Model Checking of Timed Systems

A UPPAAL Tutorial

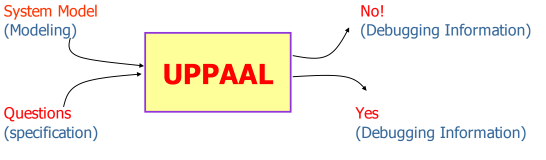
Wang Yi
Uppsala University, Sweden

SFM 2010, Bertinoro

This is simple, simple, simple … …

LESLIE LAMPORT

UPPAAL *A model checker for real-time systems*



Developed by UPPsala Univ + AALborg Univ = UPPAAL

Main Authors/Contributors of UPPAAL

- Johan Bengtsson
- Gerd Behrman
- Alexandre David
- Kim Larsen
- Fredrik Larsson
- Paul Pettersson and
- Wang Yi

OUTLINE

- Model Checking in a Nutshell
- Timed automata and TCTL
- A UPPAAL Tutorial
  - Data structures & central algorithms
  - UPPAAL input languages

(Recent Work: Multi-core Timing Analysis)

Main references

- **Temporal Logics (CTL)**
  - Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications: A Practical Approach. Edmund M. Clarke, E. Allen Emerson, A. Prasad Sistla, POPL 1983: 117-126, also as "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. ACM Trans. Program. Lang. Syst. 8(2): 244-263 (1986) "
- **Timed Systems (Timed Automata, TCTL)**
  - A Theory of Timed Automata. Rajeev Alur, David L. Dill. Theor. Comput. Sci. 126(2): 183-235 (1994)"
  - Symbolic Model Checking for Real-Time Systems, *Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Information and Computation* 111:193-244, 1994.
  - UPPAAL in a Nutshell. Kim Guldstrand Larsen, Paul Pettersson, Wang Yi. STTT 1(1-2): 134-152 (1997)
  - **Timed Automata – Semantics, Algorithms and Tools**, a tutorial on timed automata Johan Bengtsson and Wang Yi: (a book chapter in Rozenberg et al, 2004, LNCS).
  - **On-line help of UPPAAL**: www.uppaal.com

Model-Checking
in a Nutshell

7

Merits of model checking …

- Checking simple properties (e.g. deadlock-free) is already extremely useful!
  - It is not to prove that a system is completely correct (bug-free)
- The goal is to have tools that can help a developer find errors and improve the quality of her/his design.
  - It is to complement testing
- Now widely used in hardware design, protocol design, and hopefully soon, embedded systems!

8

History: Model-checking invented in 70's/80s
[Pnueli 77, Clarke et al 83, POPL83, Sifakis et al 82]

- Restrict attention to finite-state systems
  - Control skeleton + boolean (finite-domain) variables
  - Found in hardware design, communication protocols, process control
- Specification using CTL, LTL etc [Pnueli, Lamport, Clarke]
  - Safety, Progress/Liveness, Responsiveness etc
- BDD-based symbolic technique [Bryant 86]
  - SMV 1990 Clarke, McMillan et al, state-space 10^{20}
  - Now powerful tools used in hardware design
- On-the-fly enumerative technique [Holzman 89]
  - SPIN, COSPAN, CAESAR, KRONOS, IF/BIP, UPPAAL (since 1993) etc
- SAT-based techniques [Clarke et al …]

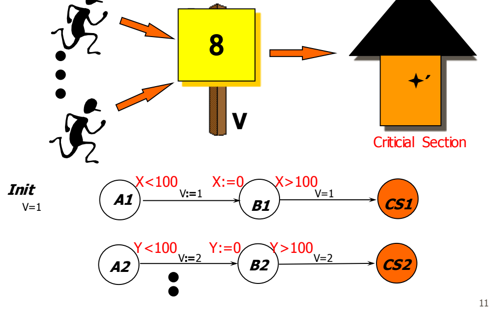
9

History: Model checking for real time systems, started in the 80s/90s

- Models of timed systems
  - Timed automata, [Alur&Dill 1990]
  - Timed process algebras, Timed CSP, Timed CCS [Wang 1990]
- Extension of model checking to consider time quantities
  - Timed variants of temporal logics e.g TCTL
- Tools
  - KRONOS, Hytech: 1993 --
  - UPPAAL 1995 –
    - TAB 1993/Prototype of UPPAAL [FORTE94, Wang et al]

