An Introduction to UPPAAL

Purandar Bhaduri
Dept. of CSE
IIT Guwahati
Email: pbhaduri@iitg.ernet.in

GLOBAL INITIATIVE OF ACADEMIC NETWORKS

OUTLINE

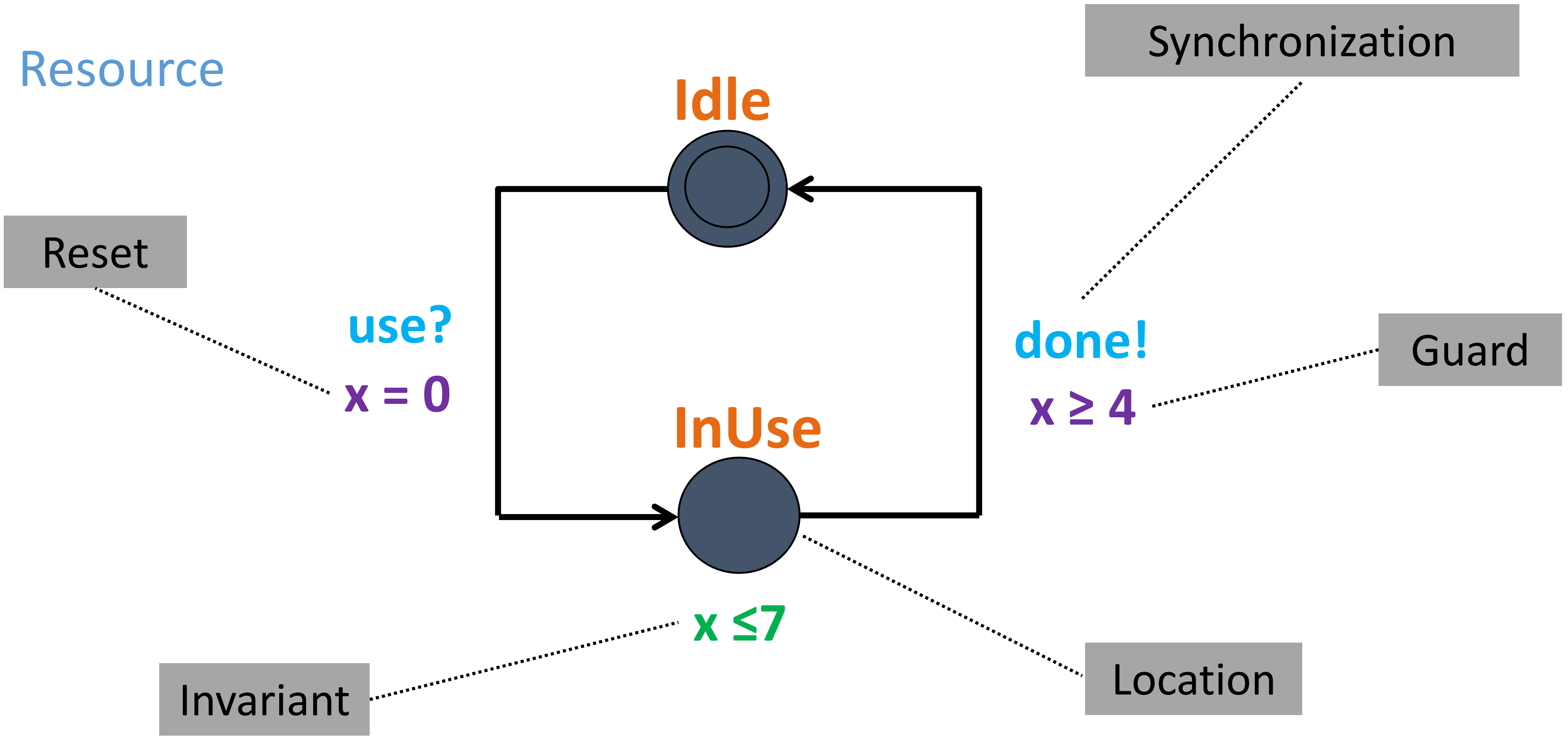
- Introduction
- Timed Automata
- UPPAAL
- Example: Train Gate
- Example: Task Scheduling

Introduction

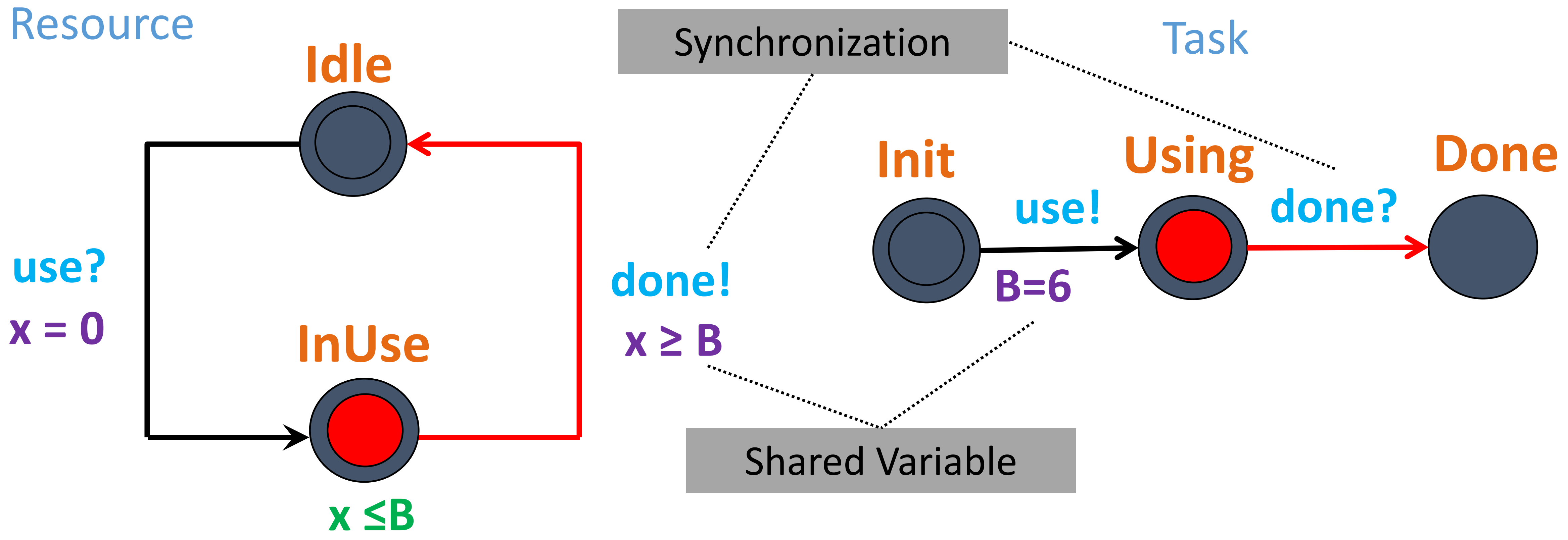
- UPPAAL: a toolbox for modelling, simulating and verifying real-time systems
  - Appropriate for systems that cab be modelled as a network of timed automata
    - Nondeterministic finite automata
    - Real-valued clocks
    - Communication through channels and shared variables
- Applications: where time is a critical resource
  - Real-time controllers
  - Communication protocols

