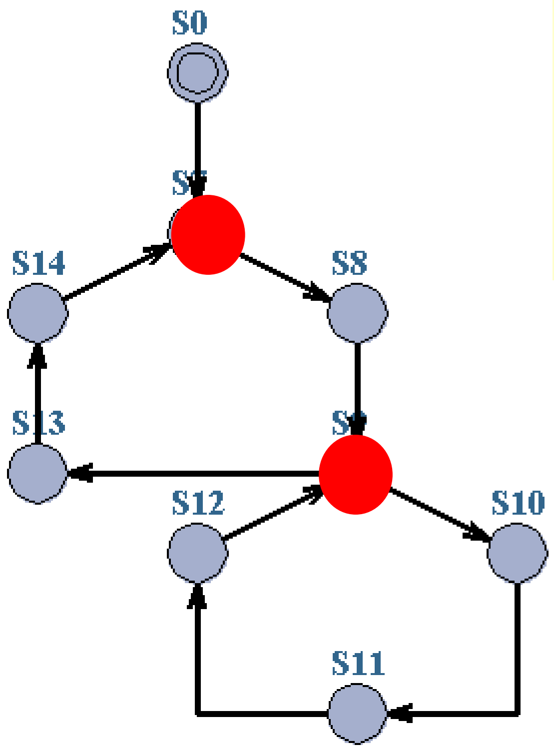


However,
Passed list useful for efficiency

No Cycles: **Passed** list not needed for *termination*

STATE SPACE REDUCTION

SAL
SILICON AUSTRIA LABS



Cycles:

Only symbolic states
involving loop-entry points
need to be saved on **Passed** list

TO STORE OR NOT TO STORE

SAL SILICON AUSTRIA LABS

117 states$_{total}$

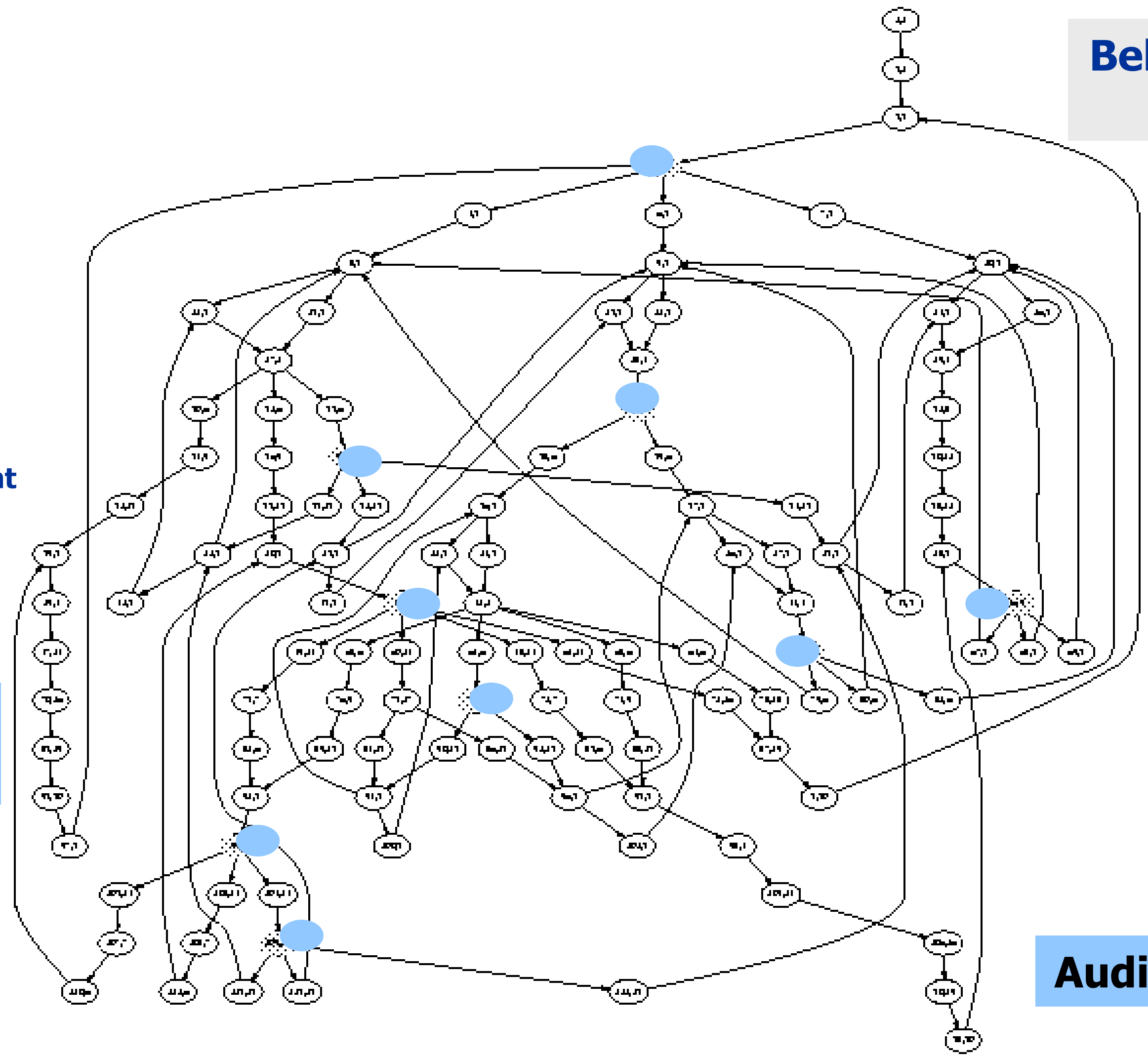
!

81 states$_{entrypoint}$

!

