

Automated Planning Tools for Intelligent Decision Making



AALBORG UNIVERSITET

Planning

Planning

From Wikipedia, the free encyclopedia

"Forethought" redirects here. For the defunct software company, see [Forethought, Inc.](#)

Planning (also called **forethought**) is the process of thinking about and organizing the activities required to achieve a desired goal. It involves the creation and maintenance of a [plan](#), such as psychological aspects that require conceptual skills. There are even a couple of tests to measure someone's capability of planning well. As such, planning is a fundamental property of intelligent behavior.

Also, planning has a specific process and is necessary for multiple occupations (particularly in fields such as [management](#), [business](#), etc.). In each field there are different types of plans that help companies achieve efficiency and effectiveness. An important, albeit often ignored aspect of planning, is the relationship it holds to [forecasting](#). Forecasting can be described as predicting what the future will look like, whereas planning predicts what the future should look like for multiple scenarios. Planning combines forecasting with [preparation](#) of scenarios and how to react to them. Planning is one of the most important project management and time management techniques. Planning is preparing a sequence of action steps to achieve some specific goal. If a person does it effectively, they can reduce much the necessary time and effort of achieving the goal. A plan is like a map. When following a plan, a person can see how much they have progressed towards their project goal and how far they are from their destination.

Scheduling

In **computing**, **scheduling** is the method by which work specified by some means is assigned to resources that complete the work. The work may be virtual computation elements such as **threads**, **processes** or data **flows**, which are in turn scheduled onto hardware resources such as **processors**, **network links** or **expansion cards**.

A scheduler is what carries out the scheduling activity. Schedulers are often implemented so they keep all computer resources busy (as in **load balancing**), allow multiple users to share system resources effectively, or to achieve a target **quality of service**. Scheduling is fundamental to computation itself, and an intrinsic part of the **execution model** of a computer system; the concept of scheduling makes it possible to have **computer multitasking** with a single **central processing unit** (CPU).

A scheduler may aim at one of many goals, for example, maximizing **throughput** (the total amount of work completed per time unit), minimizing **response time** (time from work becoming enabled until the first point it begins execution on resources), or minimizing **latency** (the time between work becoming enabled and its subsequent completion),^[1] maximizing **fairness** (equal CPU time to each process, or more generally appropriate times according to the priority and workload of each process). In practice, these goals often conflict (e.g. throughput versus latency), thus a scheduler will implement a suitable compromise. Preference is given to any one of the concerns mentioned above, depending upon the user's needs and objectives.

In **real-time** environments, such as **embedded systems** for **automatic control** in industry (for example **robotics**), the scheduler also must ensure that processes can meet **deadlines**; this is crucial for keeping the system stable. Scheduled tasks can also be distributed to remote devices across a network and **managed** through an administrative back end.

Topics / People

I. Planning via Model Checking

Kim Larsen, Peter Gjøøl Jensen



II. Planning via Operations Research

Peter Nielsen, Mohamed El Yafrani, Inkyung Sung



III. Planning via AI

Alvaro Torralba



Kim Larsen [4]

Planning via **Model Checking**



AALBORG UNIVERSITET

“Plan”

- Timed Automata
 - Model Checking
 - Time-optimal Planning
 - Zones
- Priced Timed Automata
 - Cost-optimal Planning
 - Priced Zones A*
- Timed Games
 - Dynamic Planning
 - Strategies, Zones
- Stochastic Timed Automata
 - Performance Analysis
 - Statistical Model Checking
- Stochastic Priced Timed Games
 - Expected Cost Optimal Adaptive Planning
 - Strategies
 - Reinforcement Learning
- Applications



Verification

CLASSIC

1995

Optimization

CORA

2001

Synthesis

TIGA

2005

Component

ECDAR

2010

Testing

TRON

2004

Performance
Analysis

SMC

2011

2014

STRATEGO



Timed Automata

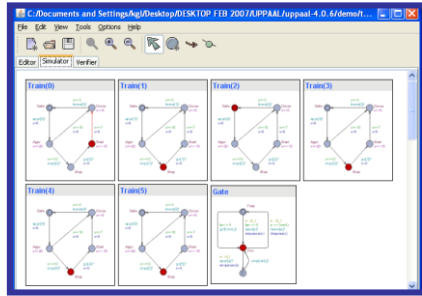
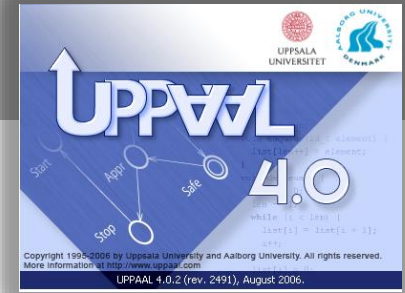
Model Checking



