Verification of Real-Time Systems

In general, two types of system requirements (to verify)

- Safety: nothing bad should happen
  - e.g.
    - Deadlock freeness,
    - no deadline missed
- Liveness: good thing should be repeated
  - e.g.
    - any message sent should be delivered eventually to the receiver
    - all service requests should be granted eventually or
    - any failure should be recovered within 10 ms (bounded liveness)

Formalizing Safety Properties in UPPAAL/TIMES

- Reachability properties: E<> Q
  - E<> P.stop
  - E<> (y>200)
- Invariant properties: A[] Q (not E<> not Q)
  - A[] not (P1.CS and P2.cs)
  - A[] (i < 100)
  - A[] (x>10 imply i>100)
    - After 10, i should be larger than 100
- Schedulability analysis (by TIMES)
  - No deadline missed
- Deadlock-freedom
  - Verified by default by UPPAAL/TIMES

Examples

Example: Petersson's algorithm

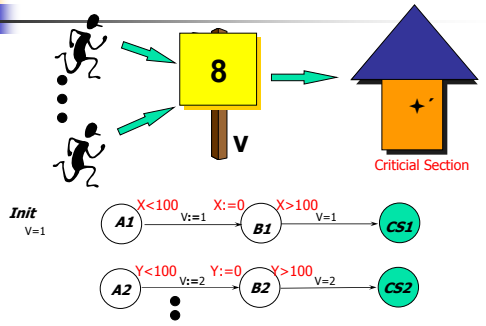
turn: shared variable

- Process 1
- Loop
- flag1:=1; turn:=2
- While (flag2 and turn=2) wait
- CS1
- flag1:=0
- End loop

- Process 2
- Loop
- flag2:=1; turn:=1
- While (flag1 and turn=1) wait
- CS2
- flag2:=0
- End loop

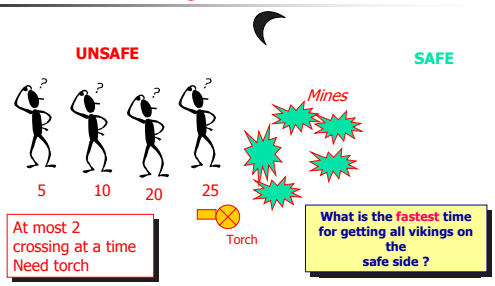
Question: no more than one process run in CS?

Example: Fischer's Protocol



Example: the Vikings Problem

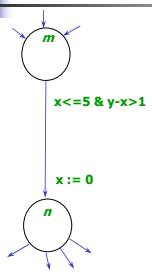
Real time scheduling



Problem: reachability analysis

- Give an automaton and a location n, or a local property F
- Question: does it exist an execution of the automaton, that leads to n (or a state where F holds)?
- This is the so called reachability problem.

Timed Automata: Semantics



State

(*location* , *clock-assignment*)

Transitions

(*m* , *x*=2.4 , *y*=3.1415) $\xrightarrow{1.1}$ (*m*, *x*=3.5 , *y*=4.2415)

(*m* , *x*=1.14 , *y*=3.1415) $\longrightarrow$ (*n* , *x*=0 , *y*=3.1415)

Reachability Problems

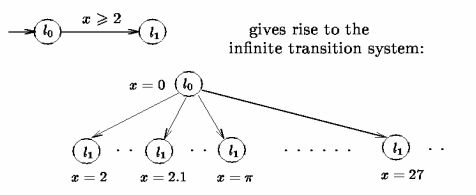
n is reachable from m if there is a sequence of transitions:

(*m*, *x*=r, *y*=s) $\longrightarrow^{*}$ (*n* , *x*=r' , *y*=s')

Formalizing requirements

- Reachability properties: E<> Q
  - E<> P.stop
  - E<> (y>200)
- Invariant properties: A[] Q (not E<> not Q)
  - A[] not (P1.CS and P2.cs)
  - A[] (i < 100)
  - A[] (x>10 imply i>100)
    - After 10, i should be larger than 100
- Schedulability analysis (in TIMES)
  - No deadline missed

Infinite State Space!



However, the reachability problem is decidable ☺ Alur&Dill 1991

Algorithms and Data Structures for Verification of Timed Automata

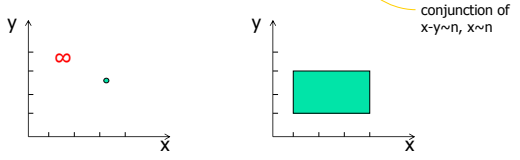
ZONES

Zones: From infinite to finite

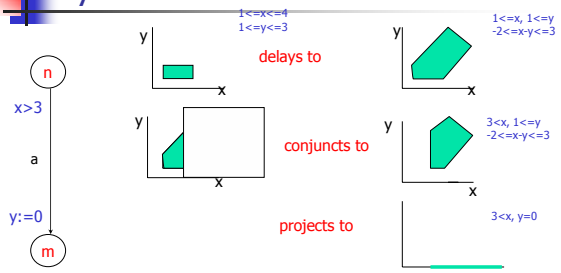
State
(n, x=3.2, y=2.5)