9 states

Time OH less than 10%

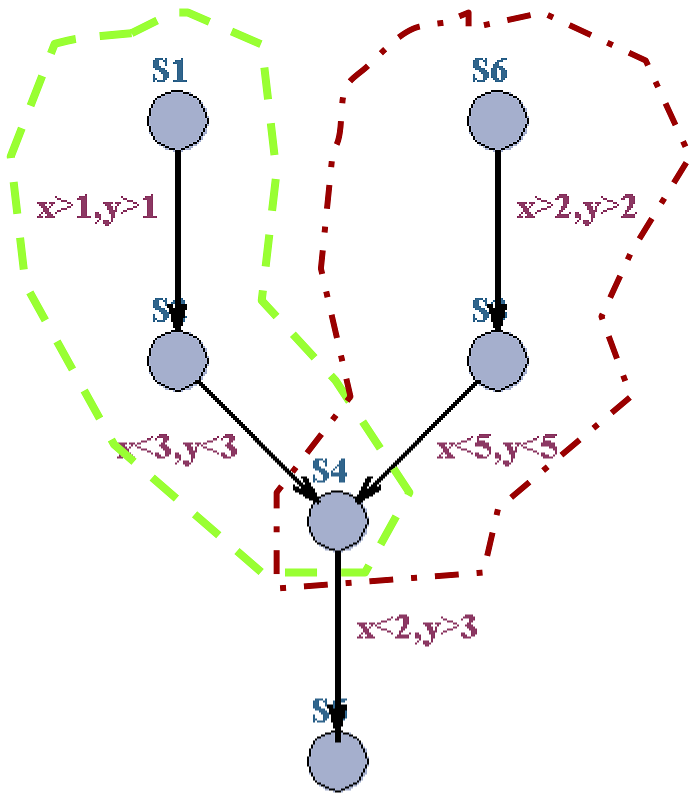


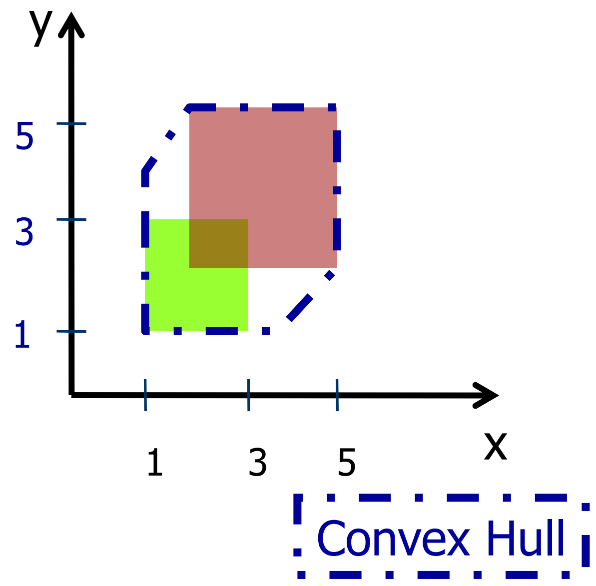
Behrmann, Larsen, Pelanek 2003

Audio Protocol

OVER-APPROXIMATION

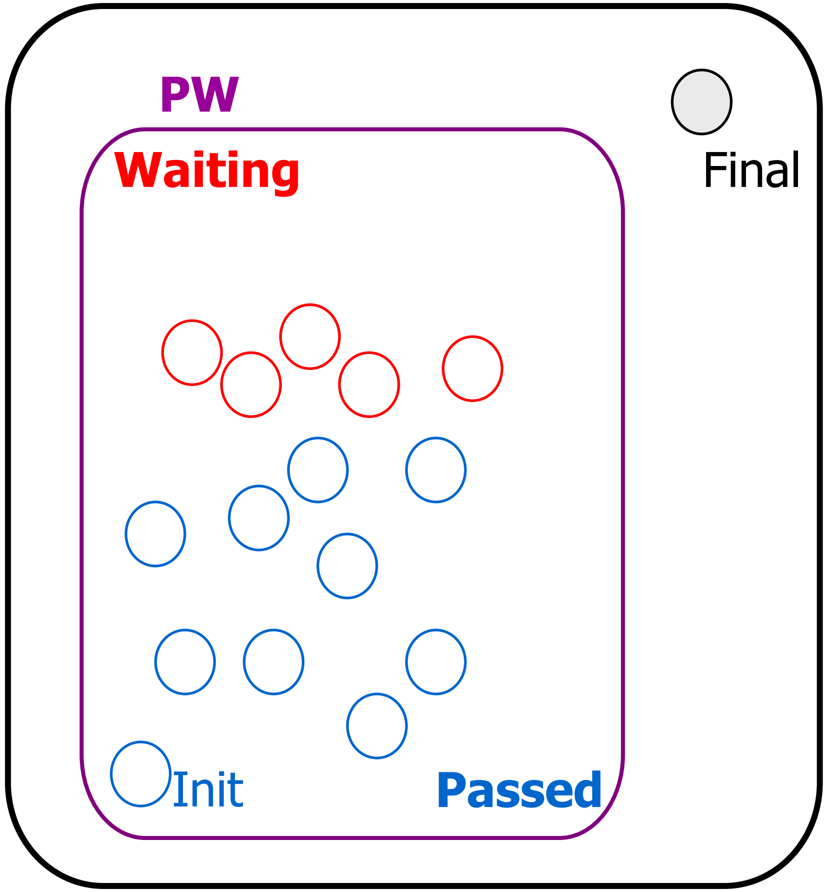
TACAS04: An **EXACT** method performing as well as Convex Hull has been developed based on abstractions taking max constants into account.





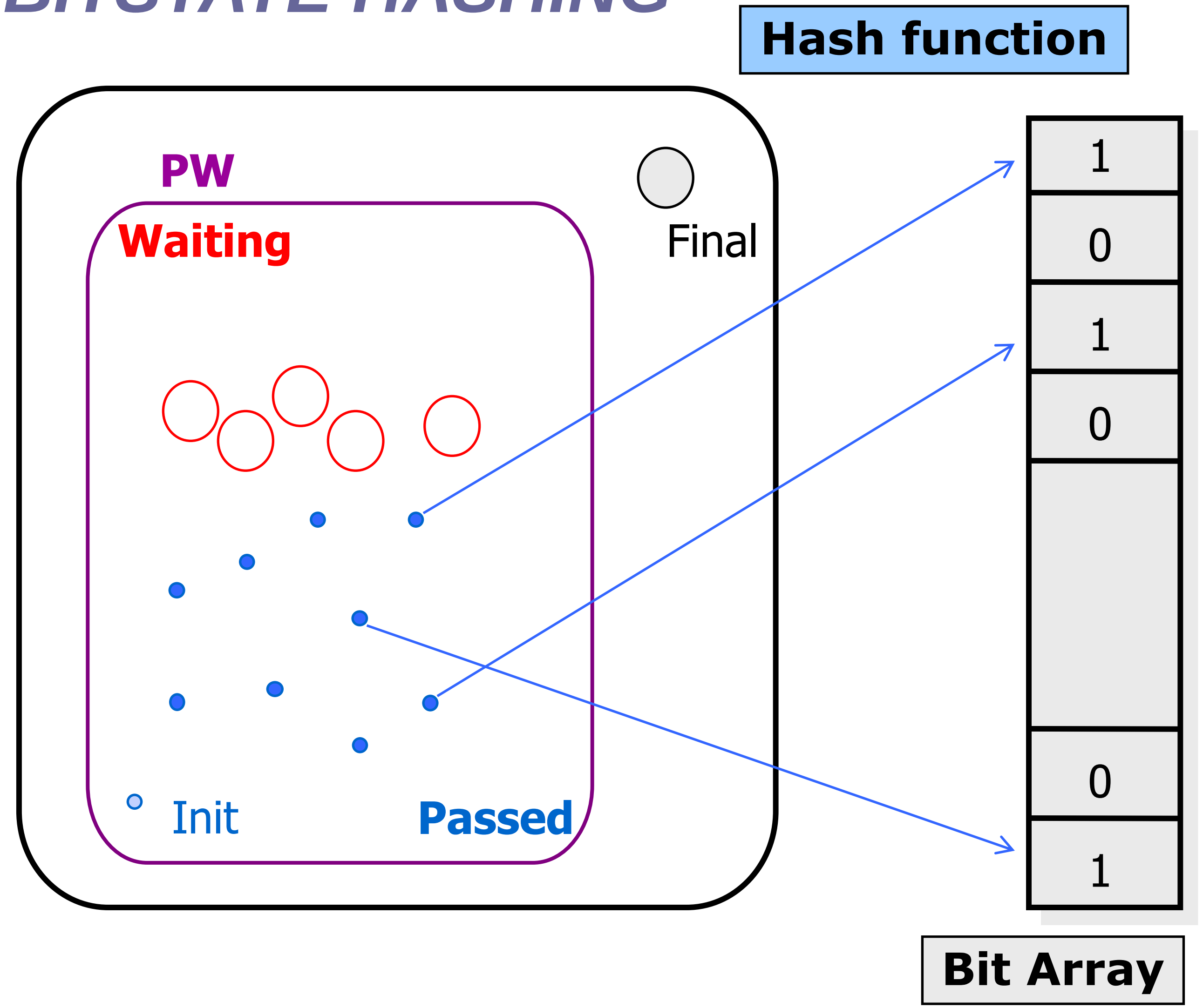
SAL SILICON AUSTRIA LABS

UNDER-APPROXIMATION
BITSTATE HASHING



UNDER-APPROXIMATION

BITSTATE HASHING



1 bit per passed state

Under-approx. Several states may collide on the same bit.

Inclusion check only with waiting states. "Equality" with passed.

MODELLING PATTERNS

VARIABLE REDUCTION

- Reduce size of state space by explicitly resetting variables when they are not used!
- Automatically performed for clock variables (active clock reduction)

```
// Remove the front element of the queue
void dequeue()
{
        int i = 0;
        len -= 1;
        while (i < len)
        {
                list[i] = list[i + 1];
                i++;
        }
        list[i] = 0;
}
```