AALBORG UNIVERSITET



QUANTITATIVE Model Checking



Time

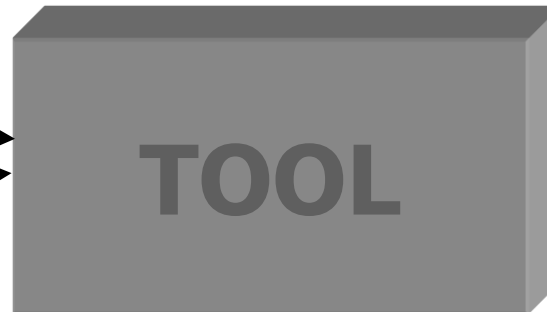


Cost



Probability

System Description



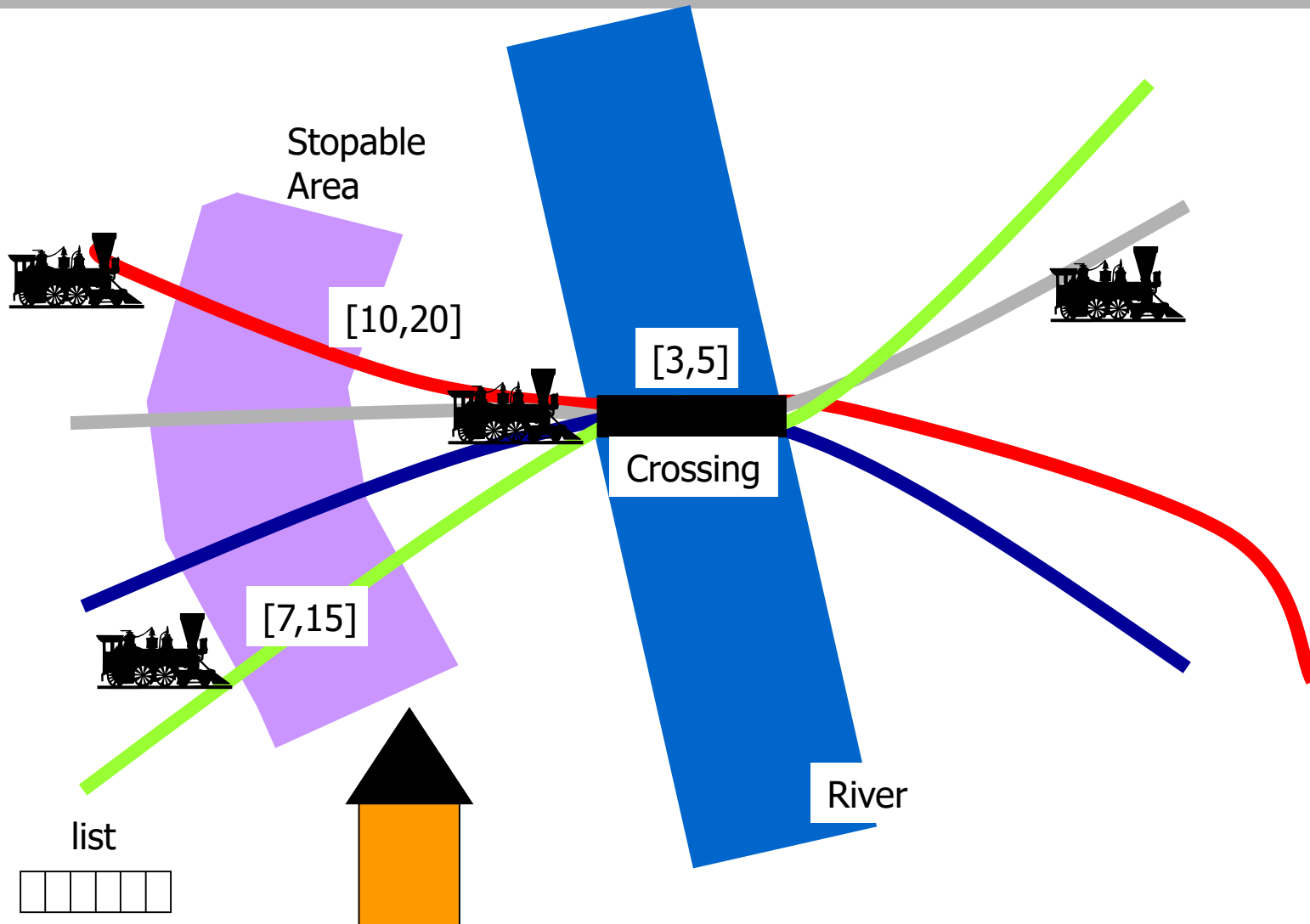
Requirement

$A[](\text{req} \Rightarrow A \nleftrightarrow \text{grant})$

No!
Debugging Information

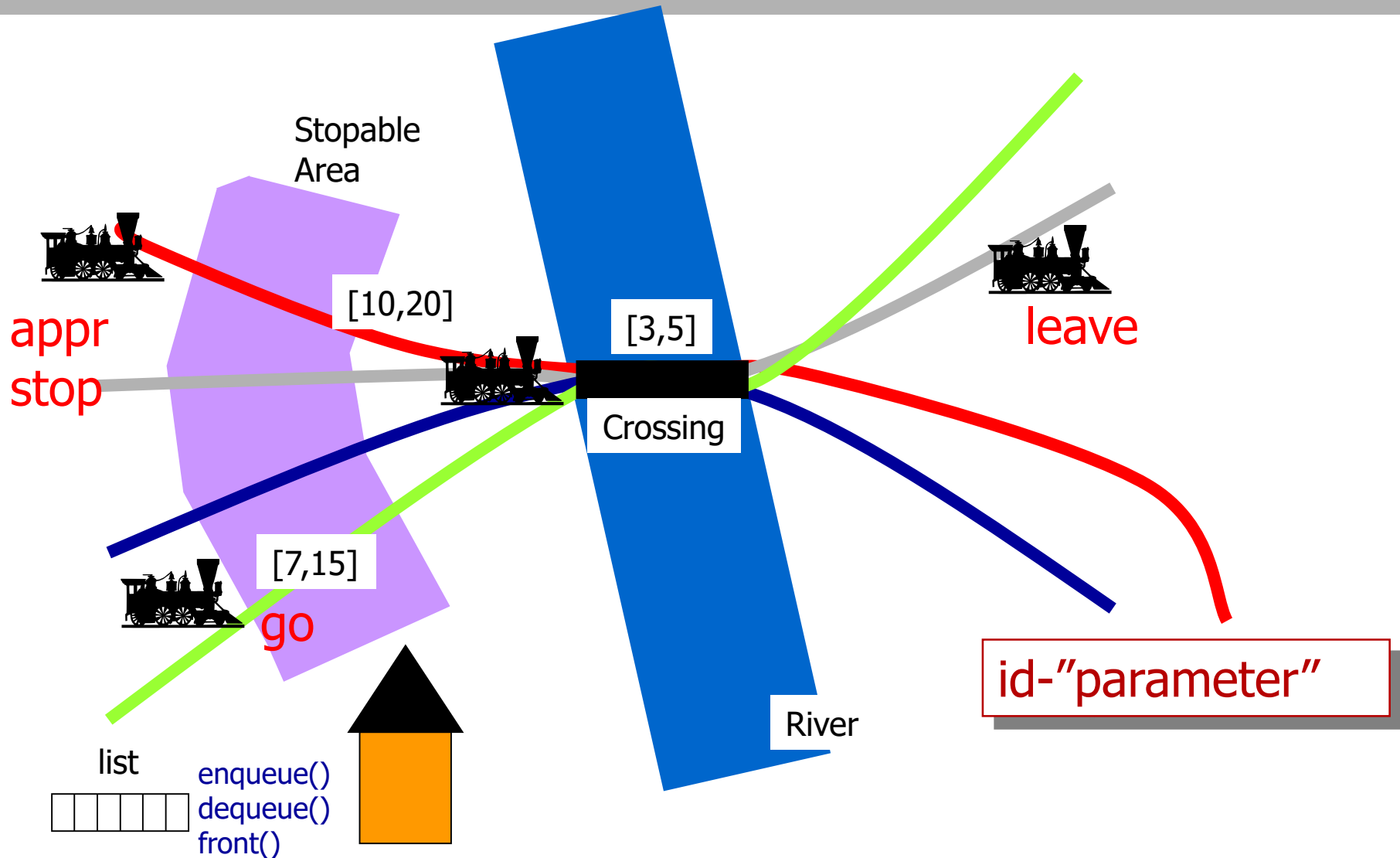
Yes
Prototypes
Executable Code
Test sequences

Train Crossing



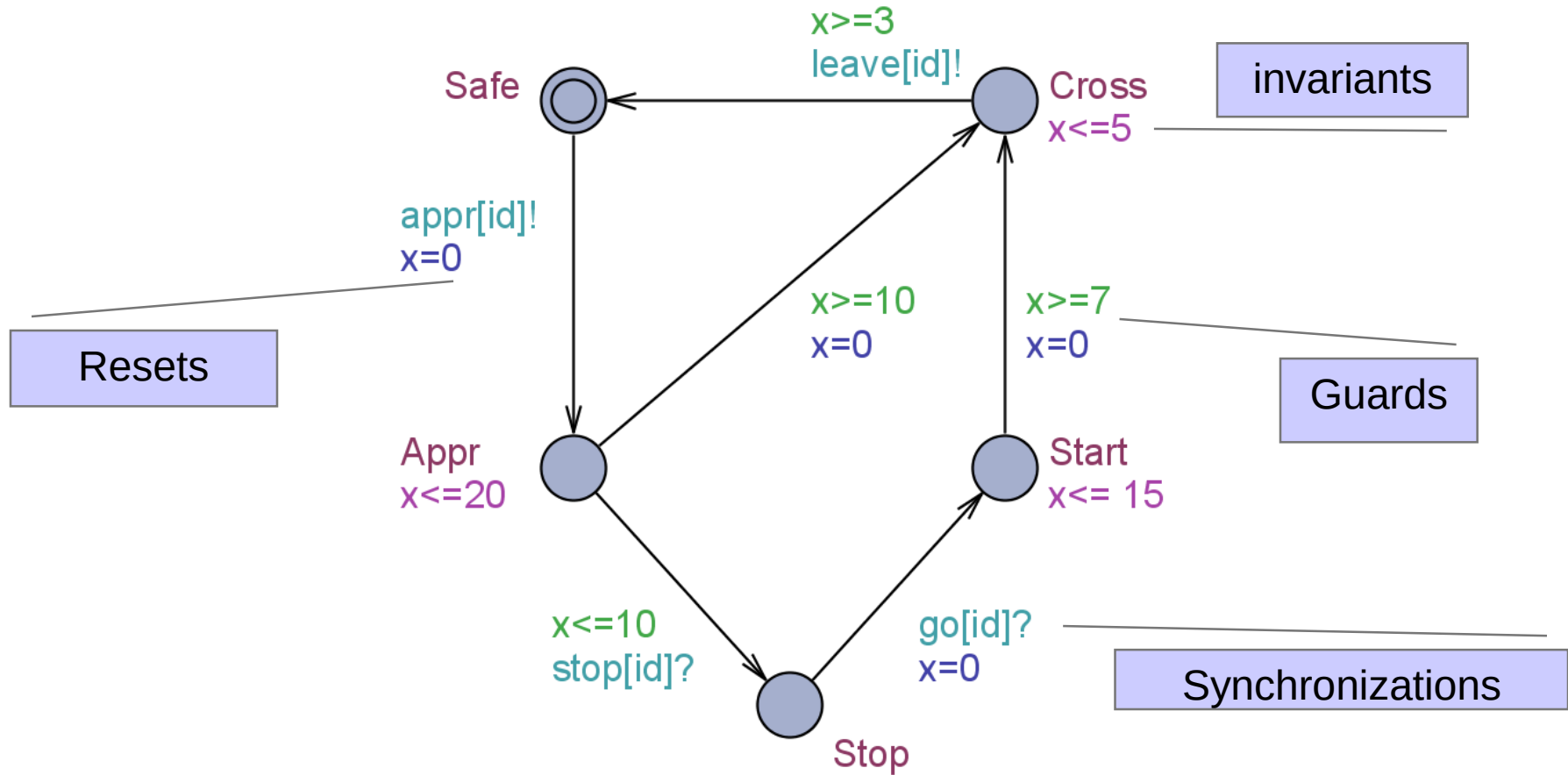
Train Crossing

Communication via **channels**!