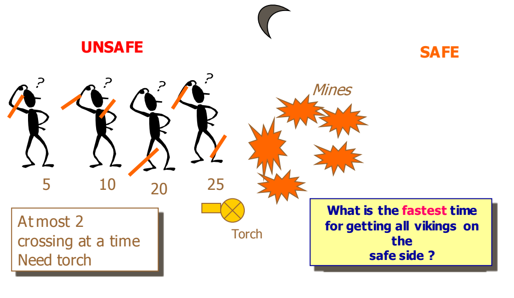
10

Example: Fischer's Protocol



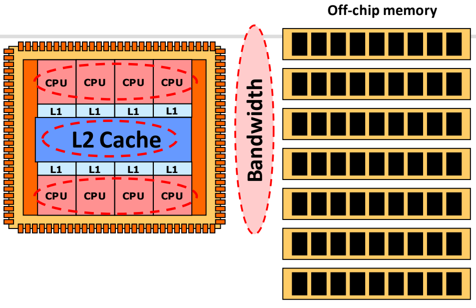
11

Example: the Vikings Problem
Real time scheduling



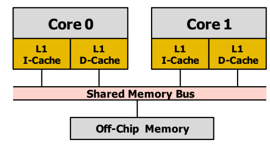
12

Multicore Challenges



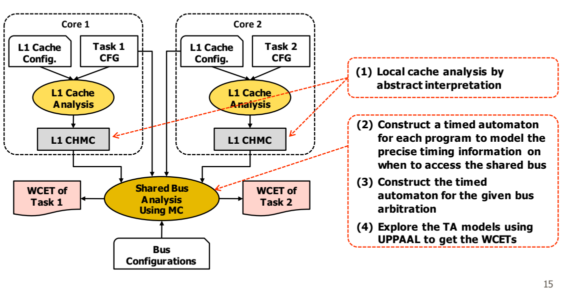
Shared Resources -- cpu's, caches, bandwidth, energy budget etc.

Worst-Case Execution Time Analysis of Concurrent Programs on Multicores

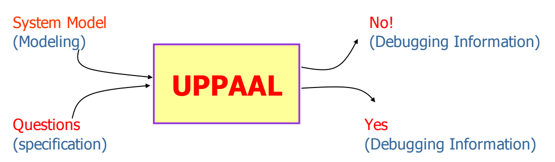


A duo-core processor with private L1 cache and shared memory bus

Combining Static Analysis & Model-Checking [RTSS 2010, submitted]



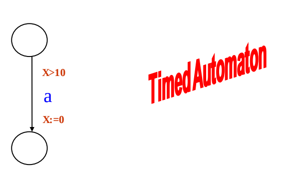
UPPAAL *A model checker for real-time systems*



MODELING

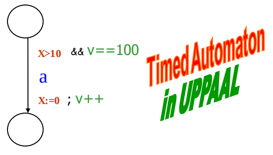
How to construct Model ?

Modeling Real Time Systems



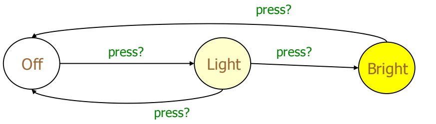
- Events
  - synchronization
  - interrupts
- Timing constraints
  - specifying event arrivals
  - e.g. Periodic and sporadic

Modeling Real Time Systems



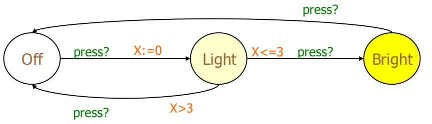
- Events
  - synchronization
  - interrupts
- Timing constraints
  - specifying event arrivals
  - e.g. Periodic and sporadic
- Data variables & C-subset
  - Guards
  - assignments

A Light Controller



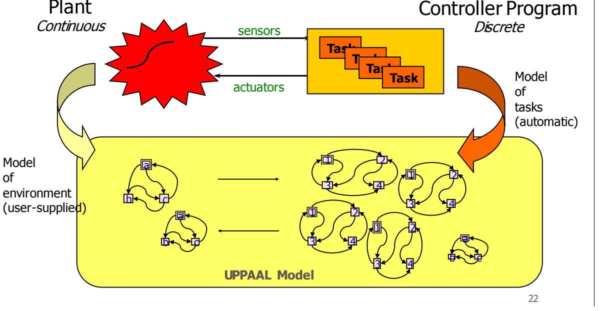
WANT: if press is issued twice quickly then the light will get brighter; otherwise the light is turned off.

A Light Controller (with timer)



Solution: Add real-valued clock x

Construction of Models: Concurrency



SPECIFICATION

How to ask questions: Specs ?