VARIABLE REDUCTION

- Railway controller

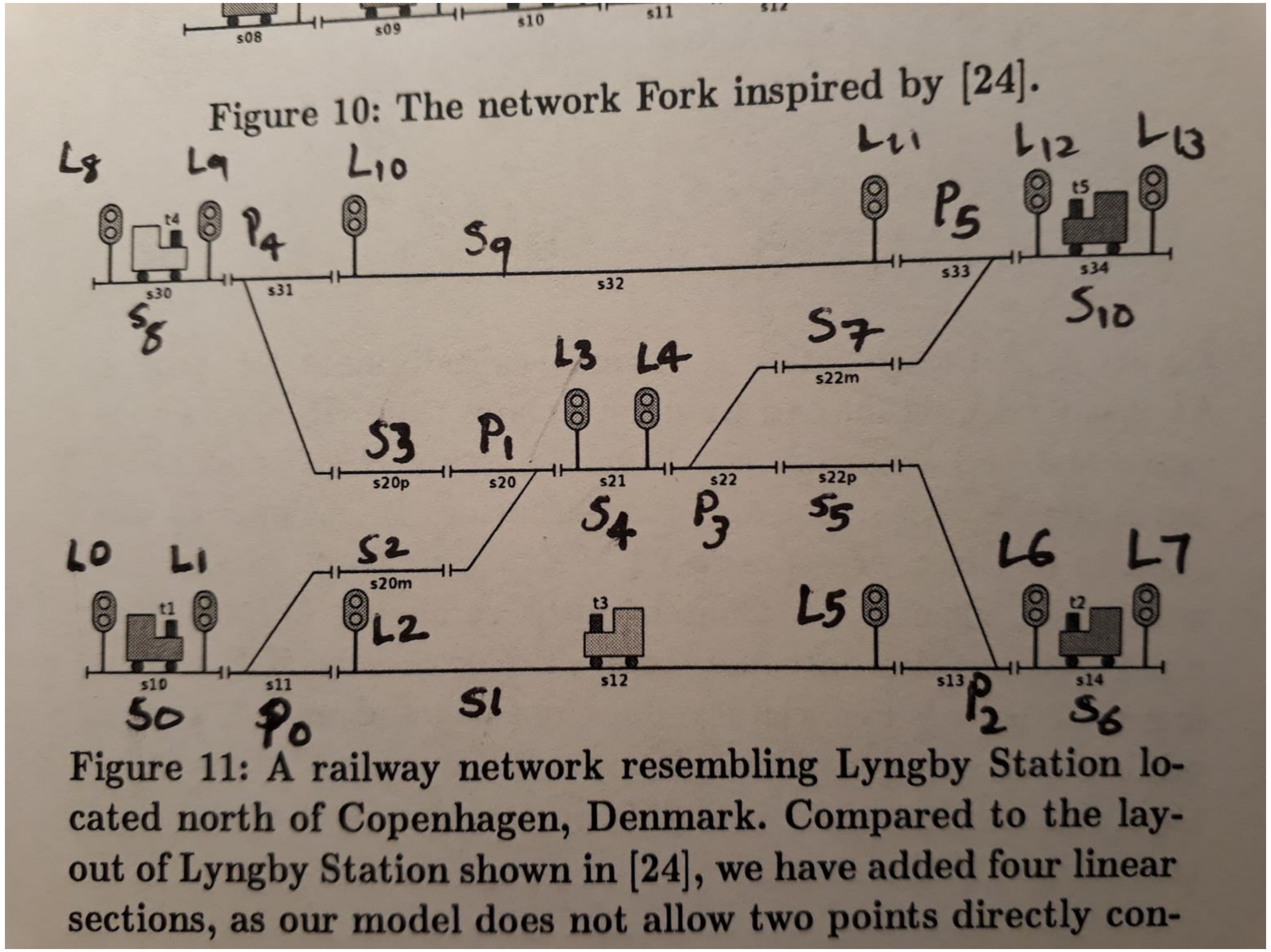


Figure 10: The network Fork inspired by [24].

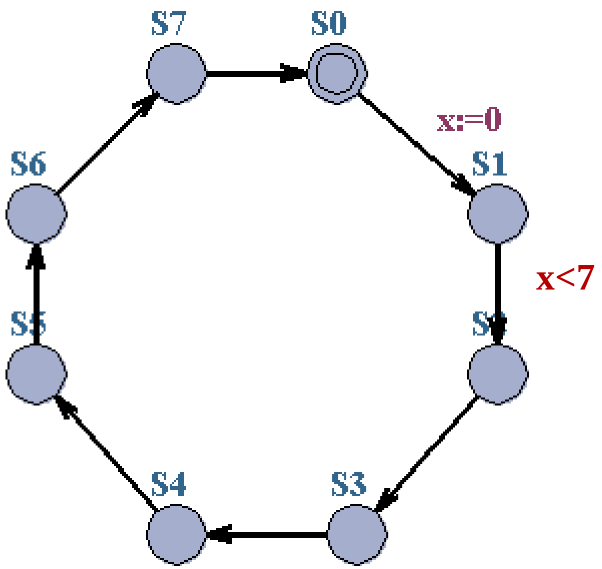
Figure 11: A railway network resembling Lyngby Station located north of Copenhagen, Denmark. Compared to the layout of Lyngby Station shown in [24], we have added four linear sections, as our model does not allow two points directly con-

VARIABLE REDUCTION

SAL SILICON AUSTRIA LABS

```
section_Id go(int tId){
int pos=position[tId];
bool dir=direction[tId];
if(pos==0 and dir==0) {crash=true; return pos;}
if(pos==0 and dir==1) {if(config_Of_Points[0]==0){
if(occupied[1]==0){
    occupied[1]=1;
    occupied[0]=0;
    return 1;
} else {
    crash = true;
     return pos;}
} else {
    if(occupied[2]==0){
        occupied[2]=1;
        occupied[0]=0;
        return 2;
    } else {
        crash = true;
         return pos;}
    }
}
if(pos==1 and dir==0) {if(occupied[0]==0){
occupied[0]=1;
occupied[1]=0;
return 0;
} else {
```

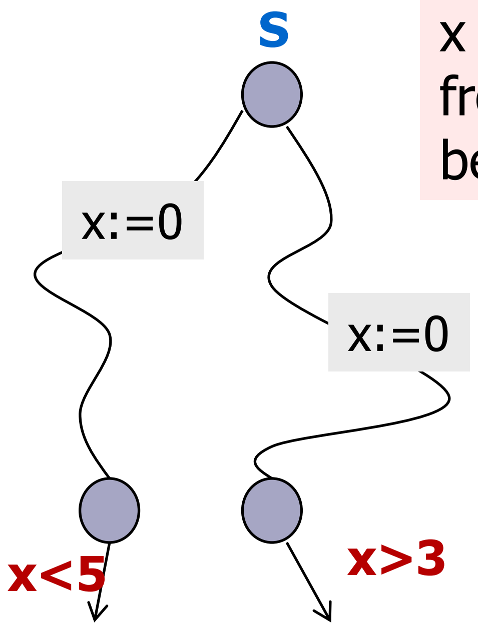
CLOCK REDUCTION (AUTOMATIC)



x is only ***active*** in location **S1**

Definition

x is ***inactive*** at **S** if on all path from **S**, x is always reset before being tested.



THINGS YOU SHOULD KNOW BY THE END OF TODAY

SAL
SILICON AUSTRIA LABS

- What is a Timed Automata?
- How is time treated in a finite way?
- Why do we need both committed and urgent locations?
- How can I check if a model can reach a certain state?

PART 2: OTHER UPPAAL BASED TOOLS

AGENDA

SAL SILICON AUSTRIA LABS

- TIMES
- Uppaal TIGA
- Uppaal CORA
- Uppaal PORT
- Uppaal cover
- Uppaal PRO
- Uppaal SMC (a.k.a. Uppaal 4.1.9)
- Uppaal Stratego
- Uppaal TRON
- ECDAR/Jecdar/Hecdar

WHY SO MANY TOOLS?