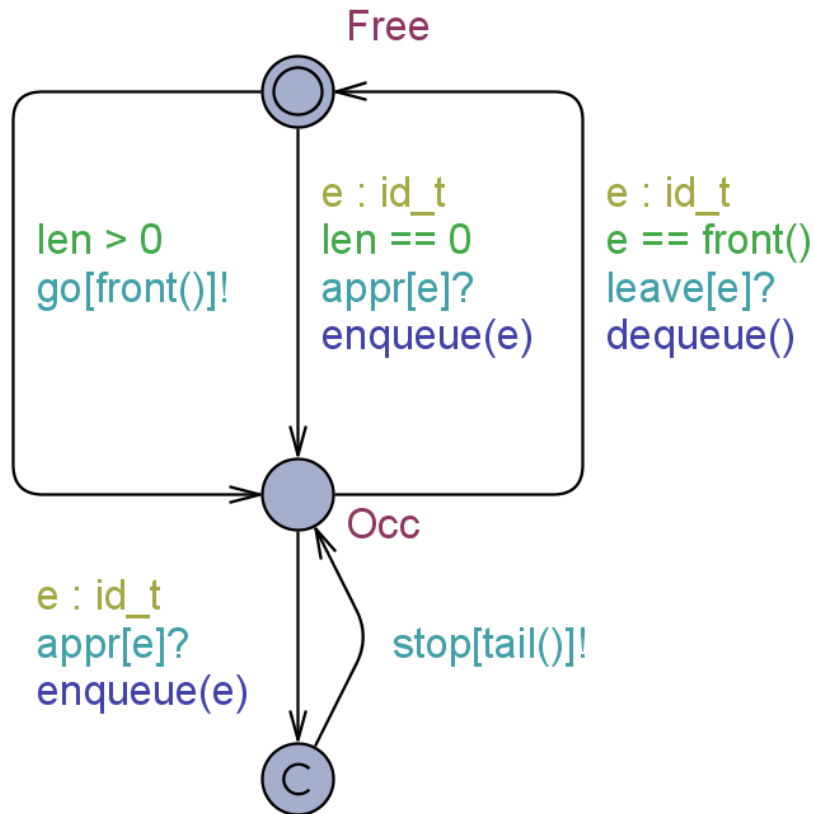
Timed Automata [Train]

= Finite State Control
+ Real Valued Clocks



Timed Automata [Gate]

= Finite State Control
+ Real Valued Clocks
+ Discrete Variables



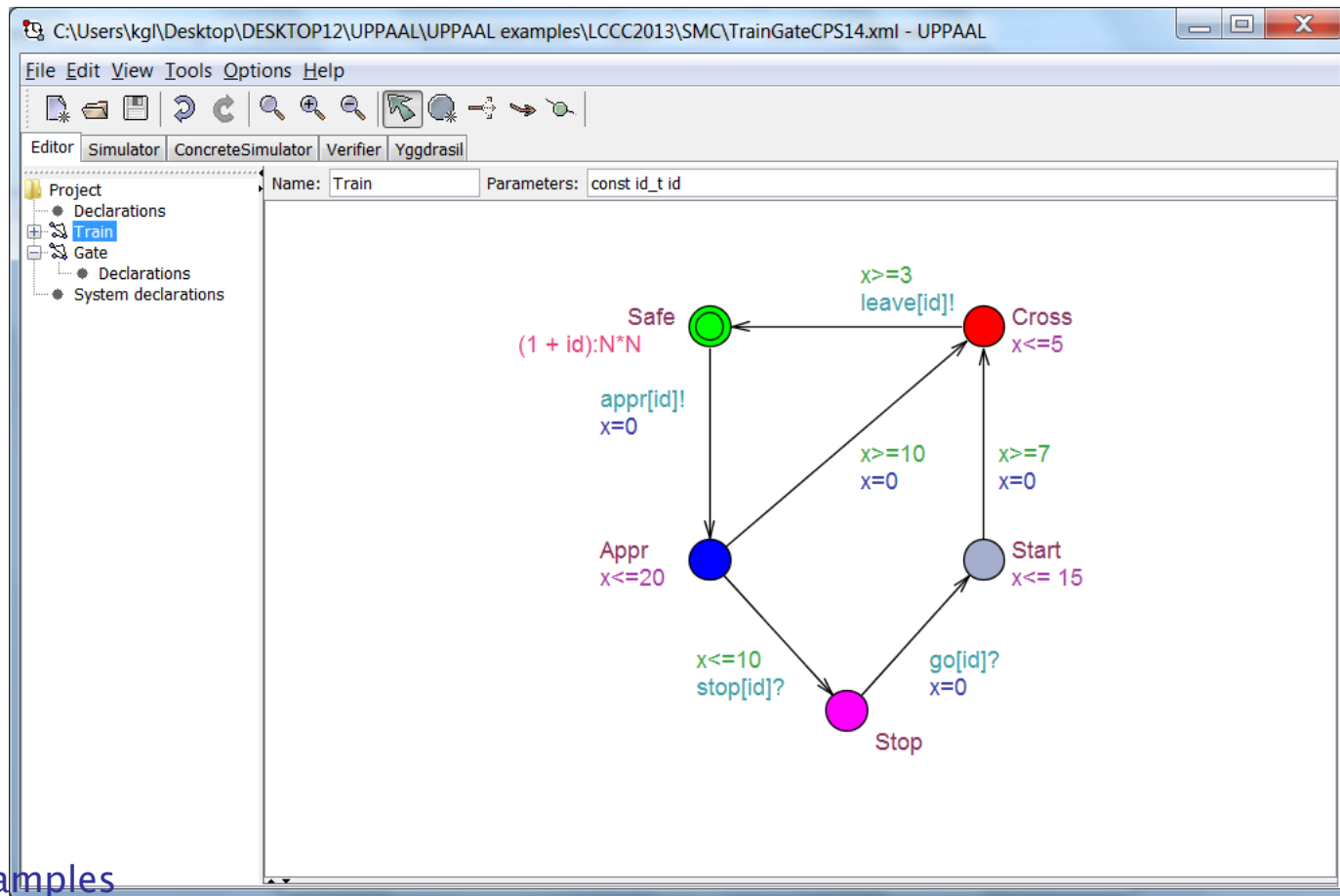
```

id_t list[N+1];
int[0,N] len;

// Put an element at the end of the queue
void enqueue(id_t element)
{
    list[len++] = element;
}

// 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;
}
  
```

UPPAAL DEMO



UPPAAL Examples
SSFT15/UPPAAL SMC/

Timed Automata

Time-Optimal Scheduling

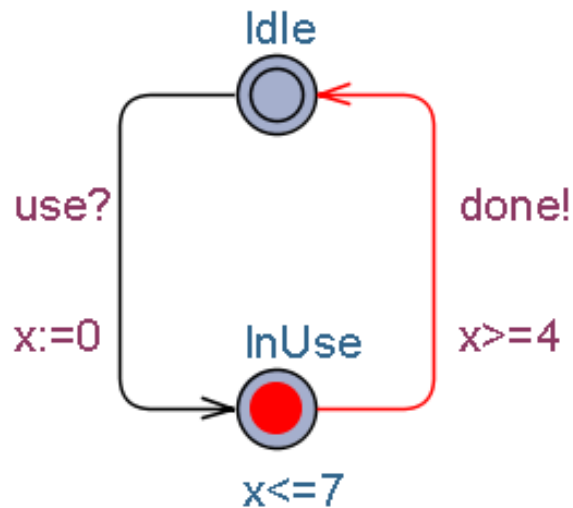


AALBORG UNIVERSITET



Resources & Tasks

Resource



Semantics:

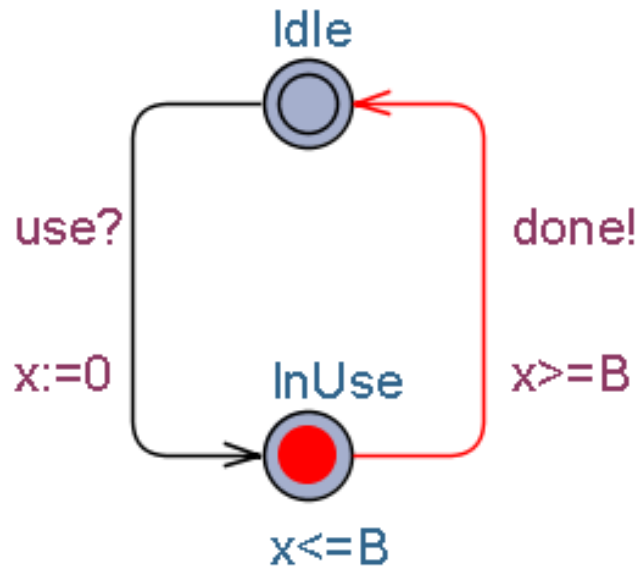
(Idle , x=0)

- (Idle , x=2.5)
- (InUse , x=0)
- (InUse , x=5)
- (Idle , x=5)
- (Idle , x=8)
- (InUse , x=0)

d(2.5)
use?
d(5)
done!
d(3)
use?