Timed Automata

Resource



Timed Automata Composition



[Alur & Dill 90]

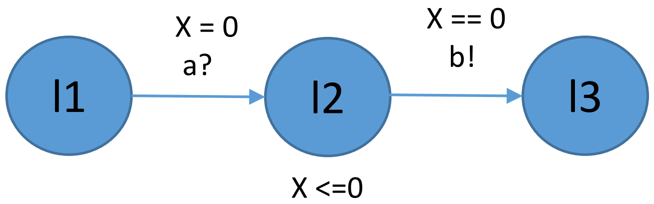
Locations in UPPAAL

- Normal Location
  - Time can pass as long as the invariant is satisfied
  - When the invariant becomes false the location must be exited
- Urgent Location
  - No delay
- Committed Location
  - No delay
  - In a composition the transition out of the committed location must be exited first if more than one transition is enabled

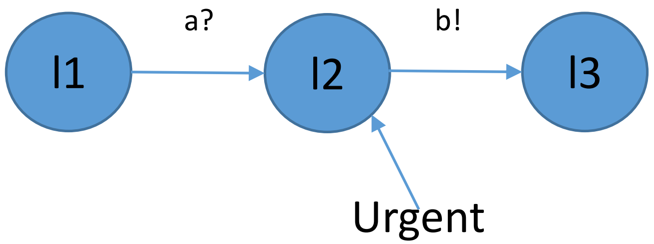
Urgent Location

- No delay

P:



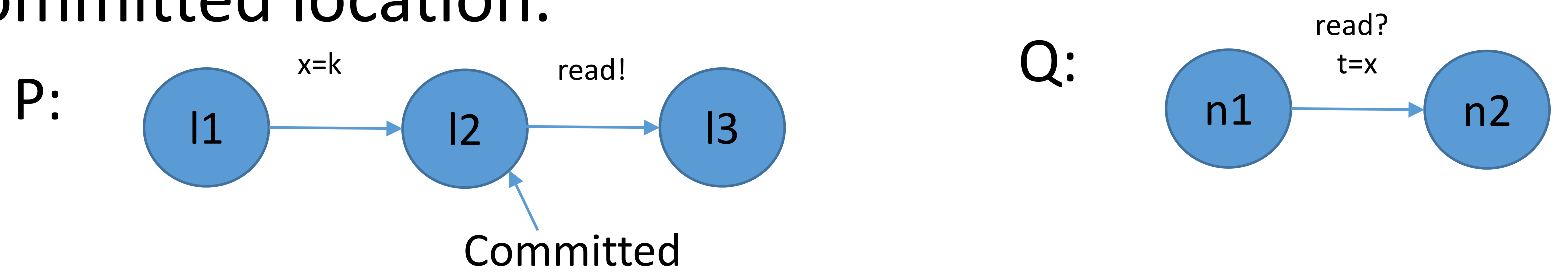
Q:



- P and Q have the same behaviour.

Committed Location

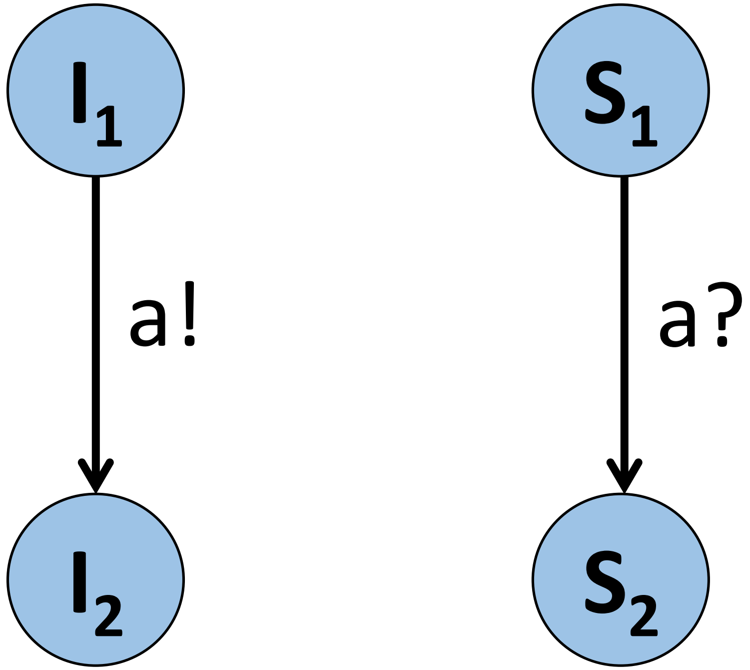
- No delay
- Next transition must involve an edge in one of the processes in a committed location.



- Location l2 is committed to ensure that no automaton can modify the x before automaton Q reads x .
- Enables accurate modelling of **atomic** behaviours.

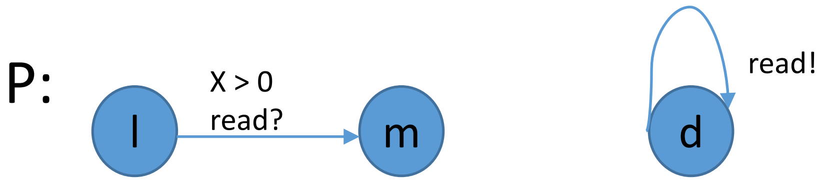
Synchronization Semantics in UPPAAL

- Used to coordinate the action of two or more processes.
- Transitions with the same synchronization channel are activated simultaneously
  - Guards must be true



Urgent Channel

- urgent chan a;
- Specifies synchronization that must be taken when the transition is enabled, without delay
  - No clock guard are allowed on the edges
  - Guards on data-variables
- Encode urgent transition on a variable (e.g., busy waiting on a variable)



Analysis: Model Checking

- Can check for invariant and reachability properties
  - Whether certain combinations of locations and constraints on variables (clock and integer) are reachable
- Bounded liveness
  - Monitor automata
  - Adding debugging information and checking reachability
- Generates diagnostic trace

Temporal Logic: TCTL

- E - exists a path (“**E**” in UPPAAL).
- A - for all paths (“**A**” in UPPAAL).
- G - all states in a path (“**[]**” in UPPAAL).
- F - some state in a path (“**<>**” in UPPAAL).