Specification=Requirement, Lamport 1977

- Safety
  - Something (bad) should not happen
- Liveness
  - Something (good) must happen/should be repeated

Computation Tree Logic, CTL

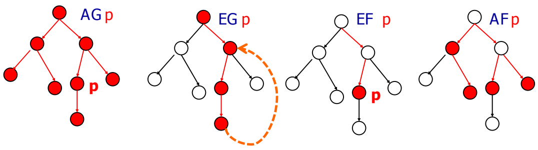
Clarke & Emerson 1980

Syntax

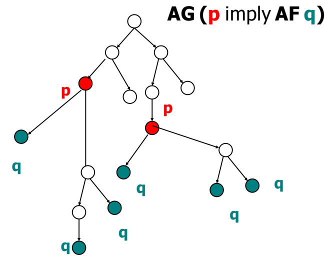
ϕ :: = P | $\neg\phi$ | $\phi \vee \phi$ | EX ϕ | E[ϕ U ϕ] | A[ϕ U ϕ]

where **P** $\in$ AP (atomic propositions)

Derived Operators



Liveness: p - -> q *"p leads to q"*



Specification: Examples

- Safety
  - AG $\neg$(P1.CS1 & P2.CS2) **Invariant**
  - AG (temp > 10 & speed < 120)
  - EF (time>60 imply viking4.safe) **Reachability**
  - EF (viking1.safe & viking2.safe & viking3.safe & viking4.safe)
- Liveness
  - AF (speed >100) **Eventually**
  - AG (P1.try imply AF P1.CS1) **Leads to**

VERIFICATION

Model meets Specs ?

Verification

- Semantics of a system
  = all states + state transitions
  (all possible executions)
- Verification
  = state space exploration + examination

Two basic verification algorithms

- Reachability analysis
  - Checking safety properties
- Loop detection
  - Checking liveness properties

UPPAAL DEMO

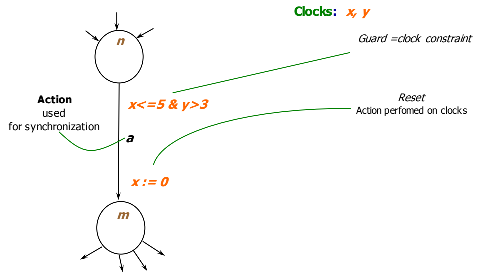
OUTLINE

- Model Checking in a Nutshell
- Timed automata and TCTL
- A UPPAAL Tutorial
  - Data stuctures & central algorithms
  - UPPAAL input languages

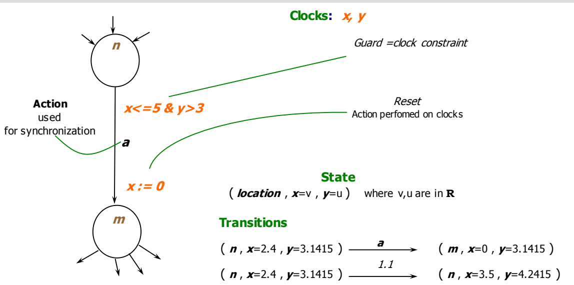
(Recent Work: Multicore Timing Analysis)