Resources & Tasks

Resource



Shared variable

Synchronization

Task



Resources & Tasks

Semantics:

(Idle , Init , B=0, x=0)

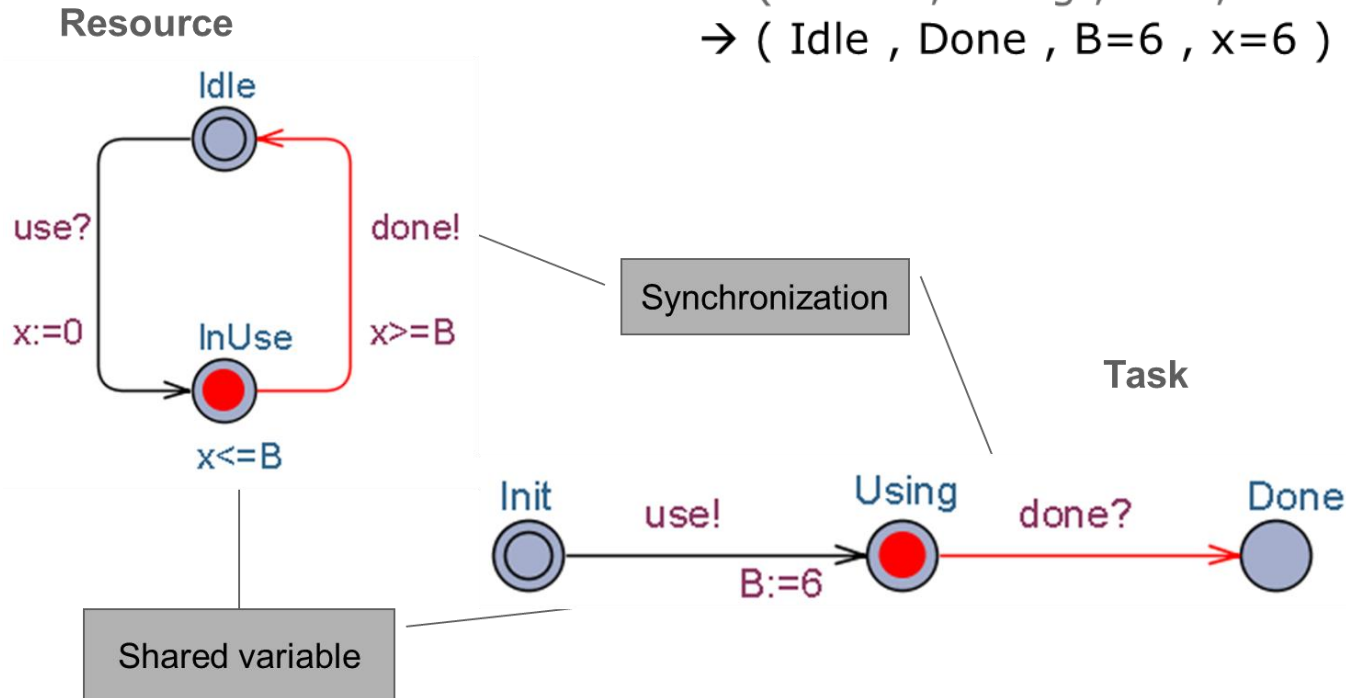
→ (Idle , Init , B=0 , x=3.1415)

→ (InUse , Using , B=6, x=0)

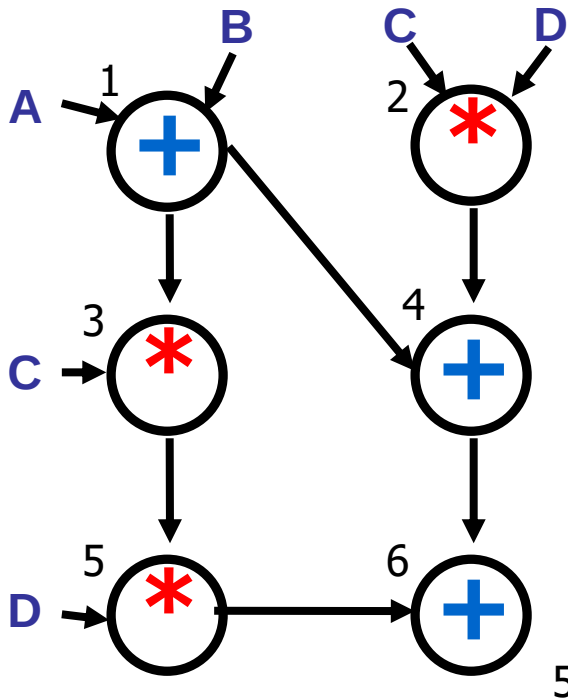
→ (InUse , Using , B=6, x=6)

→ (Idle , Done , B=6 , x=6)

d(3)
use
d(6)
done



Task Graph Scheduling – Example



Compute :

$$(D * (C * (A + B)) + ((A + B) + (C * D)))$$

using 2 processors

P1 (fast)

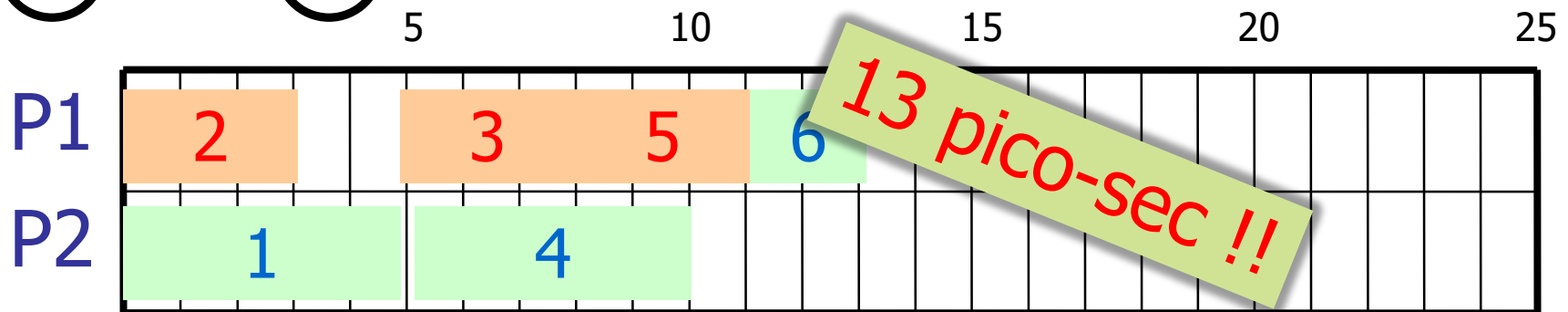


+	2ps
*	3ps

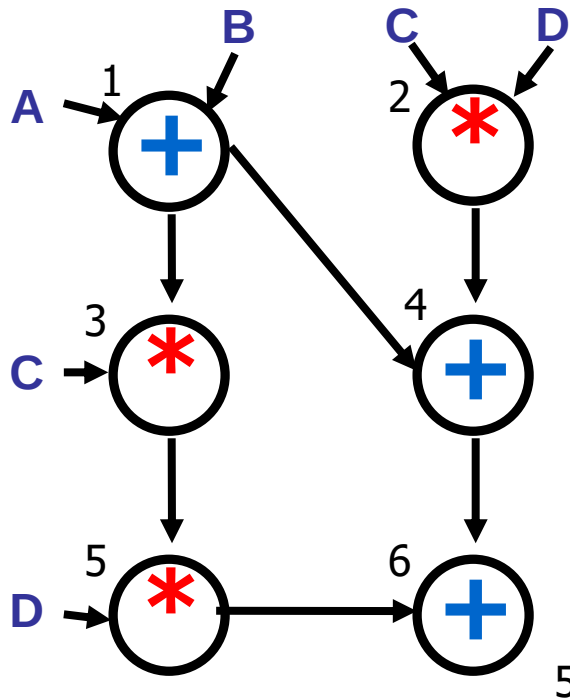
P2 (slow)



+	5ps
*	7ps



Task Graph Scheduling – Example



Compute :
 $(D * (C * (A + B)) + ((A + B) + (C * D)))$

using 2 processors

P1 (fast)

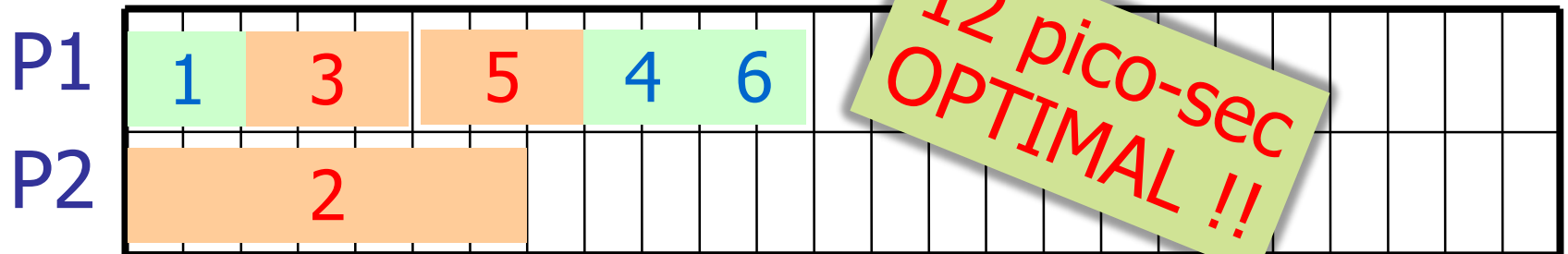


+	2ps
*	3ps

P2 (slow)