Queries in UPPAAL

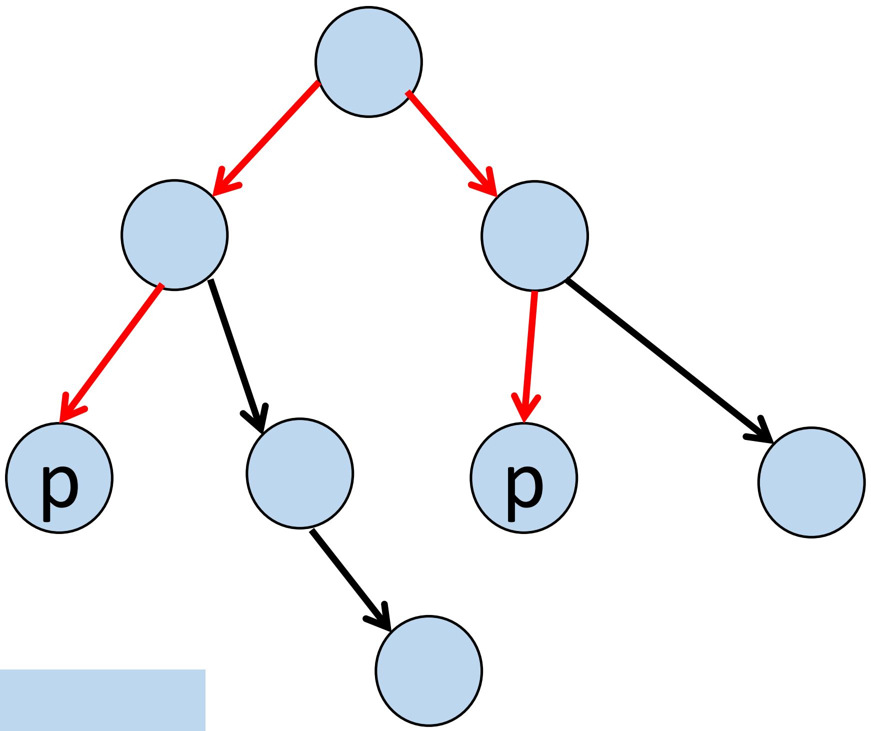
A[]p, A<>p, E<>p, E[]p and p → q

AG p   AF p   EF p   EG p

Propositions p and q are local properties
- atomic clock/data constraints: integer bounds on clock variables
- Component location

Validation Property

Possibly: E<>p



TCTL Quantifiers in UPPAAL

- E - exists a path (“**E**” in UPPAAL).
- A - for all paths (“**A**” in UPPAAL).
- G - all states in a path (“**[]**” in UPPAAL).
- F - some state in a path (“**<>**” in UPPAAL).

Queries in UPPAAL

A[]p, A<>p, E<>p, E[]p and p → q

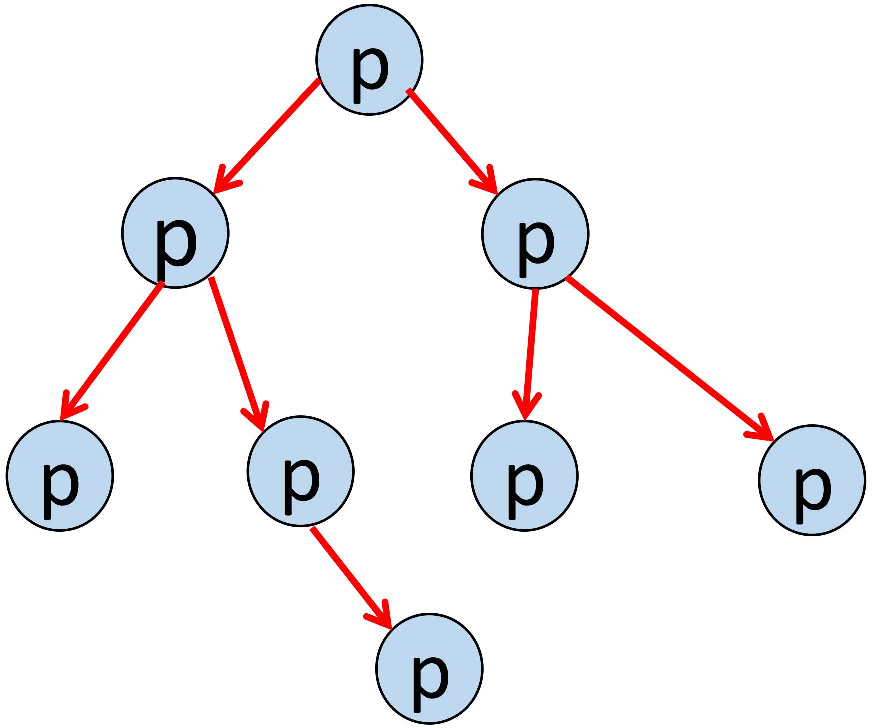
AG p   AF p   EF p   EG p

p and q are local properties

Safety Properties

Invariant: A[]p

Possibly Invariant: E[]p



TCTL Quantifiers in UPPAAL

- E - exists a path (“**E**” in UPPAAL).
- A - for all paths (“**A**” in UPPAAL).
- G - all states in a path (“**[]**” in UPPAAL).
- F - some state in a path (“**<>**” in UPPAAL).

Queries in UPPAAL

A[]p, A<>p, E<>p, E[]p and p → q

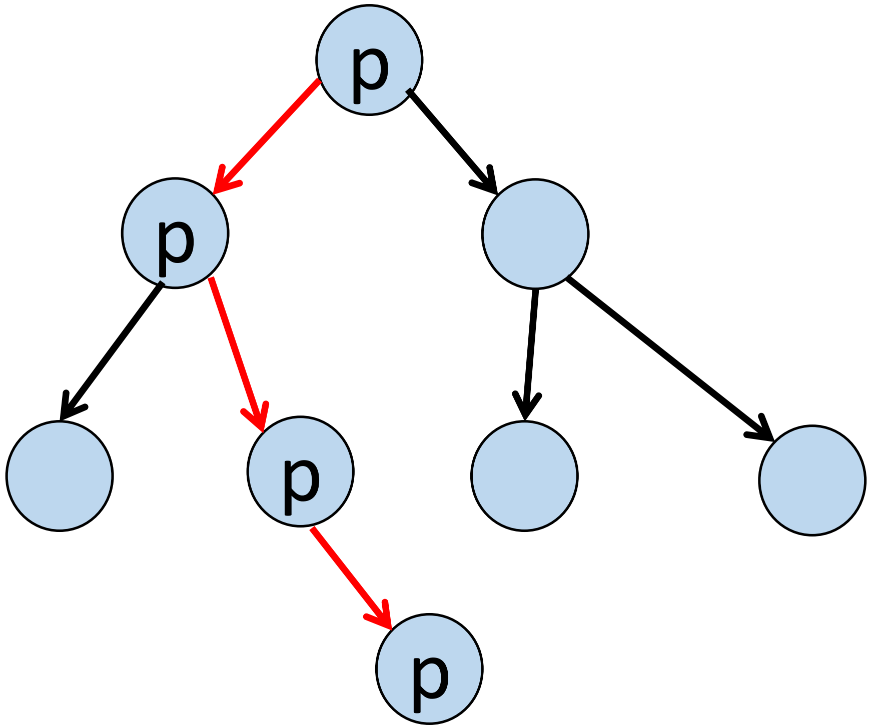
AG p   AF p   EF p   EG p

p and q are local properties

Safety Properties

Invariant: A[]p

Possibly Invariant: E[]p



TCTL Quantifiers in UPPAAL

- E - exists a path (“**E**” in UPPAAL).
- A - for all paths (“**A**” in UPPAAL).
- G - all states in a path (“**[]**” in UPPAAL).
- F - some state in a path (“**<>**” in UPPAAL).