Timed Automata, TCTL & Verification Problems

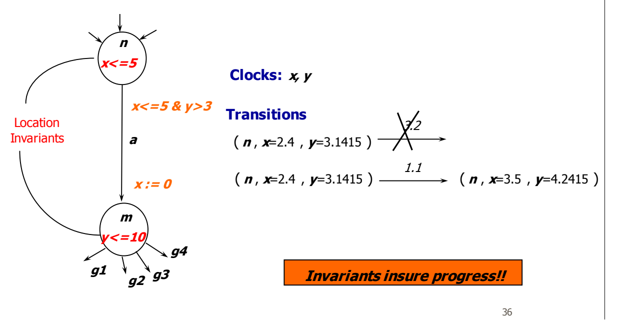
Timed Automata: Syntax



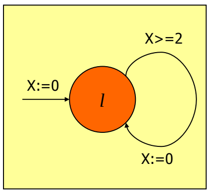
Timed Automata: Semantics



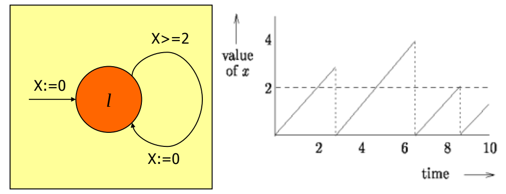
Timed Automata with *Invariants*



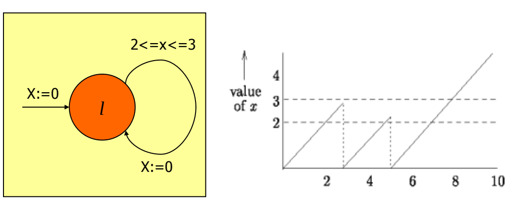
Timed Automata: Example



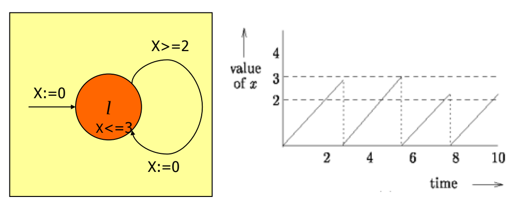
Timed Automata: Example



Timed Automata: Example



Timed Automata: Example



Timed Automata

=

Finite Automata + Clock Constraints + Clock resets

Clock Constraints

g ::= x ~ n | g & g

where

- x is a clock variable
- ~ ∈{<, >, ≤, ≥}
- n is a natural number and

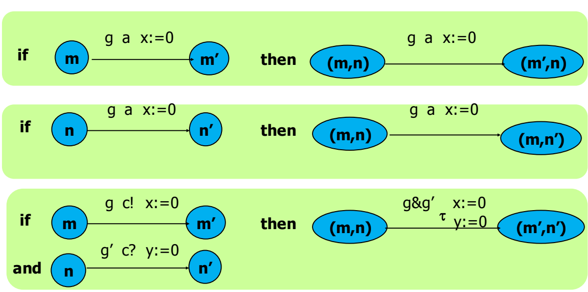
Semantics (definition)

- *clock valuations*: $V(C) \quad v: C \rightarrow R_{\geq 0}$
- *state*: $(l, v) \quad \text{where} \quad l \in L \;\; \text{and} \;\; v \in V(C)$
- *action transition*
  $(l,v) \xrightarrow{a} (l',v') \;\; iff \;\; l \xrightarrow{g\; a\; r} l'$
  $g(v) \;\; and \;\; v' = v[r] \;\; and \;\; Inv(l')(v')$
- *delay Transition*
  $(l,v) \xrightarrow{d} (l, v+d) \;\; iff$
  $Inv(l)(v+d') \;\; whenever \;\; d' \leq d \in R_{\geq 0}$

Modeling Concurrency

- Products of automata
- CCS Parallel composition
  - implemented in UPPAAL

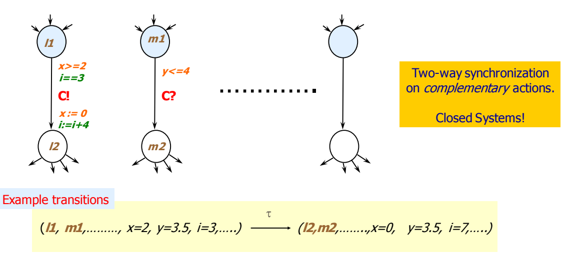
CCS Parallel Composition (implemented in UPPAAL)



where a is an action c! or c? or τ, and c is a channel name

The UPPAAL Model

= Networks of Timed Automata + Integer Variables +….



Verification Problems

Location Reachability (def.)

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

$$(m, u) \longrightarrow^{*} (n, v)$$

(Timed) Language Inclusion, $L(A) \subseteq L(B)$

$(a_0, t_0)\ (a_1, t_1)\ \dots\dots\ (a_n, t_n) \in L(A)$

If

"A can perform a_0 at t_0, a_1 at t_1 … … a_n at t_n"

$(l_0, u_0) \xrightarrow{t_0} (l_0, u_0+t_0) \xrightarrow{a_0} (l_1, u_1) \dots\dots$

49

Verification Problems

- Timed Language Equivalence & Inclusion ☹
  - 1-clock, finite traces, decidable [Ouaknine & Worrell 04]
  - 1-clock, infinite traces & Buchi-conditions, undecidable [Abdulla et al 05]
- Universality ☹
- Untimed Language Inclusion ☺
- (Un)Timed (Bi)simulation ☺
- Reachability Analysis/Emptiness ☺
- Optimal Reachability (synthesis problem) ☺
  - If a location is reachable, what is the minimal delay before reaching the location?

50

Timed CTL = CTL + clock constraints

Note that the semantics of TA defines a transition system where each state has a Computation Tree

51

Computation Tree Logic, CTL

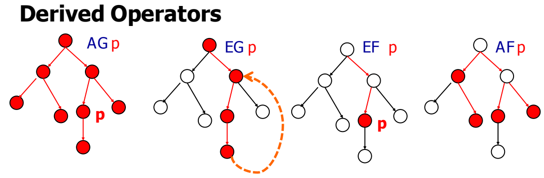
Clarke & Emerson 1980

Syntax

$\phi ::= P \mid \neg\phi \mid \phi \vee \phi \mid EX\,\phi \mid E[\phi\,U\,\phi] \mid A[\phi\,U\,\phi]$

where **P** ∈ AP (atomic propositions)

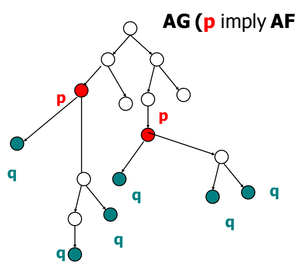
Derived Operators



52

Liveness: p - -> q *"p leads to q"*

AG (p imply AF q)