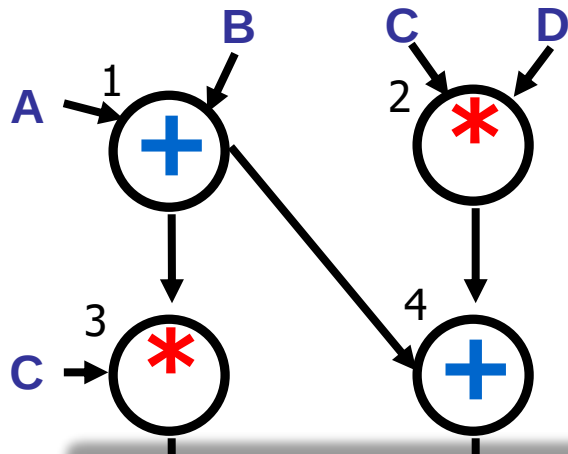


+	5ps
*	7ps

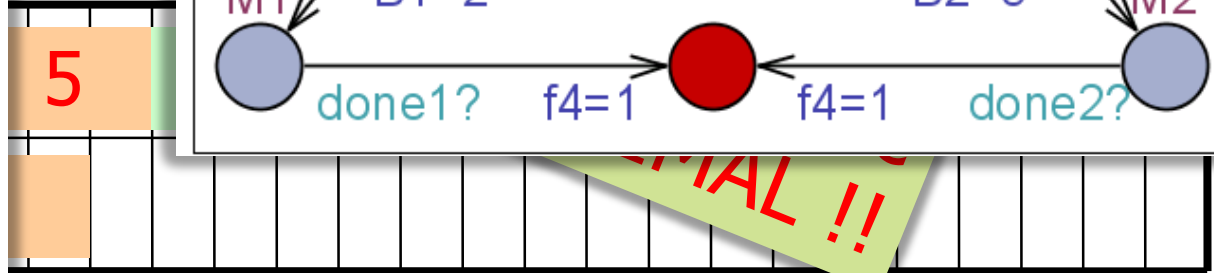
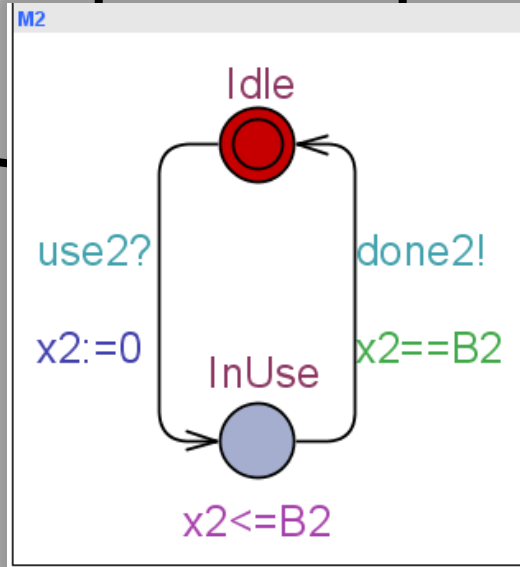
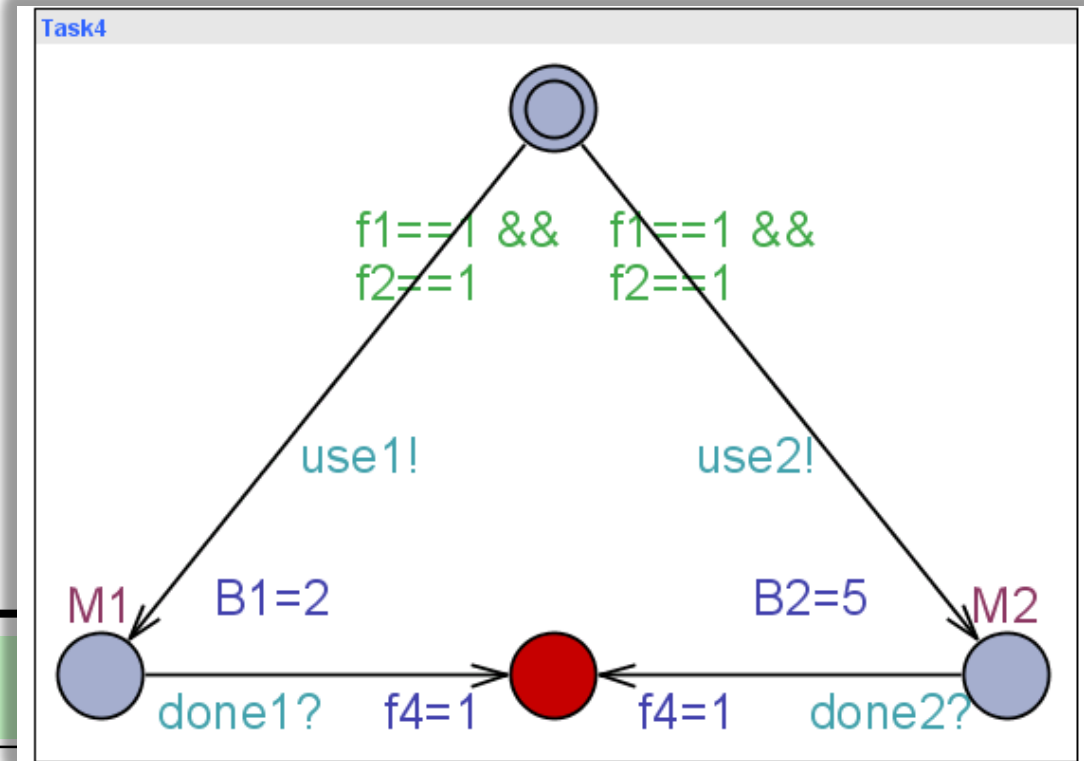