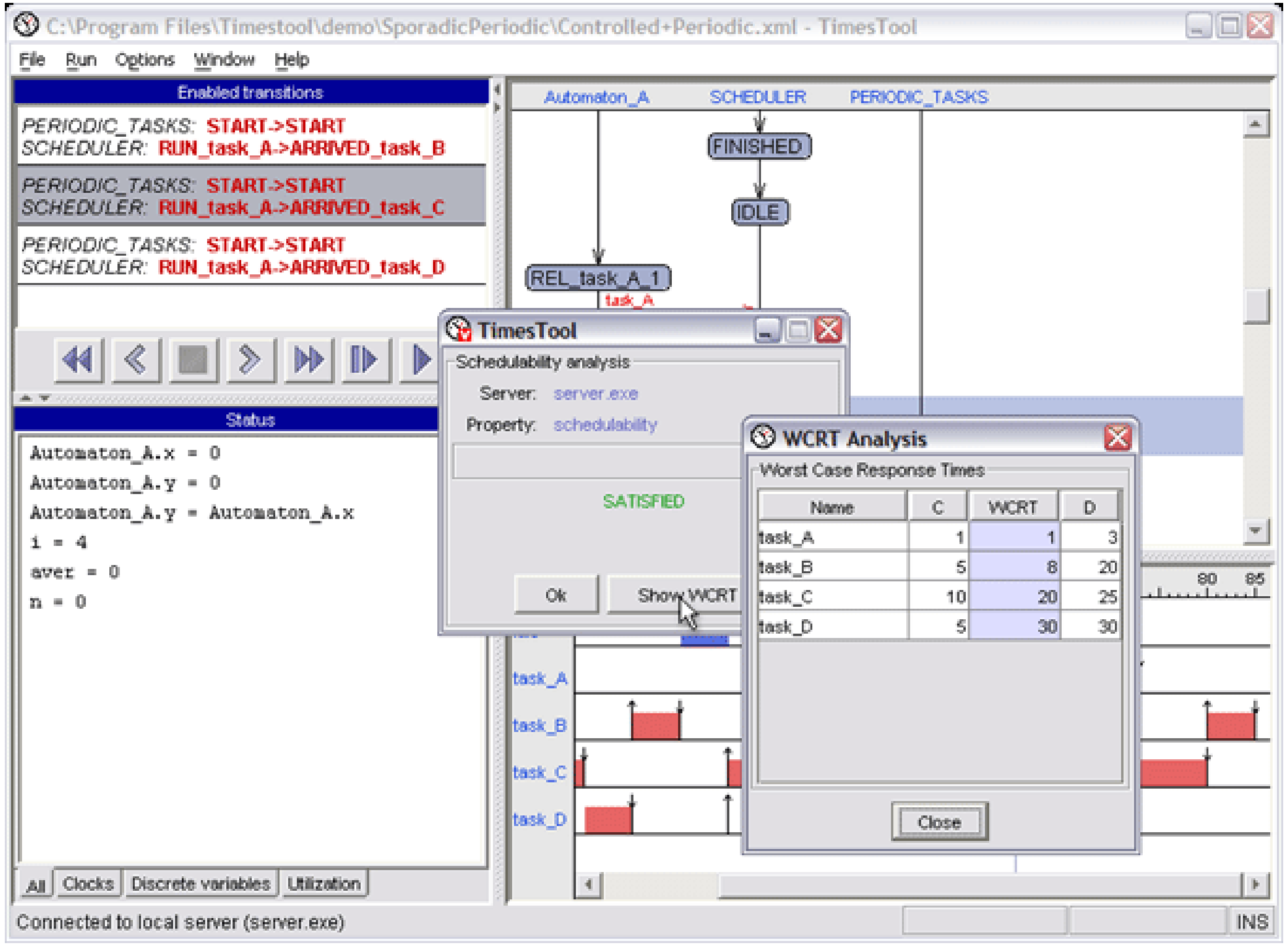
SAL
SILICON AUSTRIA LABS

- This is an academic tool
- New experiments require new tools
- **No publications in merging tools**
- No funding to hire someone for tool maintenance
- Some models may look similar, but cannot be combined

TIMES

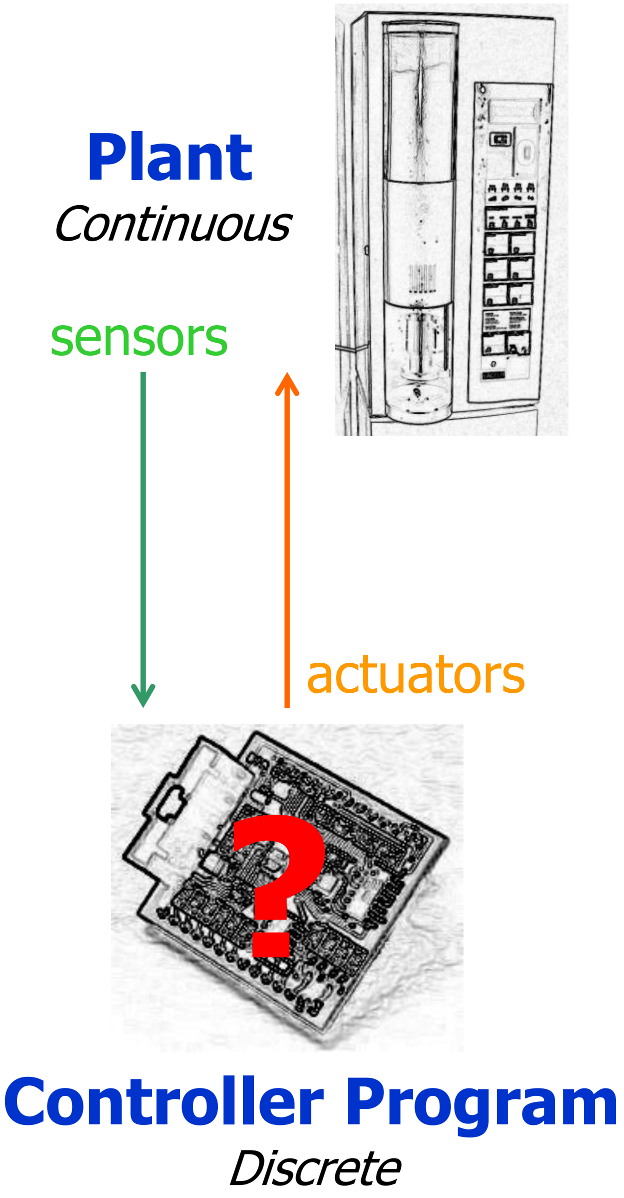
SAL
SILICON AUSTRIA LABS

- 2002
- *Disclaimer: I have never used this tool*
- Same underlying model checking engine
  - Very different modeling possibilities
- Targeted at
  - Schedulability analysis
  - Generating optimal schedules



UPPAAL TIGA

- TIGA = Timed Games
  - Two player games
  - Controllable and uncontrollable actions
- Controller synthesis:
  - Model the environment + what a controller can do.
  - Generate the controller so that controller satisfies φ!
  - Generate the right code automatically.
- 2-player timed game:
  environment moves vs. controller moves.
   Timed Game Automata.



CONTROLLER SYNTHESIS/TGA

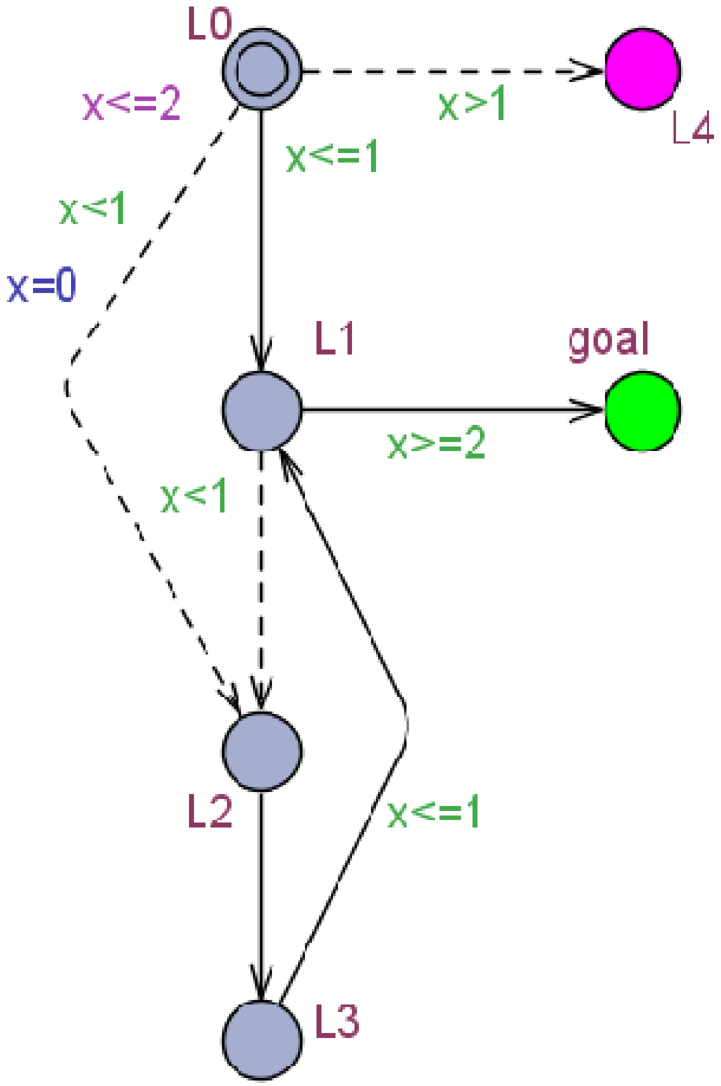
SAL SILICON AUSTRIA LABS

- Given
  - System moves S,
  - Controller moves C,
  - and a property φ,
- find
  - a strategy Sc s.t. Sc||S satisfies φ,
- or prove there is no such strategy.