Queries in UPPAAL

A[]p, A<>p, E<>p, E[]p and p → q

AG p   AF p   EF p   EG p

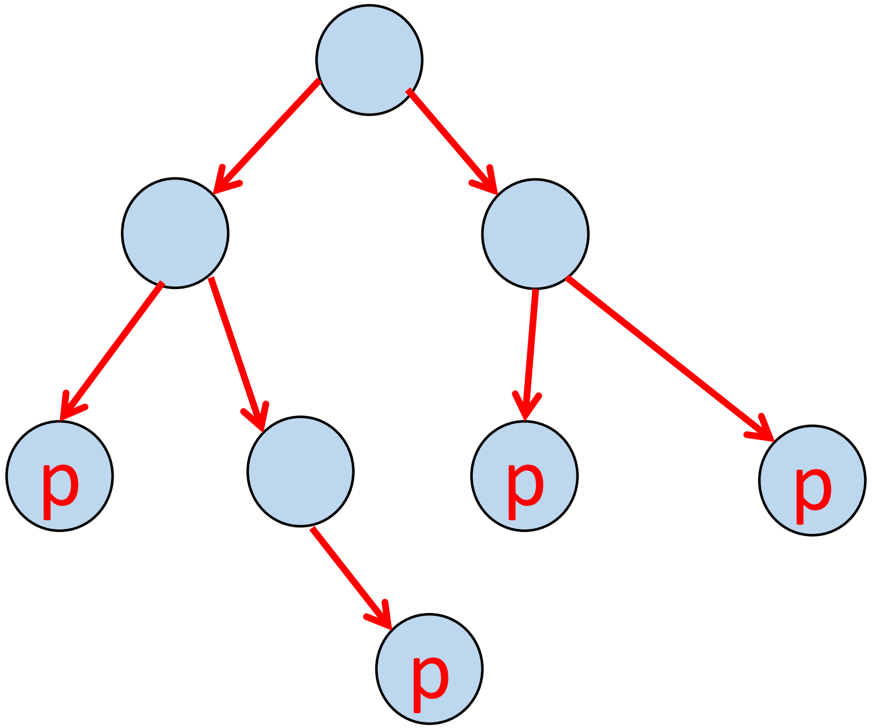
p and q are local properties

Liveness Properties

Always Eventually: A<>p

Always Leads to (p- ->q):

A<>[p -> A<>q]



TCTL Quantifiers in UPPAAL

- E - exists a path (“**E**” in UPPAAL).
- A - for all paths (“**A**” in UPPAAL).
- G - all states in a path (“**[]**” in UPPAAL).
- F - some state in a path (“**<>**” in UPPAAL).

Queries in UPPAAL

A[]p, A<>p, E<>p, E[]p and p → q

↓ AG p ↓ AF p ↓ EF p ↓ EG p

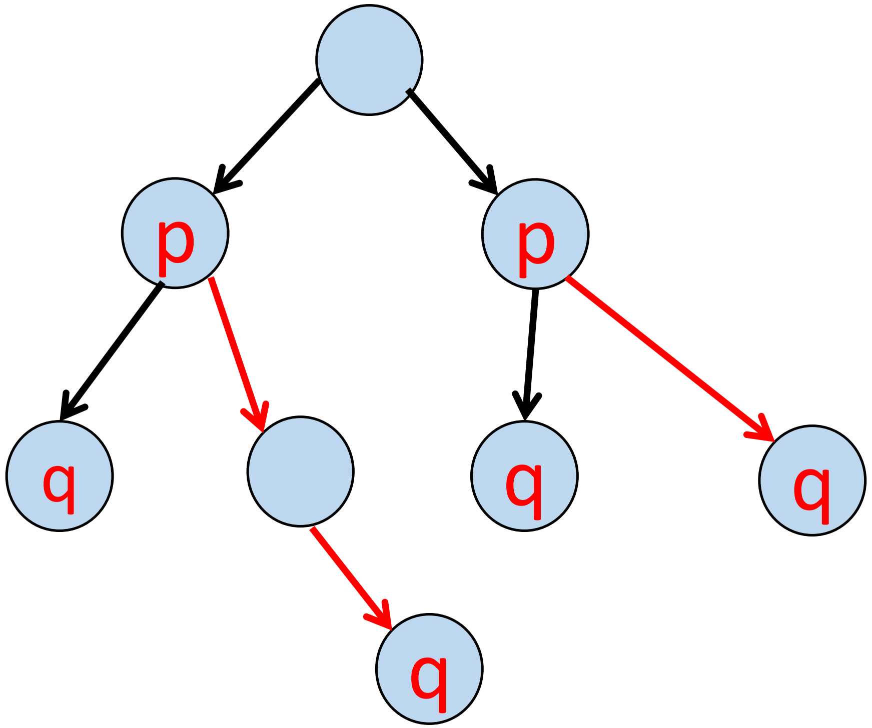
p and q are local properties

Liveness Properties

Eventually: A<>p

Always Leads to (p- ->q):

A[][p -> A<>q]