53

Timed CTL (a simplified version)

Syntax

$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid EX\,\phi \mid E[\phi\,U\,\phi] \mid A[\phi\,U\,\phi]$

where **p** ∈ AP (atomic propositions) **or Clock constraint**

54

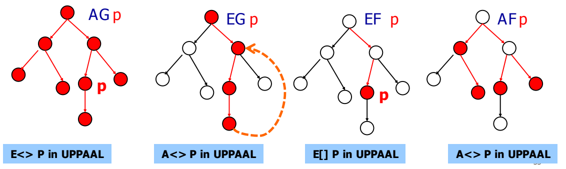
Timed CTL (a simplified version)

Syntax

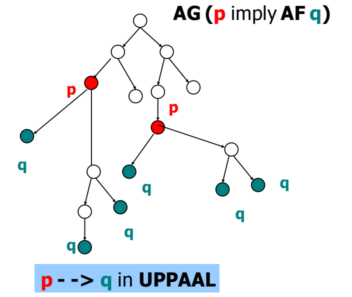
$\phi ::= p \mid \neg \phi \mid \phi \vee \phi \mid EX\ \phi \mid E[\phi\ U\ \phi] \mid A[\phi\ U\ \phi]$

where **p** ∈ AP (atomic propositions) **or Clock constraint**

Derived Operators



Derived Operators (cont.)



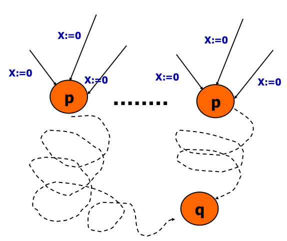
Bounded Liveness

[TACAS 98]

Verify: "whenever p is true, q should be true within 10 sec

P - - > (q and x<10)

Use extra clock x
Add x:=0 on all edges leading to P



Bounded Liveness/Responsiveness

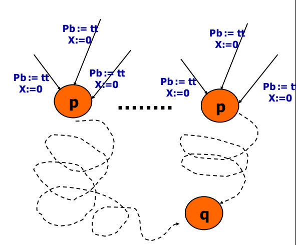
(reachability analysis, more efficient?)

[TACAS 98]

Verify: "whenever p is true, q should be true within 10 sec

AG $((P_b$ and $x>10)$ imply q)

Use extra clock x and boolean P_b
Add P_b := tt and x:=0 on all edges leading to location P



Bounded Liveness/Responsiveness

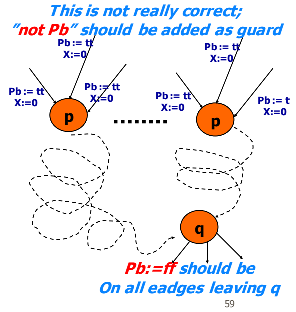
(reachability analysis, more efficient?)

[TACAS 98]

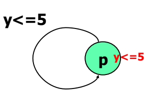
Verify: "whenever p is true, q should be true within 10 sec

AG $((P_b$ and $x>10)$ imply q)

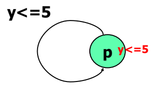
Use extra clock x and boolean P_b
Add P_b := tt and x:=0 on all edges leading to location P



Problem with Zenoness/Time-stop

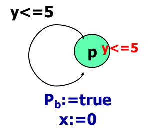


EXAMPLE



We want to specify "whenever P is true, Q should be true within 10 time units

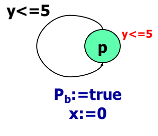
EXAMPLE



We want to specify "whenever P is true, Q should be true within 10 time units

AG ((P_b and x>10) imply Q)

EXAMPLE



We want to specify "whenever P is true, Q should be true within 10 time units

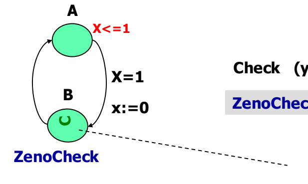
AG ((P_b and x>10) imply q)

is satisfied !!!

Solution with UPPAAL

Check Zeno-freeness by an extra observer

System || ZenoCheck



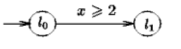
Check (yes means "no zeno loops")

ZenoCheck.A - - > ZenoCheck.B

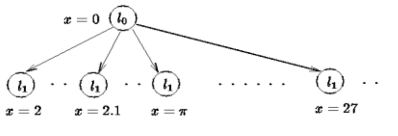
Committed location!