- The controller continuously observes the system (all delays & moves are observable).
- The controller can
  - wait (delay action),
  - take a controllable move, or
  - prevent delay by taking a controllable move

TIMED GAME AUTOMATA

- Timed automata with controllable and uncontrollable transitions.
- Reachability & safety games.
  - control: A<> TGA.goal
  - control: A[] not TGA.L4
- Memoryless strategy:
  - state → action.



TGA – LET'S PLAY!

SAL
SILICON AUSTRIA LABS

- control: A<> TGA.goal

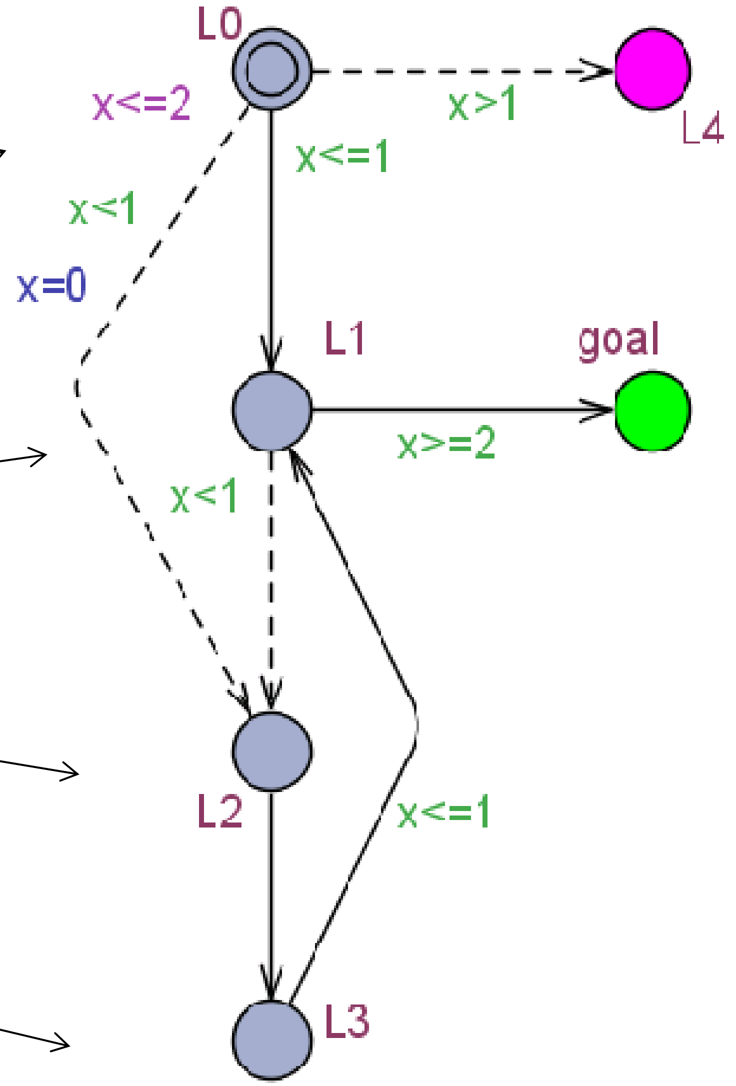
Strategy

$x<1$	: λ
$x==1$	: c

$x<2$	: λ
$x\geq 2$	: c

$x\leq 1$	: c

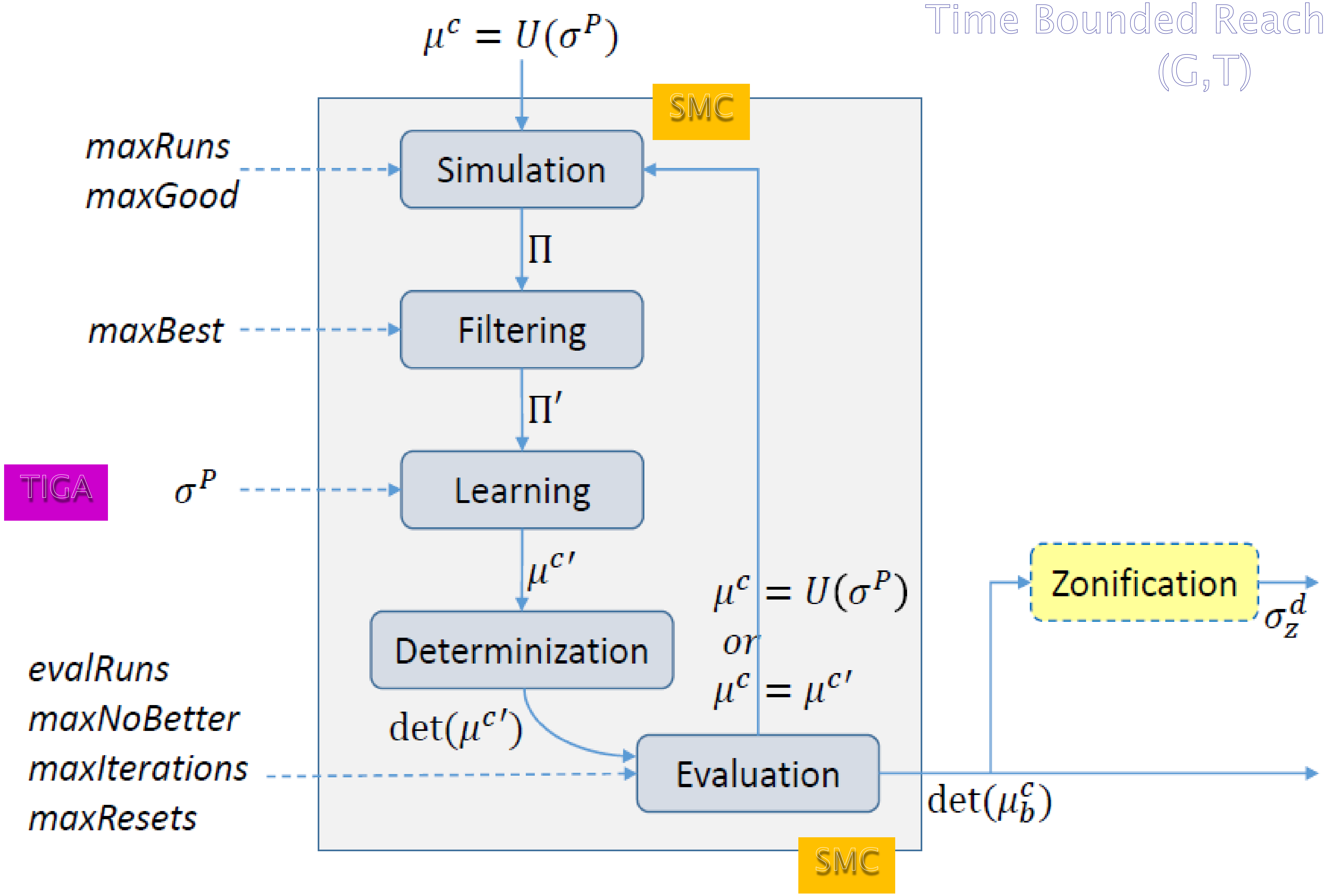
$x<1$	: λ
$x==1$	: c



Note: This is *one* strategy.
There are other solutions.