**12 pico-sec
OPTIMAL !!**

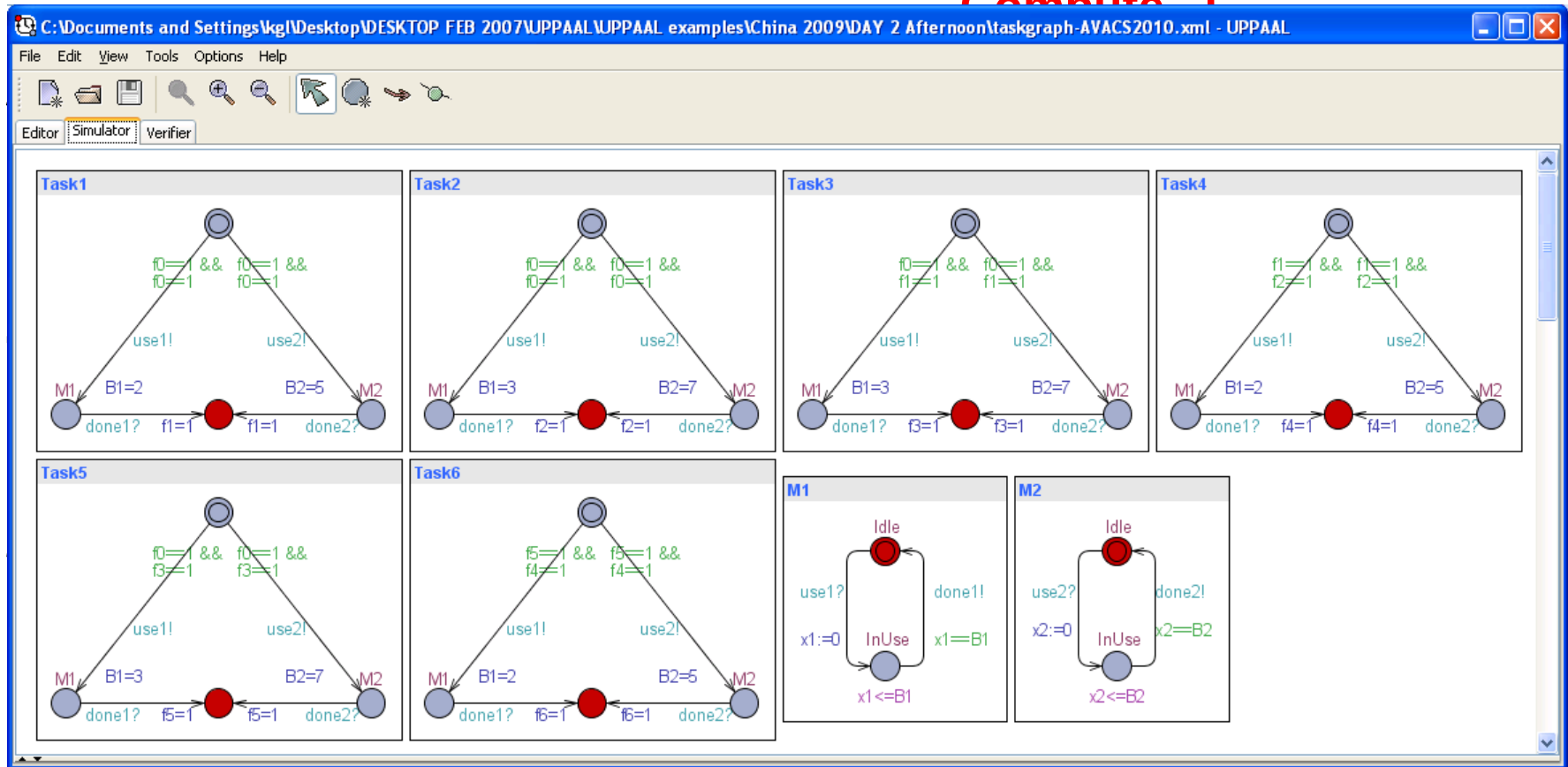
Task Graph Scheduling – Example



Compute :
 $(D * (C * (A + B))) + ((A + B) + (C * D))$



Task Graph Scheduling – Example



P2

E<> (Task1.End and ... and Task6.End)

time

Experimental Results

name	#tasks	#chains	# machines	optimal	TA
001	437	125	4	1178	1182
000	452	43	20	537	537
018	730	175	10	700	704
074	1007	66	12	891	894
021	1145	88	20	605	612
228	1187	293	8	1570	1574
071	1193	124	20	629	634
271	1348	127	12	1163	1164
237	1566	152	12	1340	1342
231	1664	101	16	t.o.	1137
235	1782	218	16	t.o.	1150
233	1980	207	19	1118	1121
294	2014	141	17	1257	1261
295	2168	965	18	1318	1322
292	2333	318	3	8009	8009
298	2399	303	10	2471	2473



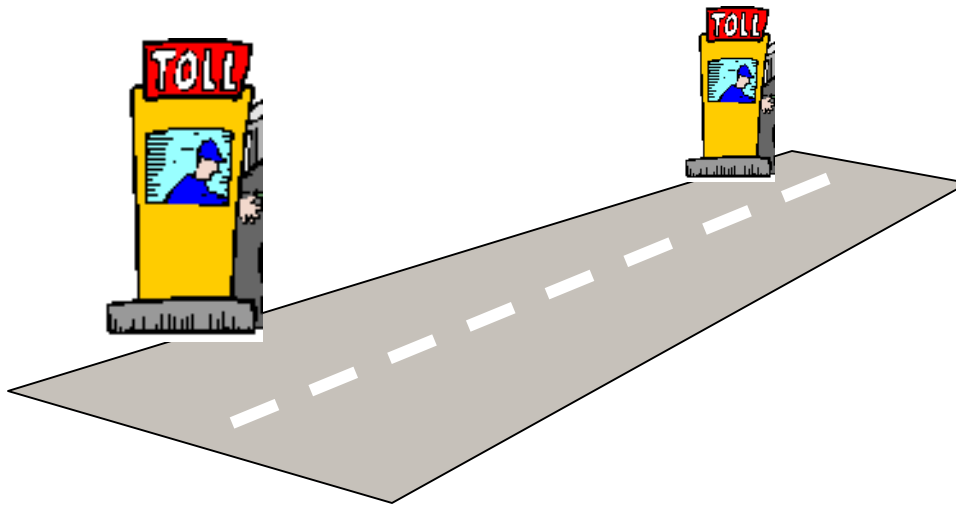
Symbolic A*
Branch-&-Bound
60 sec

Abdeddaïm, Kerbaa, Maler



Real Time Scheduling

- Only 1 "Pass"
- Cheat is possible
(drive close to car with "Pass")



UNSAFE



5

Crossing
Times



10



Pass



20



25

SAFE

**CAN THEY MAKE IT TO SAFE
WITHIN 70 MINUTES ???**



Let us play!

Solving scheduling problems using Uppaal

A number of cars are to pass a bridge. There is a toll for passing the bridge -- and a device (known as the 'BroBizz' or 'EasyPass') must be used in order to pass the bridge.

There is only one BroBizz available to the cars -- but luckily the toll booth system can be *cheated* if two cars drive close to each other. Only cars from the side at which BroBizz is located can pass the bridge. The toll booth at the side at which the BroBizz is located is colored green.

All cars must pass the bridge within a given time limit (shown at the center of the screen). Each car spends a given number of minutes passing the bridge. This *schedulability* problem can be solved using the Uppaal tool.

Using the buttons above you can:

- **Configure:**
Setup the number of cars, their speed, and the time limit
- **Interact:**
Try to solve the problem manually
- **Find some solution:**
Use Uppaal to solve the problem and display the solution
- **Find best solution:**
Use Uppaal to find the best solution to the problem

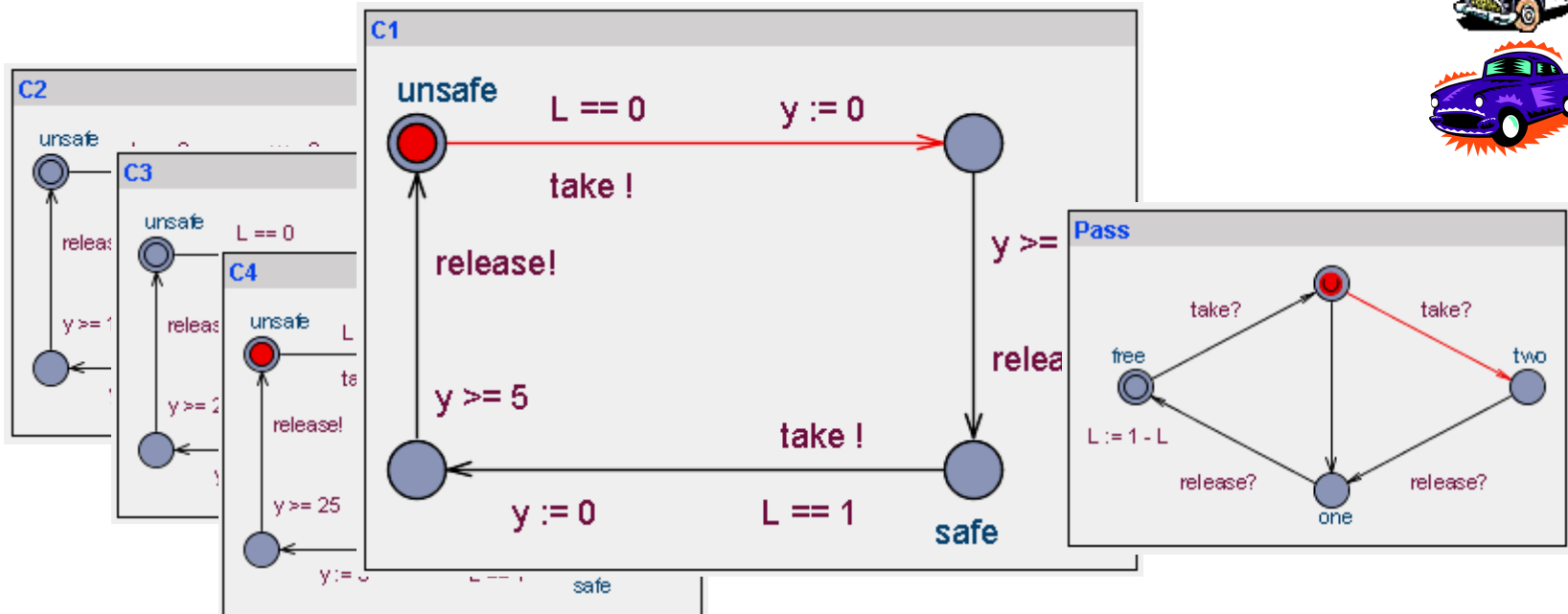
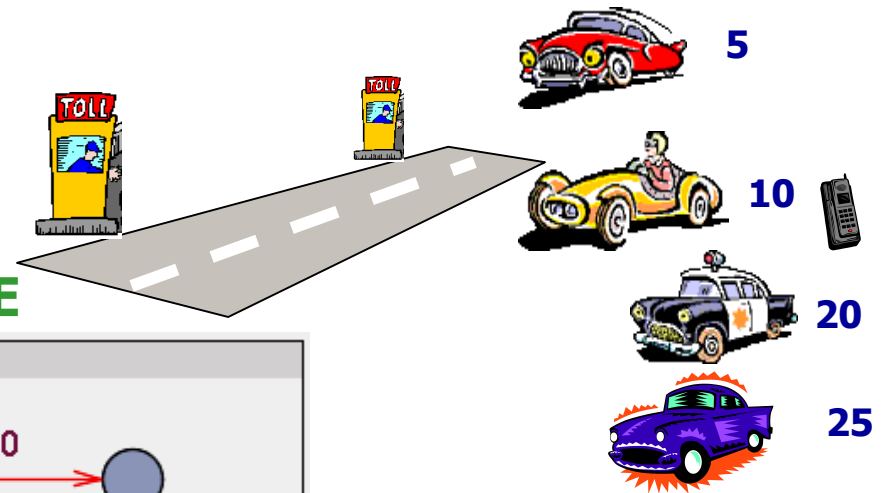
Real Time Scheduling