REACHABILITY ANALYSIS
using Regions

Infinite State Space!



gives rise to the infinite transition system:

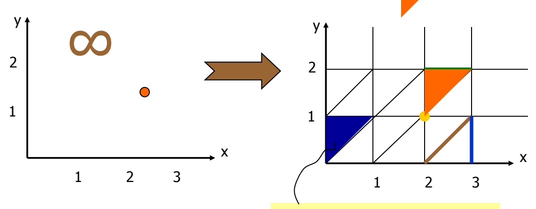


However, the reachability problem is decidable ☺ Alur&Dill 1991

Region: From infinite to finite

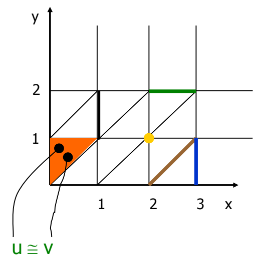
Concrete State
(n, x=2.2, y=1.5)

Symbolic state (region)
(n,)



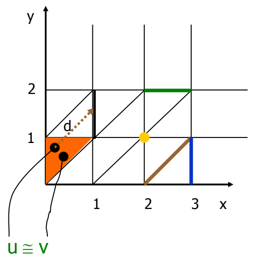
An equivalence class (i.e. a *region*)
There are only *finite* many such!!

Region equivalence (Intuition)



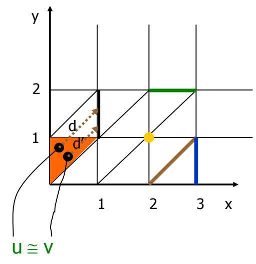
u ≅ v iff (l,u) and (l,v) may reach the same set of eqivalence classes

Region equivalence (Intuition)



u ≅ v iff (l,u) and (l,v) may reach the same set of eqivalence classes

Region equivalence (Intuition)

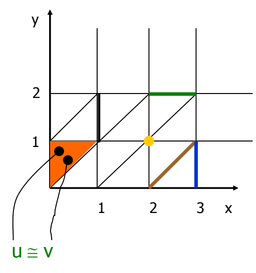


u ≅ v iff (l,u) and (l,v) may reach the same set of eqivalence classes

Region equivalence *[Alur and Dill 1990]*

- u,v are clock assignments
- u≈v iff
  - For all clocks x,
    either (1) u(x)>Cx and v(x)>Cx
    or (2) ⌊u(x)⌋=⌊v(x)⌋
  - For all clocks x, if u(x)<=Cx,
    {u(x)}=0 iff {v(x)}=0
  - For all clocks x, y, if u(x)<=Cx and u(y)<=Cy
    {u(x)}<= {u(y)} iff {v(x)}<= {v(y)}

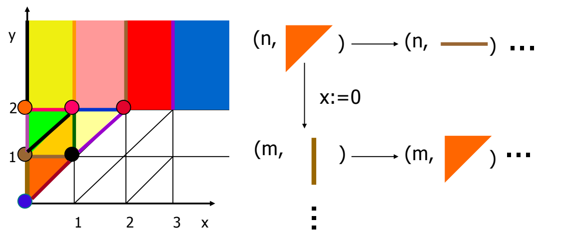
Region equivalence (alternatively)



u ≅ v iff u and v satisfy exactly the same set of constraints in the form of
xi ~ m and xi-xj ~ n
where ~ is in {<,>,≤,≥}
and m,n < MAX

This is not quite correct;
we need to consider the MAX
more carefully

Region Graph
Finite-State Transition System!!



OBS: there are only
Finite many regions

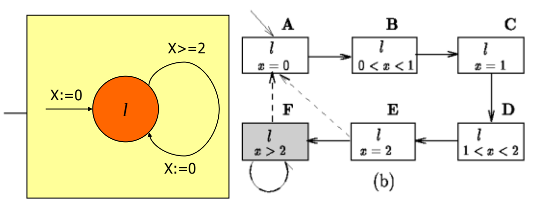
(m, [u]) ⟶ (n, [v]) if (m, u) ⟶ (n,v)

Theorem

u≈v implies
- u(x:=0) ≈ v(x:=0)
- u+n ≈ v+n for all natural number n
- for all d<1: u+d ≈ v+d' for some d'<1

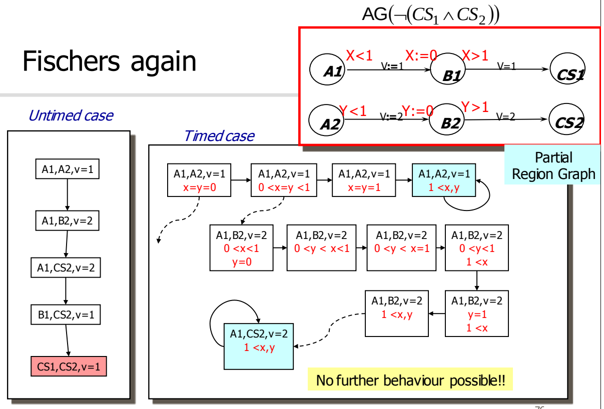
"Region equivalence' is preserved by "addition" and reset.
(also preserved by "subtraction" if clock values are "bounded")