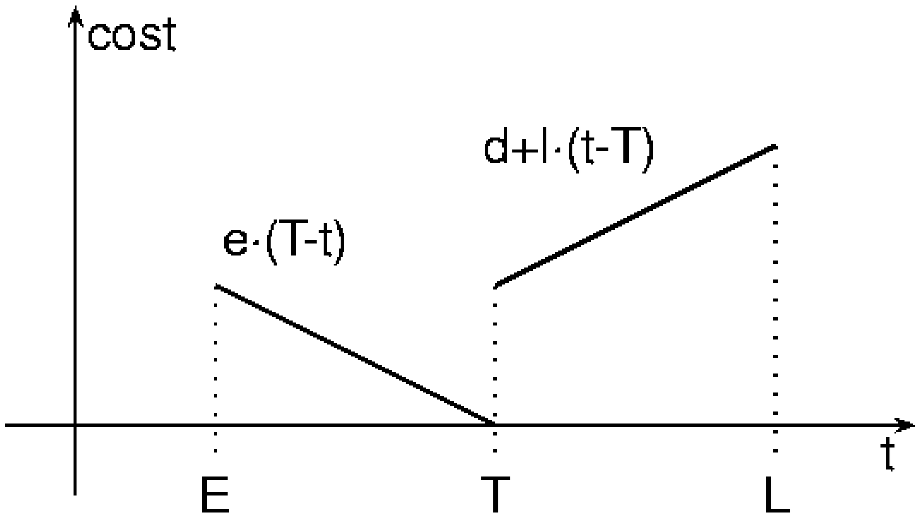
Reinforcement Learning

Time Bounded Reachability (G,T)



UPPAAL CORA

- CORA = Cost Optimal Reachability Analysis
  - UPPAAL for Planning and Scheduling
- Enables modeling LPTA
  - LPTA = Linearly priced timed automata
  - Can model e.g. energy consumption
  - Discrete costs on edges
  - Linear cost accumulating in locations



E earliest landing time

T target (cruise) landing time

L latest landing time

e early cost rate

l late cost rate

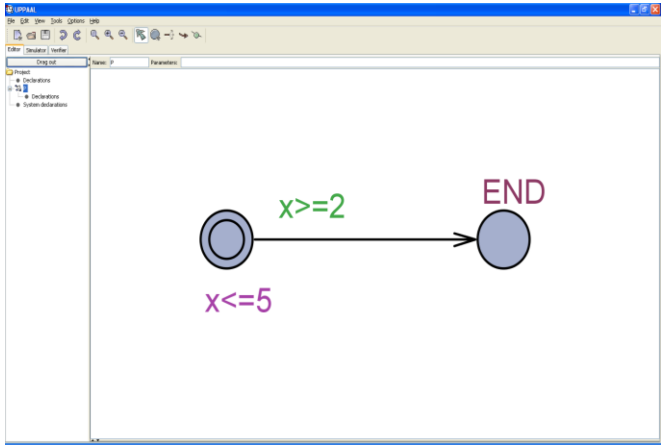
d late penalty

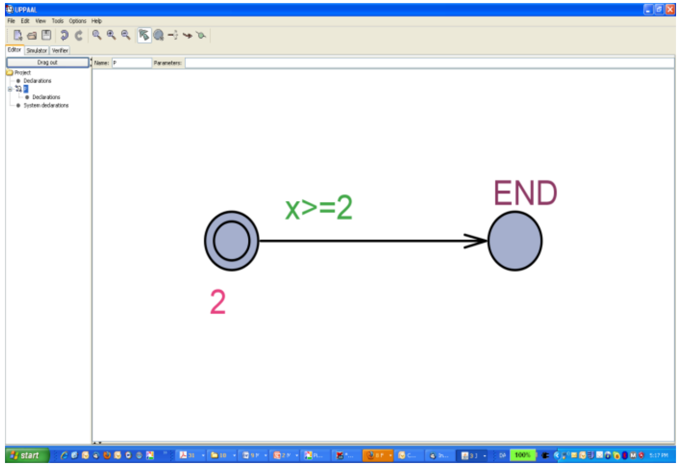
UPPAAL SMC

- Statistical model checking
- Now integrated in main UPPAAL
- This is really simulation
  - A LOT of simulation
  - With a calculated confidence level

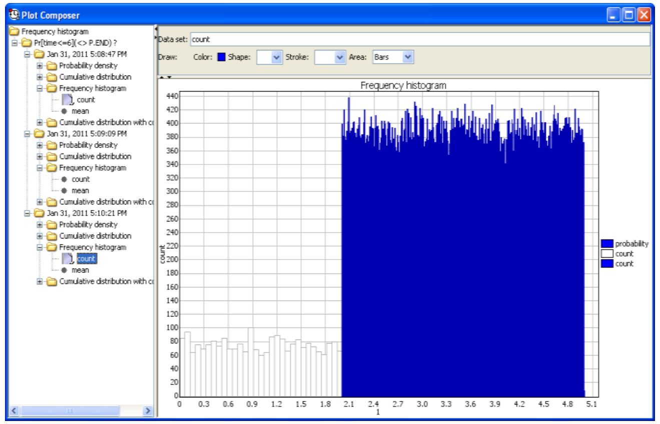
- State space explosion not a (big) problem
- Evaluation of other approaches
- Very fast compared to testing