TCTL Examples

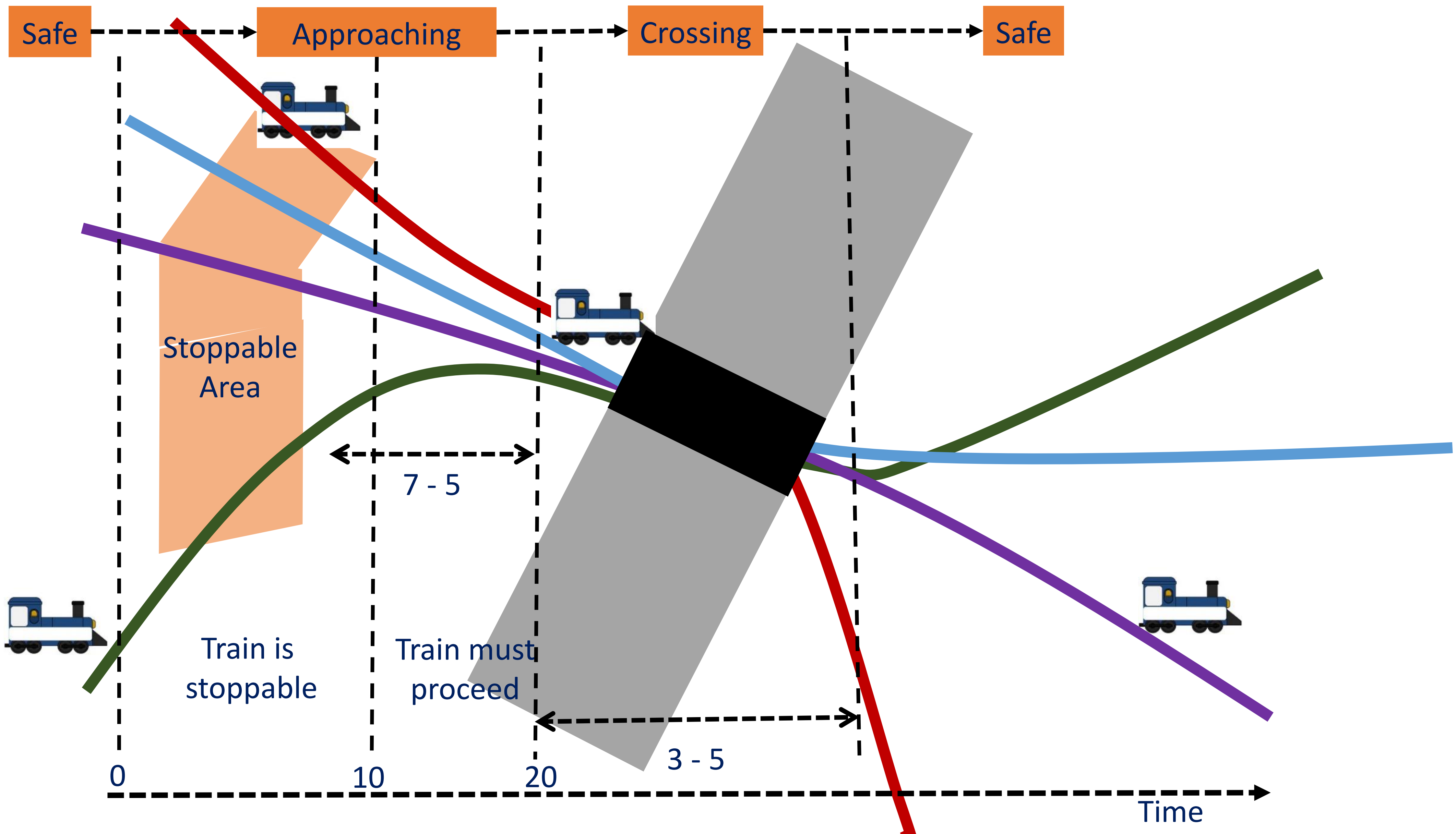
- A deadlock never occurs
  - A[] not deadlock
- An automaton A2 may never enter a location q
  - E[] not A2.q
- There exists a reachable state from which ϕ always holds
  - E<>A[] ϕ
- Infinitely often ϕ
  - A[]A<> ϕ
- Always ϕ is possible
  - A[]E<> ϕ

Example: Train Gate

- Two components: train, gate controller
- Trains running on separate tracks cross a common bridge
- Initially, trains are far enough from the bridge (location *safe*)
- When trains approach the bridge the gate controller is notified 20 time units before the train reaches the bridge(location *approaching*)
  - A train can be stopped within 10 time units; otherwise it must cross the bridge.
- Gate controller can stop a train and restart it
  - If train is stopped (location *stop*) then it will be eventually restarted (location *start*) again and it takes 7-15 time unit to reach the bridge.
- A train takes 3-5 time units to cross the bridge (location *cross*)
- After crossing, a train will go to its safe state again and notify the gate controller.
- Safety Property: Only one train at a time has access to the bridge.

[Kim Larsen: ARTIST Summer School 2009 slides + Uppaal distribution]