RIO12/DAY3/

Solve
Scheduling Problem
using **UPPAAL**

UNSAFE

SAFE



The "secret" of UPPAAL

The screenshot displays the UPPAAL software interface. The main window shows a simulation trace for a system with components Gate, Train(0), Train(1), Train(2), Train(3), Train(4), and Train(5). The trace includes events like `go[front()]: Gate → Train(5)`, `leave[1]: Train(1) → Gate[1]`, and `appr[0]: Train(0) → Gate[0]`. A list of constraints is shown, including `Train(4).x ∈ [23,60]`, `Train(5).x ∈ [30,65]`, and various time and position constraints between trains and the gate. The interface also features a menu bar (File, Edit, View, Tools, Options, Help), a toolbar, and a status bar.

Enabled Transitions

go[front()]: Gate → Train(5)

Next Reset

Simulation Trace

Train(1)

(Safe, Cross, Stop, Stop, Stop, Stop, Occ)

leave[1]: Train(1) → Gate[1]

(Safe, Safe, Stop, Stop, Stop, Stop, Free)

go[front()]: Gate → Train(5)

(Safe, Safe, Stop, Stop, Stop, Start, Occ)

appr[0]: Train(0) → Gate[0]

Trace File:

Prev Next Replay

Open Save Random

Slow Fast

Gate

list = {5,3,4,2,0,0,0}

Train(0)

Train(1)

Train(2)

Train(3)

Train(4)

Train(5)

Gate

Occ

Occ

Stop

Free

leave[1]

go[front()]

Train(4).x ∈ [23,60]

Train(5).x ∈ [30,65]

Train(0).x - time ≤ -50

Train(0).x - Train(1).x ∈ [10,20]

Train(0).x - Train(2).x ∈ [0,5]

Train(3).x - Train(0).x ∈ [17,40]

Train(4).x - Train(0).x ∈ [10,35]

Train(2).x - Train(1).x ∈ [7,20]

Train(2).x - Train(1).x ∈ [7,20]

Train(3).x - Train(5).x ∈ [-5,0]

Train(4).x - time ≤ -33

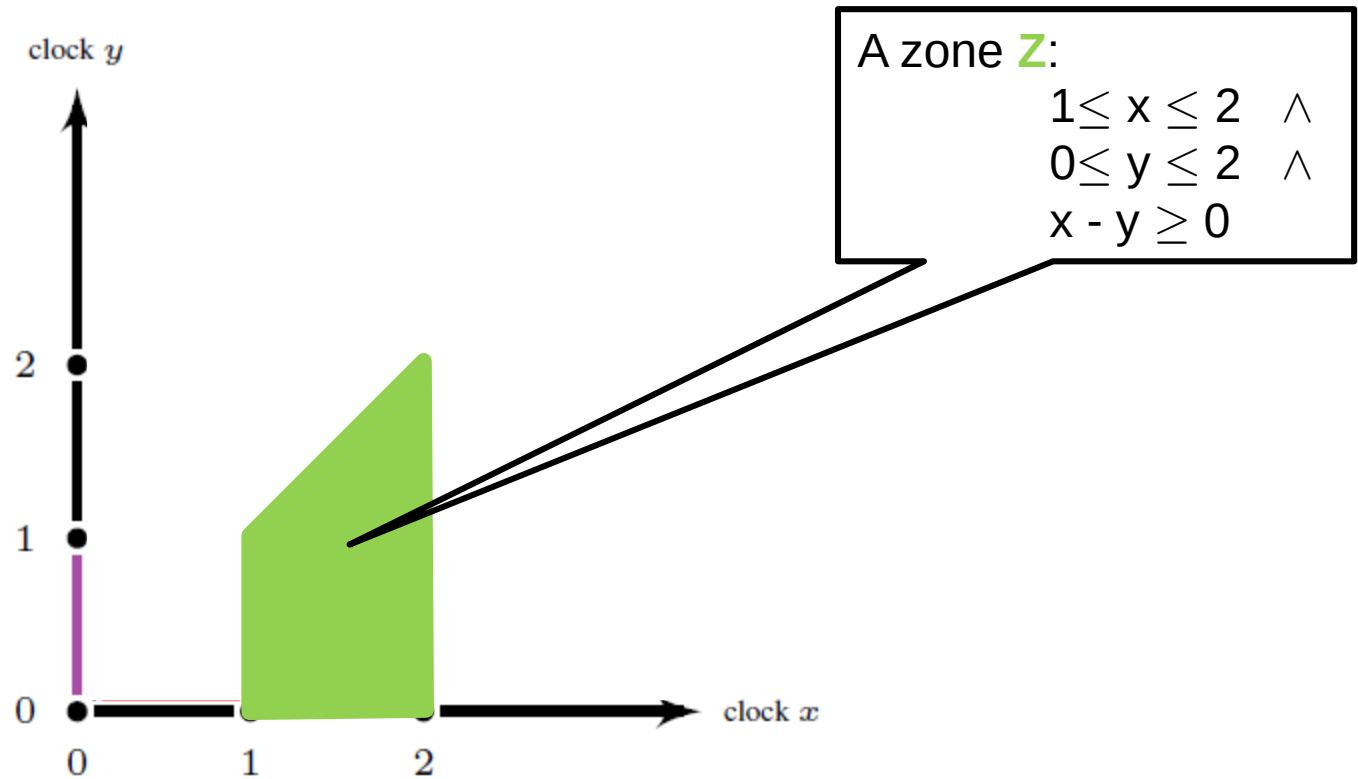
Train(4).x - Train(3).x ∈ [-20,0]

Train(5).x - time ≤ -30

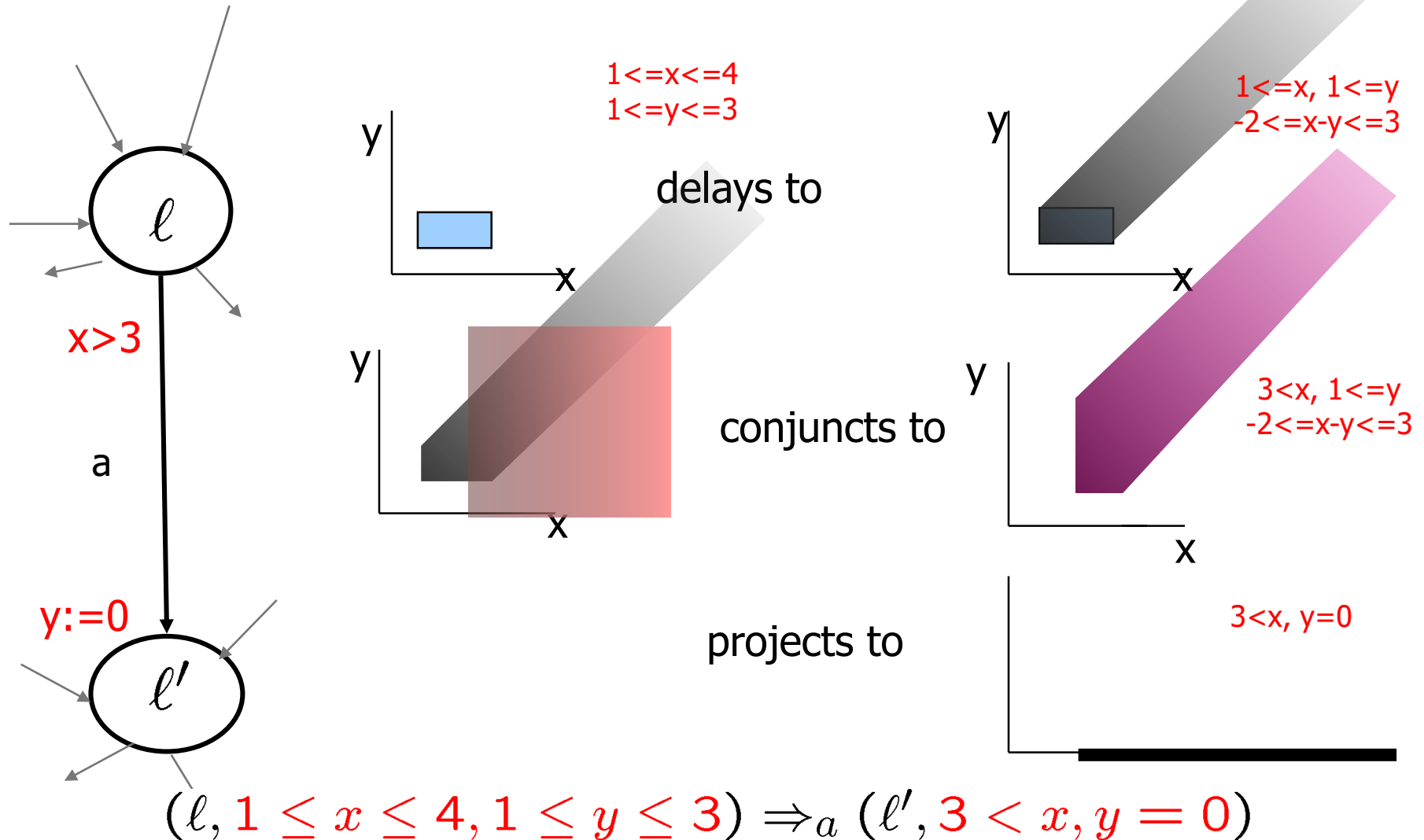
Train(5).x - Train(0).x ∈ [17,40]