DELAY

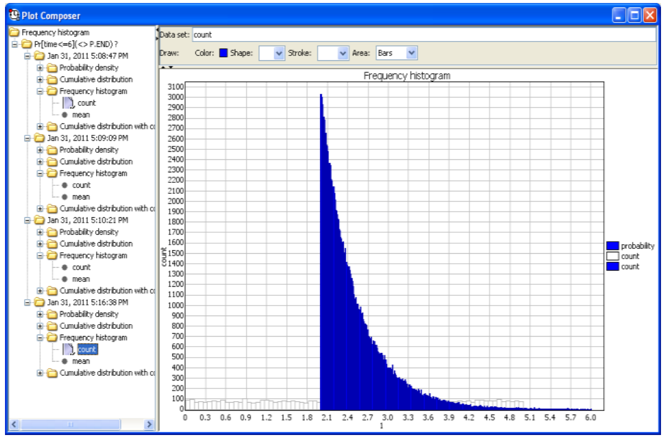




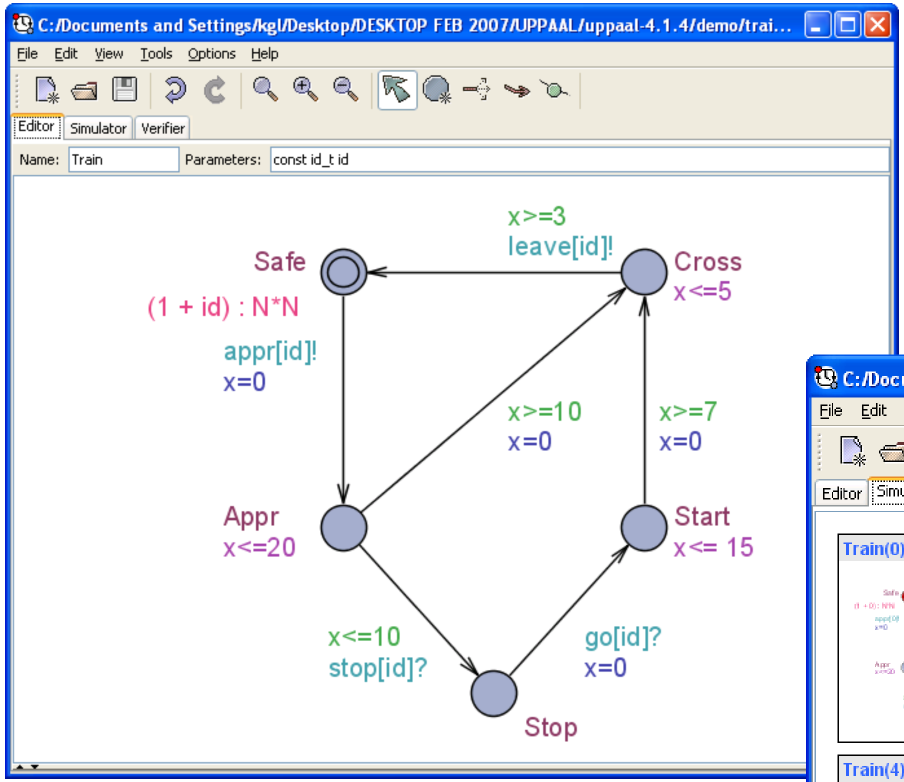
Uniform

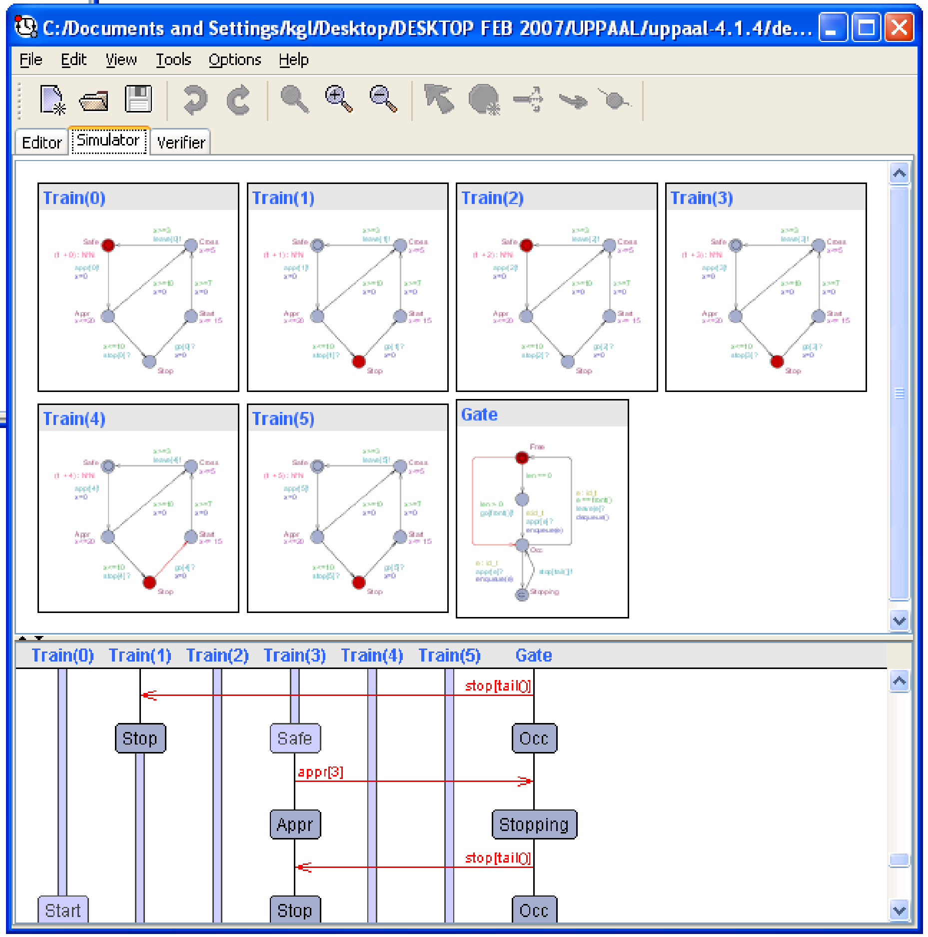


Exponential (rate)



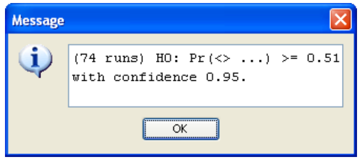
SAL SILICON AUSTRIA LABS


C:/Documents and Settings/kgl/Desktop/DESKTOP FEB 2007/UPPAAL/uppaal-4.1.4/demo/trai...
File Edit View Tools Options Help
Editor Simulator Verifier
Name: Train Parameters: const id_t id
Safe
(1 + id) : N*N
appr[id]!
x=0
x>=3
leave[id]!
Cross
x<=5
x>=10
x=0
x>=7
x=0
Appr
x<=20
Start
x<= 15
x<=10
stop[id]?
go[id]?
x=0
Stop

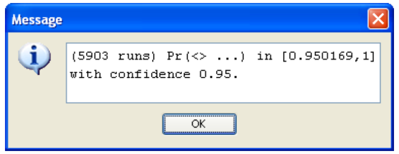

C:/Documents and Settings/kgl/Desktop/DESKTOP FEB 2007/UPPAAL/uppaal-4.1.4/de...
File Edit View Tools Options Help
Editor Simulator Verifier
Train(0) Train(1) Train(2) Train(3)
Train(4) Train(5) Gate
Train(0) Train(1) Train(2) Train(3) Train(4) Train(5) Gate
stop[tail()]
Stop
Safe
Occ
appr[3]
Appr
Stopping
stop[tail()]
Start
Stop
Occ

QUERIES

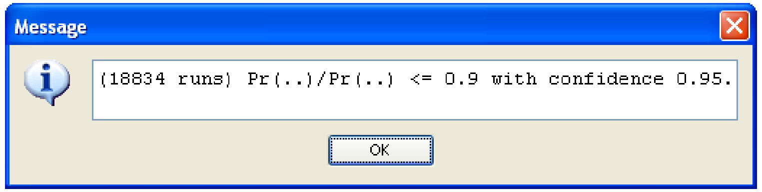
- Qualitative Check (Hypothesis Testing)
- Pr[time <= 500](<> Train(0).Cross) >= 0.5



- Quantitative Check (Estimation)
- Pr[time <= 500](<> Train(5).Cross) ?



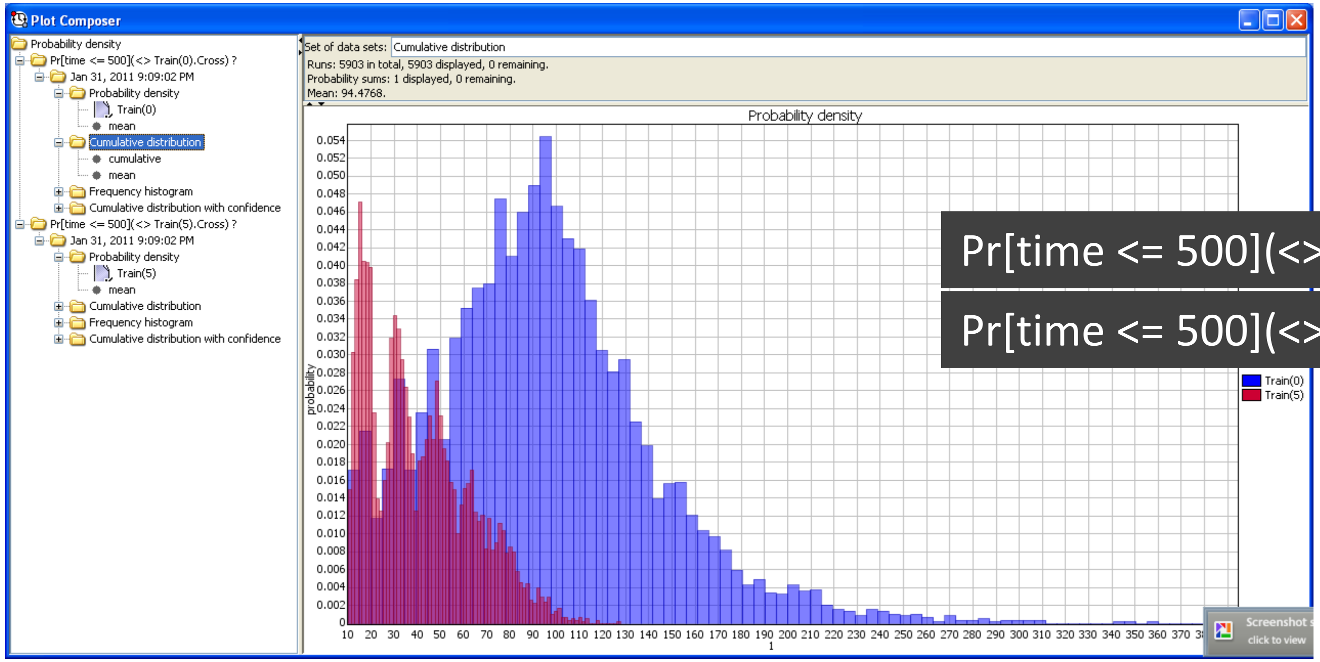
- Comparison Test
- Pr[time <= 500](<> Train(5).Cross) >=
  Pr[time <= 500](<> Train(0).Cross)