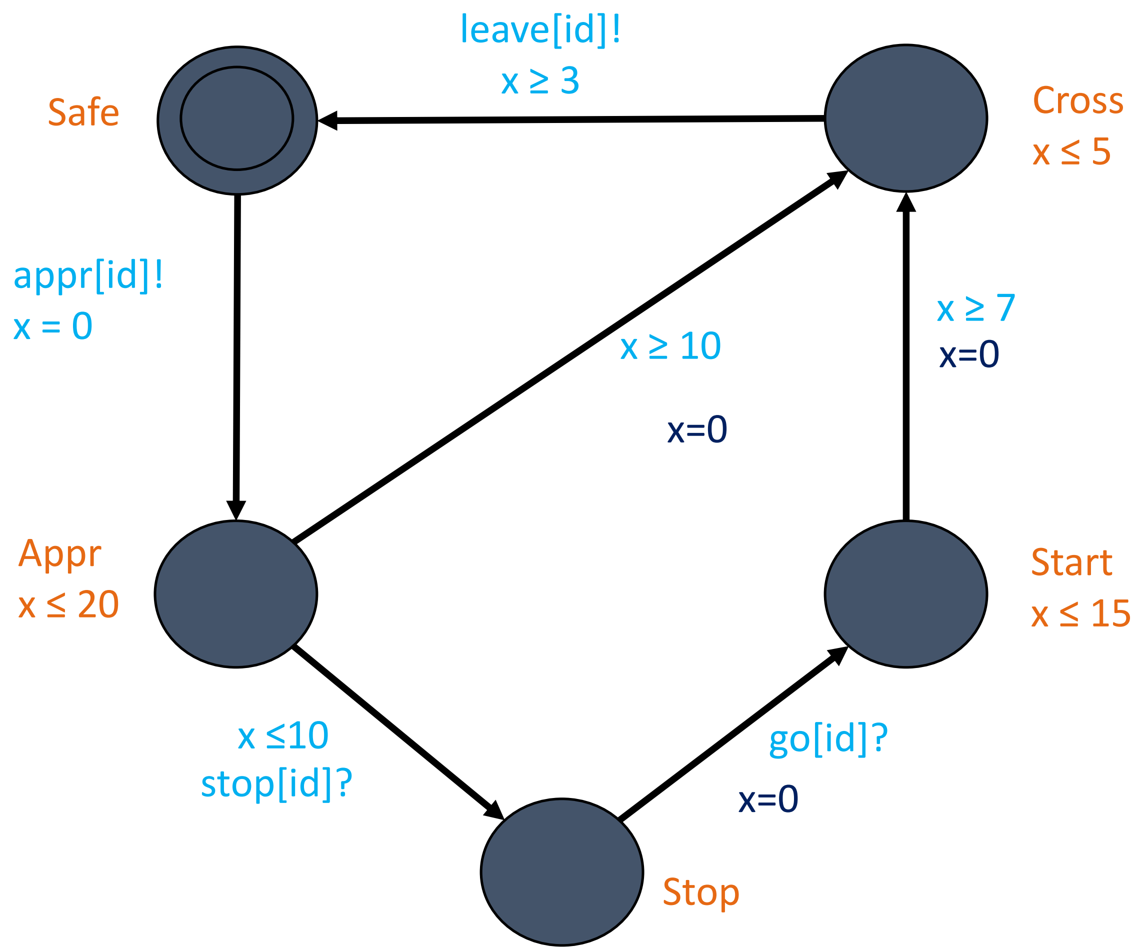
Example: Train Gate



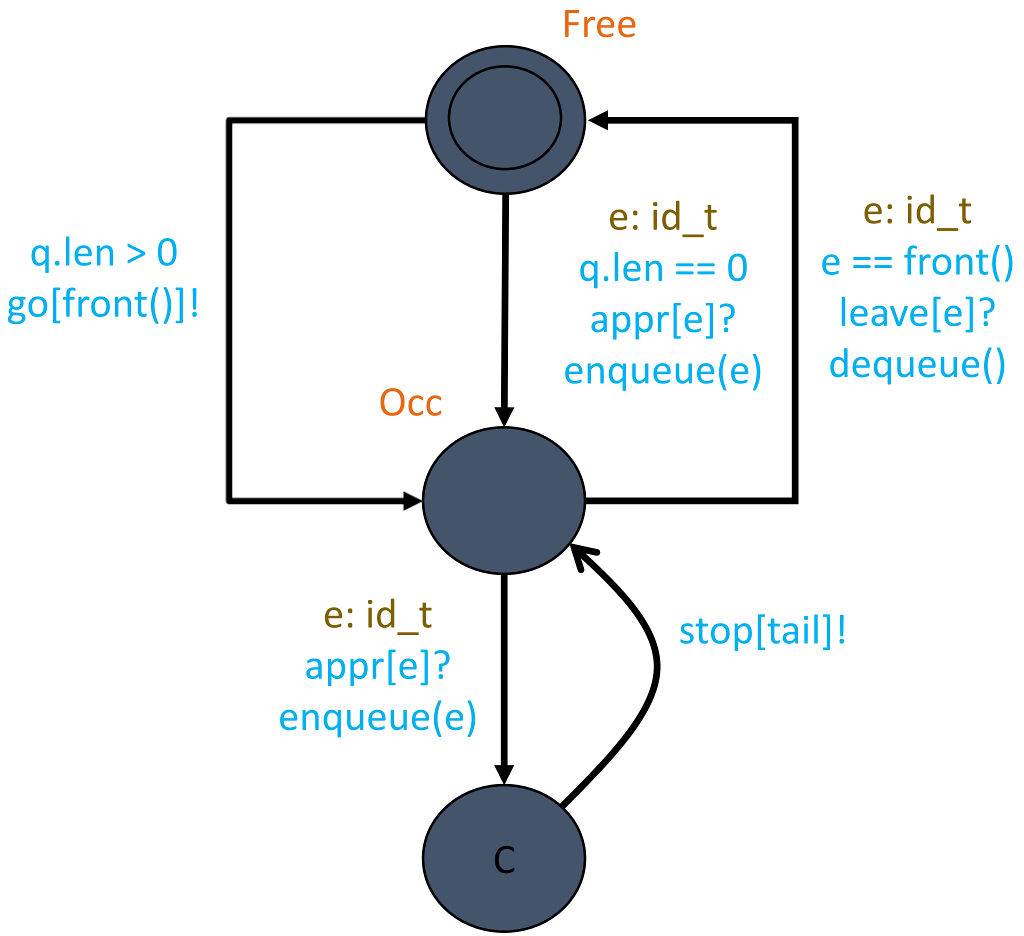
Example: Train Gate

- Global declaration
  - const int N = 6; // no of trains
  - typedef int [0,N-1] id_t; //used as argument for the template of trains
  - Chan appr[N], stop[N], leave[N];
  - urgent chan go[N];
- Local declaration (Train)
  - clock x;
- Local declaration (Gate)
  - typedef struct { id_t list[N]; int [0,N] len; } queue_t;
  - queue_t q;

Example: Train Gate



Template for the trains



Template for the gate

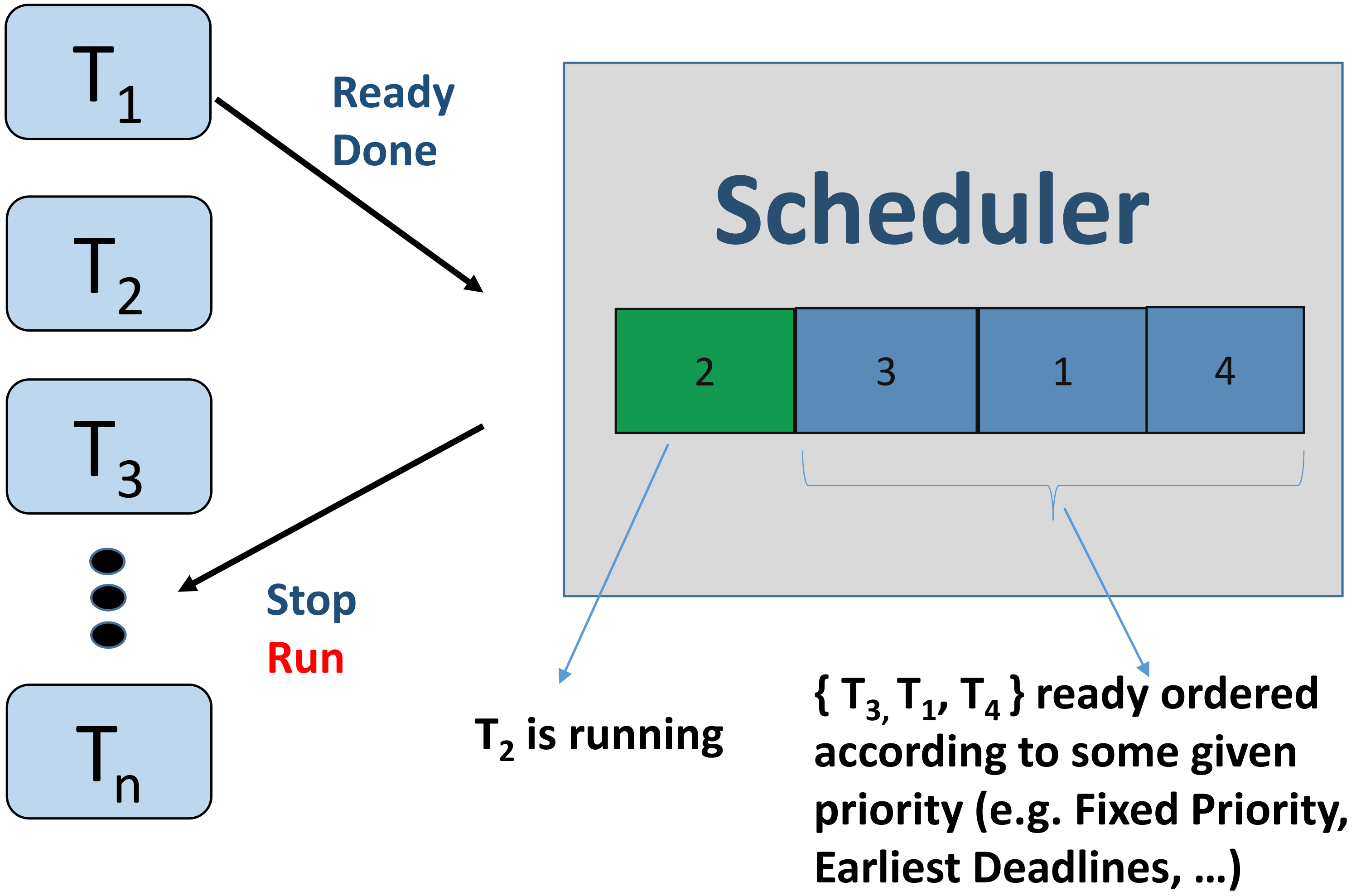
The train template has the argument **const id_t id** that defines its identifier