Train(5).x - Train(4).x ∈ [0,20]

Zones – From Finite to Efficiency

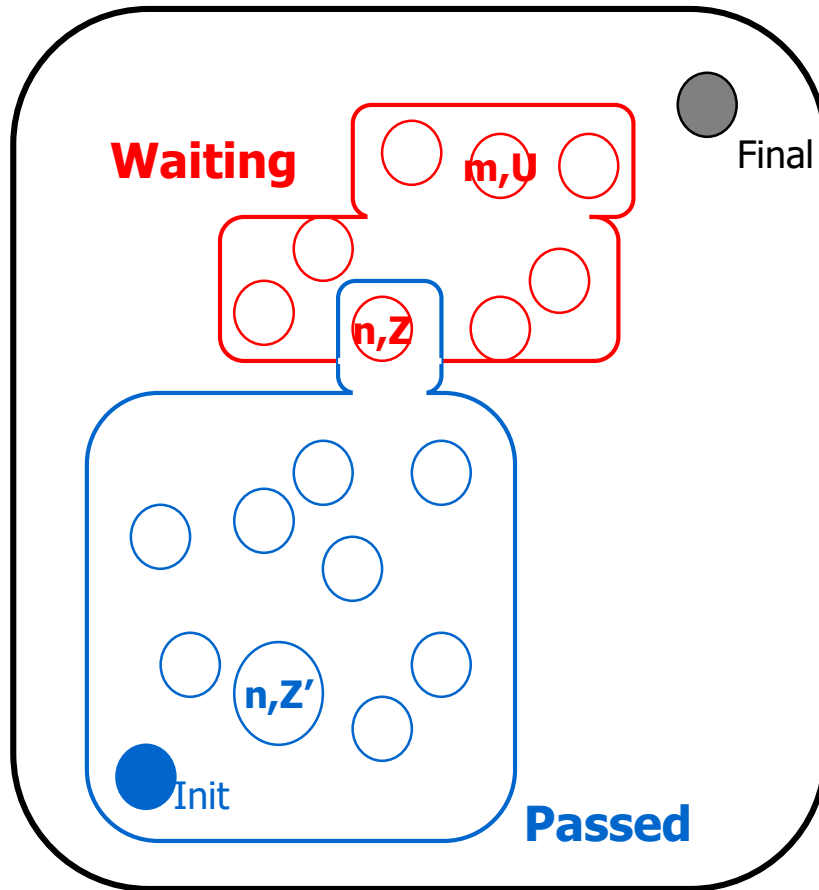


Symbolic Transitions / Zones



Forward Reachability

Init $\rightarrow$ Final ?



INITIAL **Passed** $:= \emptyset$;
Waiting $:= \{(n_0, Z_0)\}$

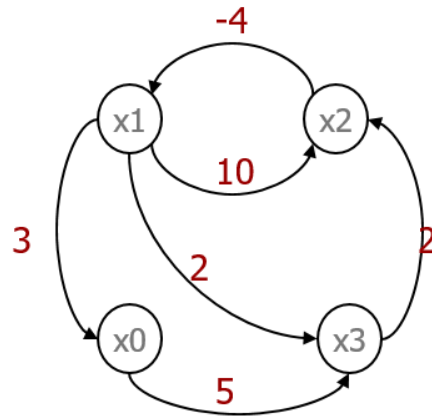
REPEAT

- pick (n, Z) in **Waiting**
- **if** for some $Z' \supseteq Z$
 (n, Z') in **Passed** **then STOP**
- **else** /explore/ add
 $\{ (m, U) : (n, Z) \Rightarrow (m, U) \}$
 to **Waiting**;
 Add (n, Z) to **Passed**

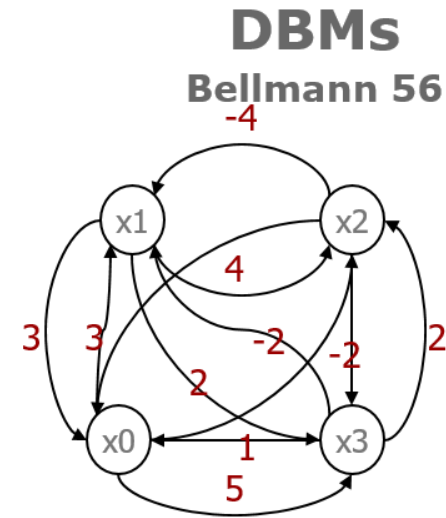
UNTIL **Waiting** $= \emptyset$
or
Final is in **Waiting**

Canonical Zone Representation

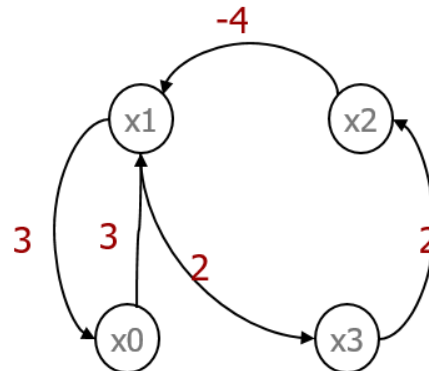
$x_1 - x_2 \leq 4$
 $x_2 - x_1 \leq 10$
 $x_3 - x_1 \leq 2$
 $x_2 - x_3 \leq 2$
 $x_0 - x_1 \leq 3$
 $x_3 - x_0 \leq 5$



**Shortest
Path
Closure
 $O(n^3)$**



**Shortest
Path
Reduction
 $O(n^3)$**



**Minimal Constraint Form
RTSS'97**

Space worst $O(n^2)$
practice $O(n)$

Datastructures for Zones

- DBM package

- Mi

- Fo

- Cl

- Di

The screenshot shows a Mozilla browser window with the address bar set to <http://www.uppaal.com/>. The website header features the UPPAAL logo and the text "DBM Library" in red. Below the header, there is a section for the current version, "1.0.2", with a "Download for Linux" button. To the left, under "Documentation", there are links for the API, related presentations, and papers. To the right, under "NEWS", there are two entries: one from 04/02/2004 about version 1.0.2 and another from 22/12/2004 about version 1.0.1. Below the news, there is a section titled "UPPAAL DBM LIBRARY" which describes the library's purpose and features. At the bottom right, there is a small diagram showing a transition from a state labeled 'X' to a state labeled 'True'.

UPPAAL - Mozilla

File Edit View Go Bookmarks Tools Window Help

http://www.uppaal.com/

Home Bookmarks

UPPAAL DBM Library

Current version is **1.0.2**

Download for Linux

Documentation

- API of the library
- Related presentation on the library and subtractions:
- Presentation (ppt)
- Presentation (pdf)
- Related papers:

NEWS

- 04/02/2004: Version 1.0.2 released. Changes are: new functions for federations and bug fixes for Federation, dbmf_predt, and the subtractions (rare bug).
- 22/12/2004: UPPAAL DBM library 1.0.1 released.

UPPAAL DBM LIBRARY

DBMs (difference bound matrices) [rokicki93, lpw:fct95, bengtsson02] are efficient data structures to represent clock constraints in timed automata [ad90]. They are used in UPPAAL [py97, b04, bdl04] as the core data structure to represent time. The library features all the common operations such as up (delay, or future), down (past), general

X True