Symbolic state (zone)
(n, 1≤x≤4,1≤y≤3)

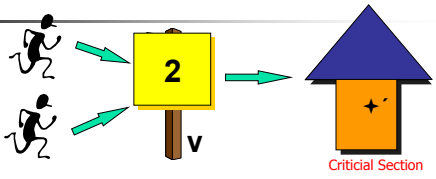
Zone: conjunction of x-y~n, x~n

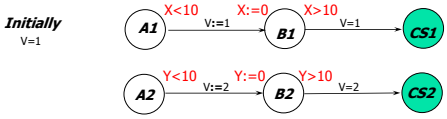


Symbolic Transitions

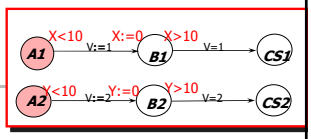


Fischer's Protocol
analysis using zones

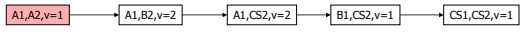




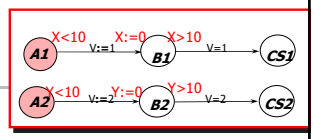
Fischers cont.



Untimed case



Fischers cont.

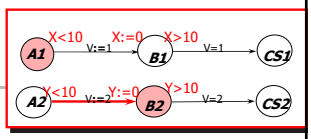


Untimed case

A1,A2,v=1 → A1,B2,v=2 → A1,CS2,v=2 → B1,CS2,v=1 → CS1,CS2,v=1

Taking time into account

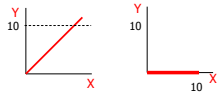
Fischers cont.



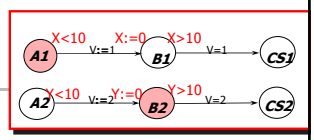
Untimed case

A1,A2,v=1 → A1,B2,v=2 → A1,CS2,v=2 → B1,CS2,v=1 → CS1,CS2,v=1

Taking time into account



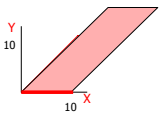
Fischers cont.



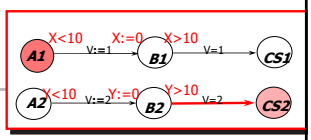
Untimed case

A1,A2,v=1 → A1,B2,v=2 → A1,CS2,v=2 → B1,CS2,v=1 → CS1,CS2,v=1

Taking time into account



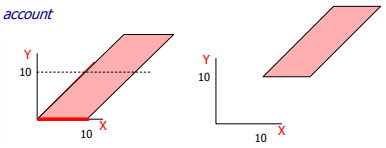
Fischers cont.



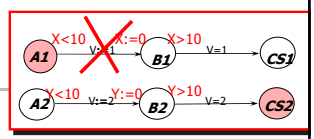
Untimed case

A1,A2,v=1 → A1,B2,v=2 → A1,CS2,v=2 → B1,CS2,v=1 → CS1,CS2,v=1

Taking time into account



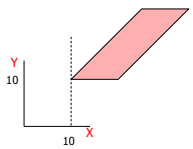
Fischers cont.



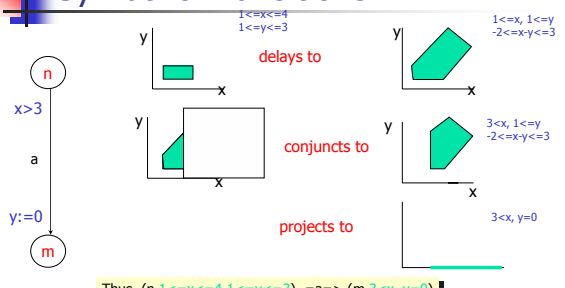
Untimed case

A1,A2,v=1 → A1,B2,v=2 → A1,CS2,v=2 → B1,CS2,v=1 → CS1,CS2,v=1

Taking time into account



Symbolic Transitions



Thus (n,1<=x<=4,1<=y<=3) =a=> (m,3<x, y=0)

Zones = Conjuctive constraints

- A zone Z is a conjunctive formula:
  $g_1 \& g_2 \& \ldots \& g_n$
  where g_i is a clock constraint:
  $x_i \sim b_i$ or $x_i - x_j \sim b_{ij}$
- Use a zero-clock x_0 (constant 0)
- A zone can be re-written as a set:
  $\{x_i - x_j \sim b_{ij} \mid \sim \text{is} < \text{or} \leq, i,j \leq n\}$
- This can be represented as a MATRIX, DBM (Difference Bound Matrices)