e: id_t to unfold the corresponding edge with **e** ranging over the type **id_t**

Example: Train Gate

- Verification
  - E<>Train1.Cross
  - E<> Train1.Cross and Train2.Stop
- Safety Properties
  - A[] Train1.Cross+Train2.Cross +Train3.Cross +Train4.Cross<=1
- Liveness Properties
  - Train1.Appr --> Train1.Cross
- System is deadlock free
  - A[] not deadlock

Example: Task Scheduling



E[i] Earliest arrival for T_i

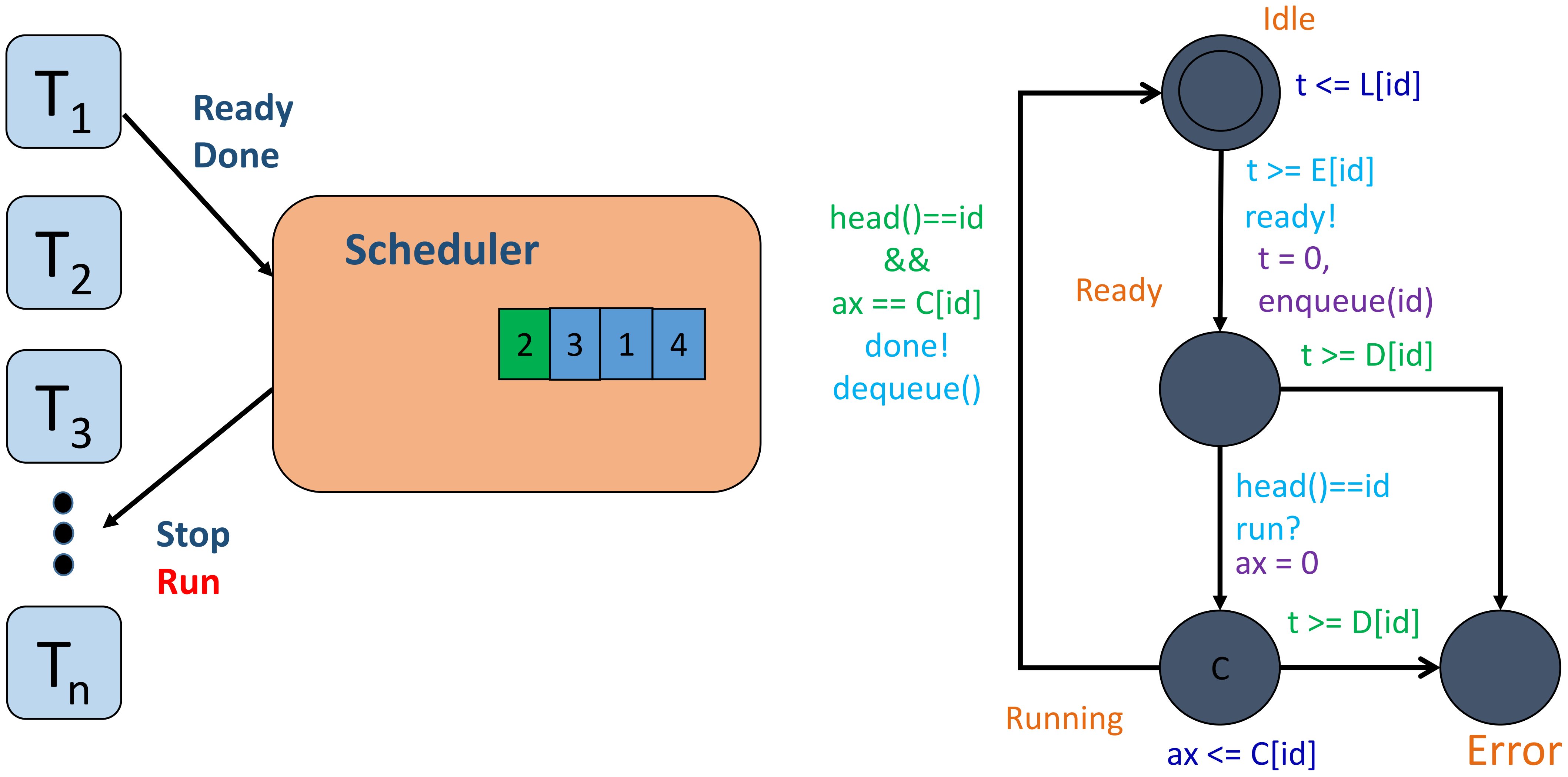
L[i] Latest arrival for T_i

C[i] Execution time for T_i

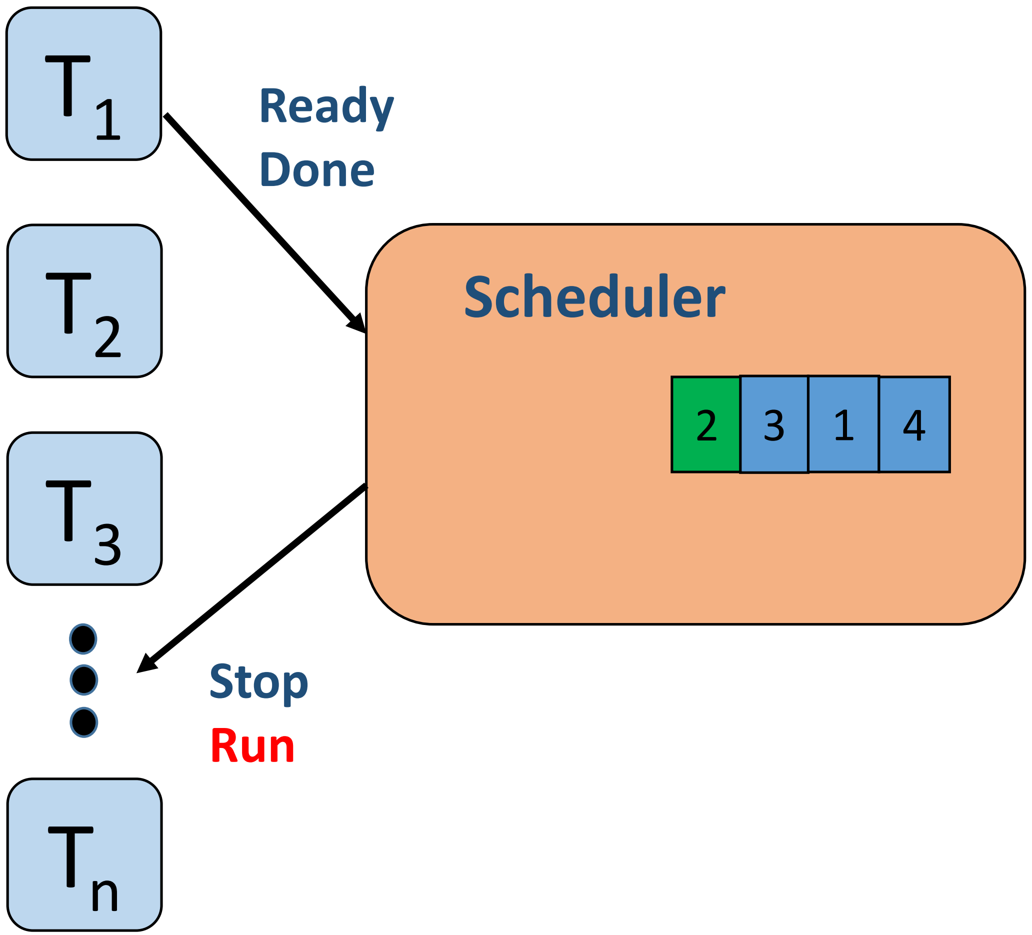
D[i] Deadline for T_i

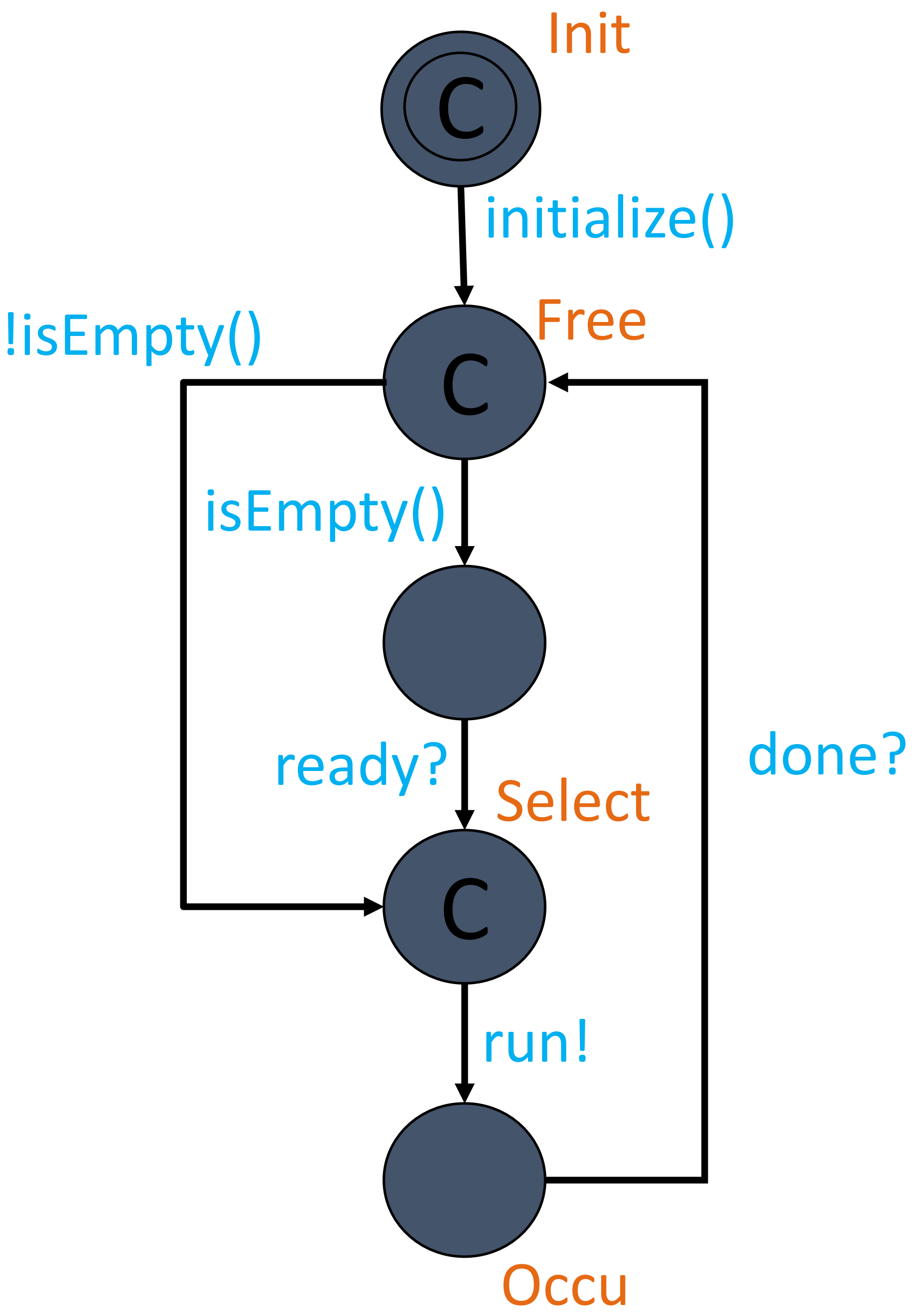
Kim G. Larsen, "Timing and Performance Analysis of Embedded Systems Using Model Checking", JTRES 2011

Modeling Task



Modeling Scheduler





References

1. UPPAAL. http://www.uppaal.com.
2. "Timed Automata: Semantics, Algorithms and Tools", Bengtsson, Johan, and Wang Yi. Advanced Course on Petri Nets. Springer, Berlin, Heidelberg, 2003.
3. "A Tutorial on Uppaal", Behrmann, Gerd, Alexandre David, and Kim Larsen. Formal methods for the design of real-time systems (2004): 33-35.
4. "Uppaal in a Nutshell", Larsen, Kim G., Paul Pettersson, and Wang Yi. International Journal on Software Tools for Technology Transfer (STTT) 1.1 (1997): 134-152.