Region graph of
a simple timed automata



Fischers again

$AG(\neg(CS_1 \wedge CS_2))$

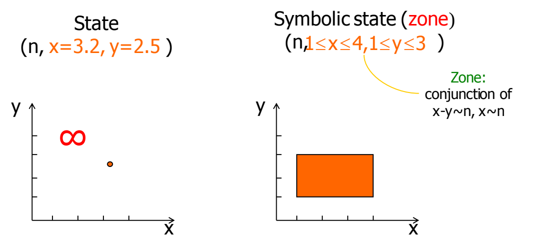


Problems with Region Construction

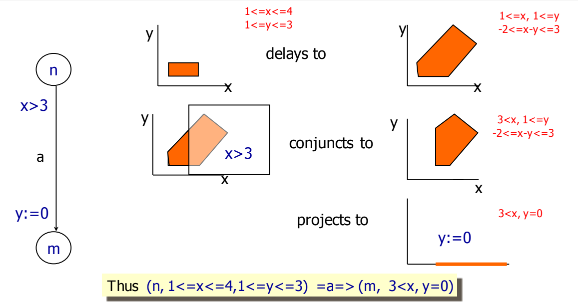
- Too many 'regions'
  - Sensitive to the maximal constants
  - e.g. x>1,000,000, y>1,000,000 as guards in TA
- The number of regions is highly exponential in the number of clocks and the maximal constants.

REACHABILITY ANALYSIS
using ZONES

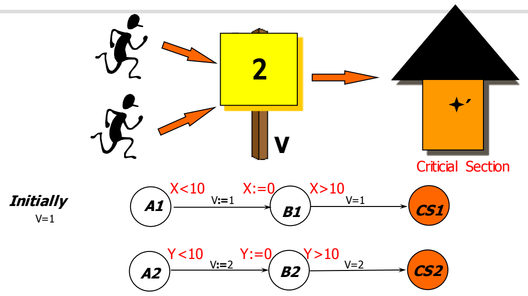
Zones: From infinite to finite



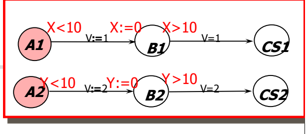
Symbolic Transitions



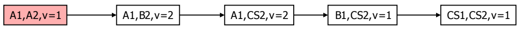
Fischer's Protocol
analysis using zones



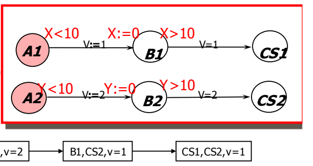
Fischers cont.



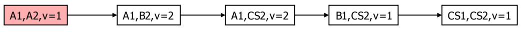
Untimed case



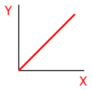
Fischers cont.



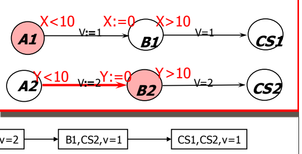
Untimed case



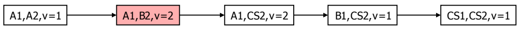
Taking time into account



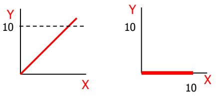
Fischers cont.



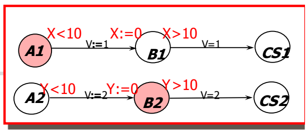
Untimed case



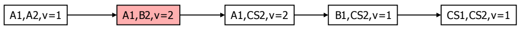
Taking time into account



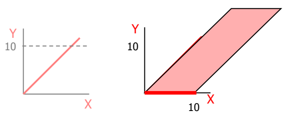
Fischers cont.



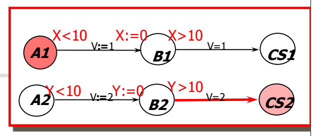
Untimed case



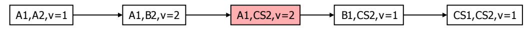
Taking time into account



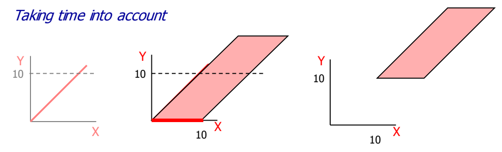
Fischers cont.