Solution set as semantics

- Let Z be a zone (a set of constraints)
- Let [Z]={u | u is a solution of Z}
  - The semantics

(We shall simply write Z instead [Z])

Operations on Zones

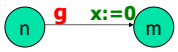
- Strongest post-condition (Delay): SP(Z) or $Z^{\uparrow}$
  - $[Z^{\uparrow}] = \{u+d \mid d \in R, u \in [Z]\}$
- Weakest pre-condition: WP(Z) or $Z^{\downarrow}$ (the dual of $Z^{\uparrow}$)
  - $[Z^{\downarrow}] = \{u \mid u+d \in [Z] \text{ for some } d \in R\}$
- Reset: {x}Z or Z(x:=0)
  - $[\{x\}Z] = \{u[0/x] \mid u \in [Z]\}$
- Conjunction
  - $[Z\&g] = [Z] \cap [g]$

An important theorem on Zones

- The set of zones is closed under all constraint operations (including x:=x-c or x:=x+c)
  - That is, the result of the operations on a zone is a zone
  - That is, there will be a zone (a finite object i.e a zone/constraints) to represent the sets: $[Z^{\uparrow}]$, $[Z^{\downarrow}]$, $[\{x\}Z]$

One-step reachability: Si→Sj

- Delay: $(n,Z) \rightarrow (n,Z')$ where $Z' = Z^{\uparrow} \wedge inv(n)$
- Action: $(n,Z) \rightarrow (m,Z')$ where $Z' = \{x\}(Z \wedge \mathbf{g})$

if 

- Successors(n,Z)={(m,Z') | $(n,Z) \rightarrow (m,Z')$, $Z' \neq \emptyset$}
  - Sometime we write: $(n,Z) \rightarrow (m,Z')$ if (m,Z') is a successor of (n,Z)

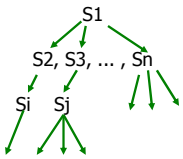
Two more operations on Zones

- Inclusion checking: $Z_1 \subseteq Z_2$
  - solution sets
- Emptiness checking: $Z = \emptyset$
  - no solution

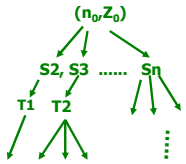
All Operations on Zones

(needed for reachability analysis)

- Transformation
  - Conjunction
  - Post condition (delay)
  - Reset
- Consistency Checking
  - Inclusion
  - Emptiness



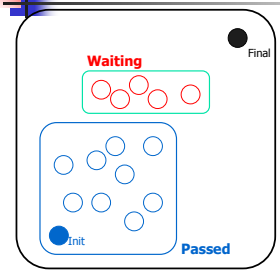
Now, we have a search problem



REACHABILITY ALGORITHM

Forward Rechability

Init -> Final ?



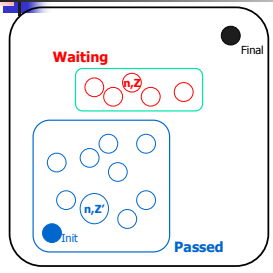
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

REPEAT
- pick (n,Z) in **Waiting**
- **if** for some Z' ⊇ **Z**
  (n,Z') in **Passed** **then** **STOP**
- **else** (explore) add
  successors(n,Z) to **Waiting**;
  Add (n,Z) to **Passed**

UNTIL **Waiting** = Ø
or
Final is in **Waiting**

Forward Rechability

Init -> Final ?



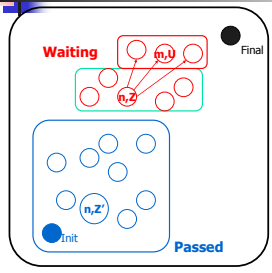
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

REPEAT
- pick (n,Z) in **Waiting**
- **if** for some Z' ⊇ Z
  (n,Z') in **Passed** **then** **STOP**

UNTIL **Waiting** = Ø
or
Final is in **Waiting**

Forward Rechability

Init -> Final ?



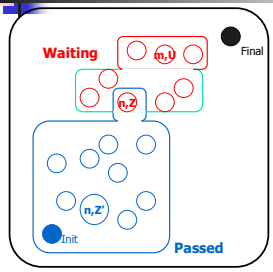
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

REPEAT
- pick (n,Z) in **Waiting**
- **if** for some Z' ⊇ Z
  (n,Z') in **Passed** **then** **STOP**
- **else** /explore/ add
  successors(n,Z) to **Waiting**;

UNTIL **Waiting** = Ø
or
Final is in **Waiting**

Forward Rechability

Init -> Final ?



INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

REPEAT
- pick (n,Z) in **Waiting**
- **if** for some Z' ⊇ Z
(n,Z') in **Passed** **then** **STOP**
- **else** /explore/ add
successors(n,Z) to **Waiting**;
Add (n,Z) to **Passed**

UNTIL **Waiting** = Ø
or
Final is in **Waiting**