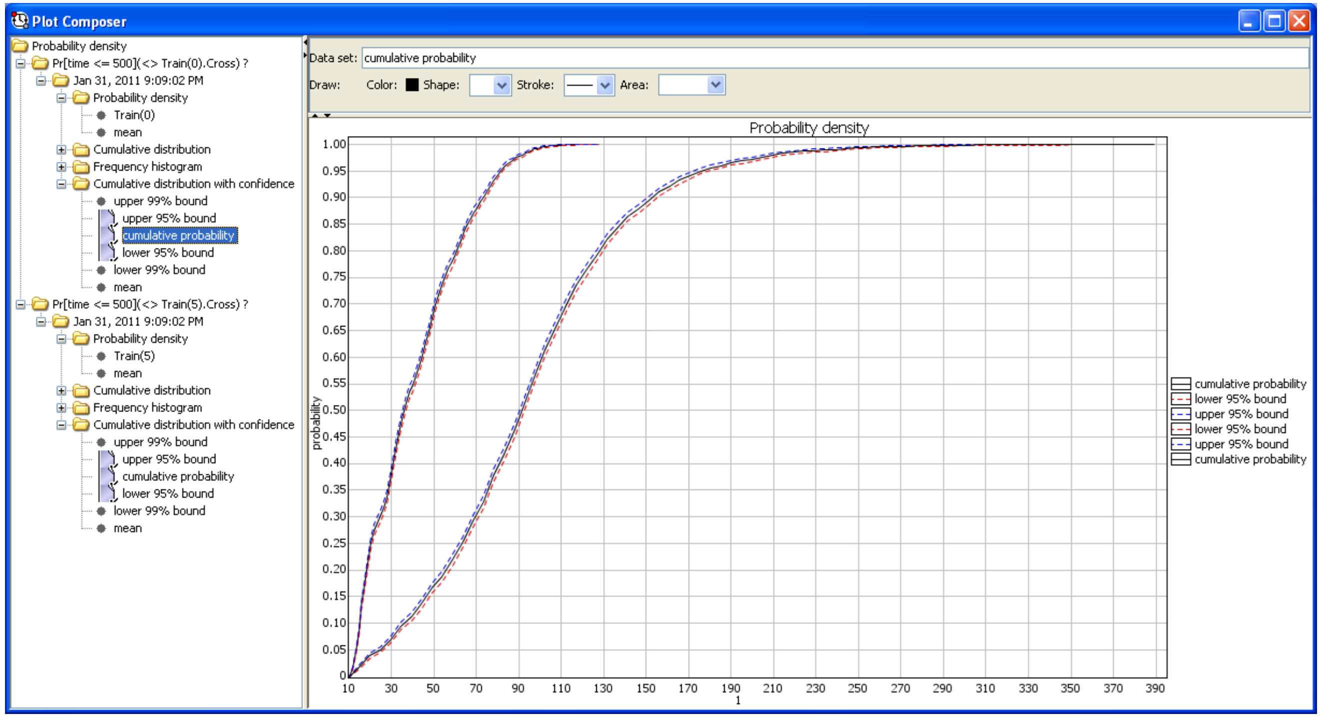


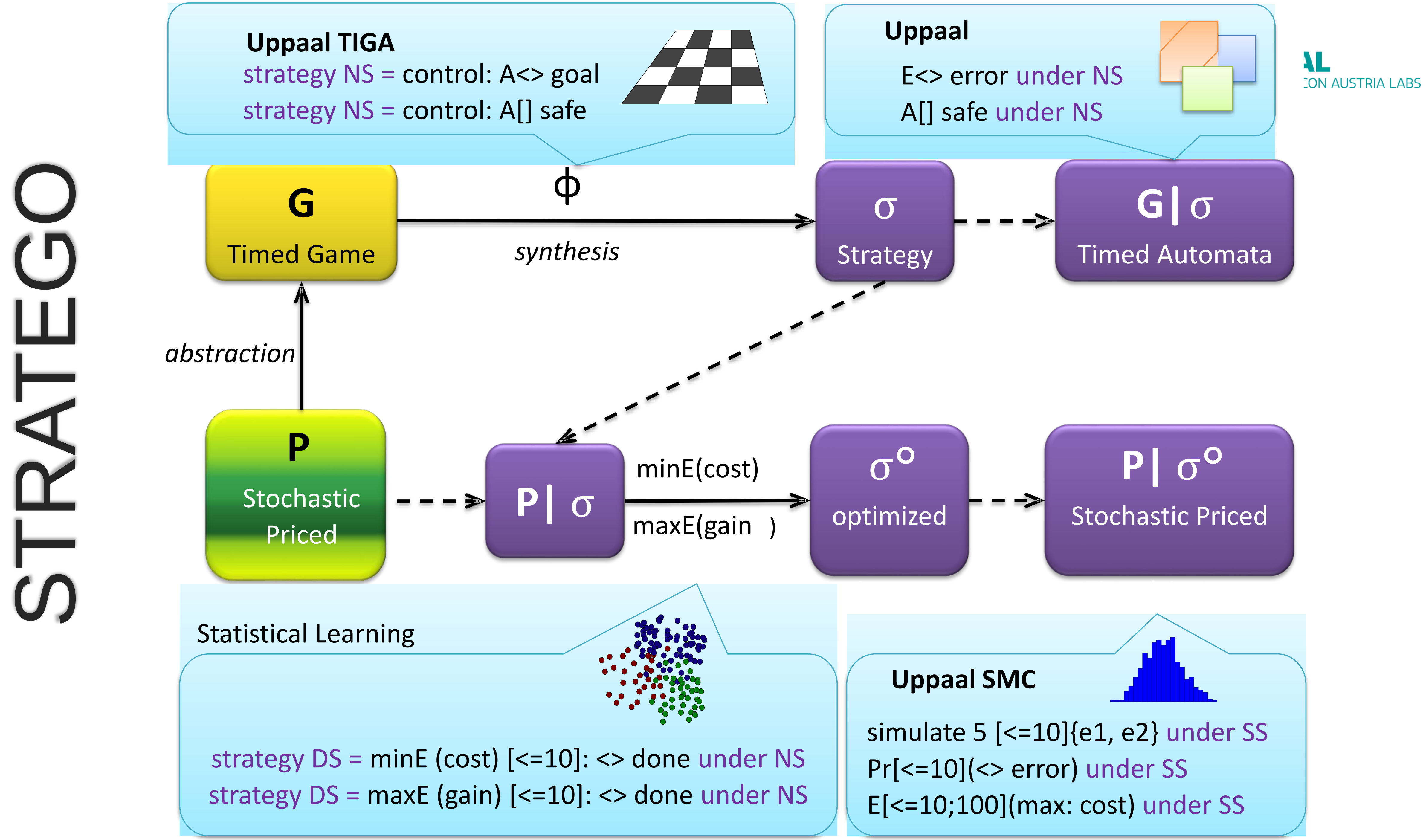
SAL SILICON AUSTRIA LABS

Pr[time <= 500](<> Train(5).Cross) ?

Pr[time <= 500](<> Train(0).Cross) ?







STRATEGO
Uppaal TIGA
strategy NS = control: A<> goal
strategy NS = control: A[] safe
Uppaal
E<> error under NS
A[] safe under NS
AL
CON AUSTRIA LABS
G
Timed Game
ϕ
synthesis
σ
Strategy
G|σ
Timed Automata
abstraction
P
Stochastic
Priced
P| σ
minE(cost)
maxE(gain)
σ°
optimized
P| σ°
Stochastic Priced
Statistical Learning
strategy DS = minE (cost) [<=10]: <> done under NS
strategy DS = maxE (gain) [<=10]: <> done under NS
Uppaal SMC
simulate 5 [<=10]{e1, e2} under SS
Pr[<=10](<> error) under SS
E[<=10;100](max: cost) under SS

HOMEWORK: PACMAN

- Create a network of timed automata
  - Ghost and Pacman
  - 3x3 grid, starting on opposite ends
  - Make the ghost edges not controllable
  - Both can stay at most 5 time units on the same field
  - Leaving a field can be done after a minimum of 2 time units

- Queries:
  - Can Pacman and Ghost be on the same grid?
  - Will they always be on different grids?
  - Can you make a strategy that lets the Pacman always escape? => if so, simulate via concrete simulator

- Allow pacman to stay for 50 instead of 5 time units, repeat the queries.
- Add a second ghost template (parametrized?)

- Hints: grid as locations or via variables? Either way, having a global variable with the current location makes the query a lot easier.
- You can combine select field and function for guard
- Use the demo folder of UPPAAL to figure out the right syntax of elements ;)

SOME EXAMPLES

- Grundfos pump controller
- Train station
- Brick Sorter
- RT OS

SAL
SILICON AUSTRIA LABS

UNFOLD THE FUTURE

WWW.SILICON-AUSTRIA-LABS.COM