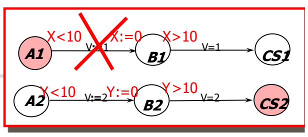
Untimed case



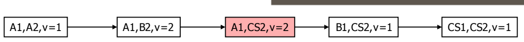
Taking time into account



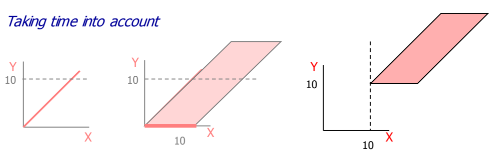
Fischers cont.



Untimed case



Taking time into account



Zones = Conjuctive constraints

- A zone Z is a conjunctive formula:
  g_1 & g_2 & ... & g_n
  where g_i may be $x_i \sim b_i$ or $x_i - x_j \sim b_{ij}$
- Use a zero-clock x_0 (constant 0), we have
  $\{x_i - x_j \sim b_{ij} \mid \sim$ is $<$ 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}

(We shall simply write Z instead [Z])

Operations on Zones

- Post-condition (Delay): SP(Z) or Z↑
  - [Z↑] = {u+d| d ∈ R, u∈[Z]}
- Pre-condition: WP(Z) or Z↓ (the dual of Z↑)
  - [Z↓] = {u| u+d∈[Z] for some d∈R}
- Reset: {x}Z or Z(x:=0)
  - [{x}Z] = {u[0/x] | u ∈[Z]}
- Conjunction
  - [Z&g]= [Z]∩[g]

Two more operations on Zones

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

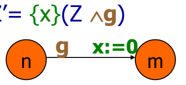
Theorem on Zones

The set of zones is closed under all zone operations

- That is, the result of the operations on a zone is a zone
- Thus, there will be a zone to represent the sets: $[Z\uparrow]$, $[Z\downarrow]$, $[\{x\}Z]$

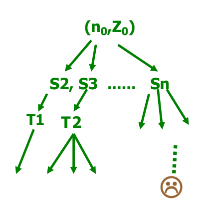
One-step reachability: Si ⁓ Sj

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

if 

- **Reach**: (n,Z) ⁓ (m,Z') if (n,Z) →→(m,Z')
- **Successors**(n,Z)={(m,Z') | (n,Z) ⁓ (m,Z'), $Z' \neq \emptyset$}

Now, we have a search problem



EF ☹

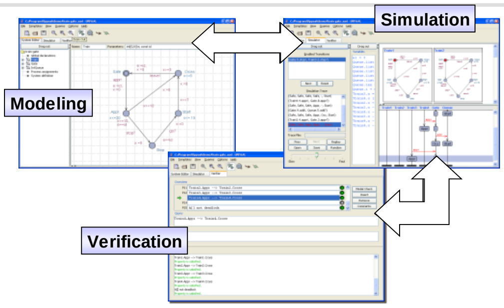
OUTLINE

- Model Checking in a Nutshell
- Timed automata and TCTL
- A UPPAAL Tutorial
  - Data stuctures & central algorithms
  - UPPAAL input languages

(Recent Work: Multicore Timing Analysis)

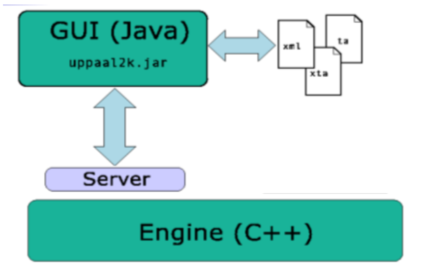
What's inside UPPAAL

UPPAAL Tool



97

Architecture of UPPAAL



Linux, Windows, Solaris, MacOS

98

Inside the UPPAAL tool

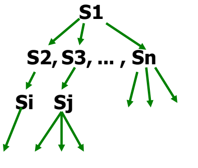
- Data Structures
  - DBM's (Difference Bounds Matrices)
  - Canonical and Minimal Constraints
- Algorithms
  - Reachability analysis
  - Liveness checking
- Verification Options

99

All Operations on Zones

(needed for verification)

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



100

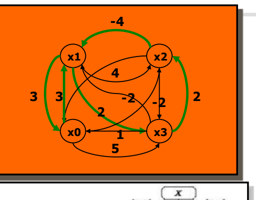
Zones = Conjuctive constraints

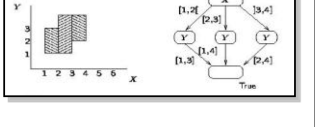
- A zone Z is a conjunctive formula:
  g_1 & g_2 & ... & g_n
  where g_i may be $x_i \sim b_i$ or $x_i - x_j \sim b_{ij}$
- Use a zero-clock x_0 (constant 0), we have
  $\{x_i - x_j \sim b_{ij} \mid \sim \text{ is } < \text{ or } \leq, i,j \leq n\}$
- This can be represented as a MATRIX, DBM (Difference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

101

Datastructures for Zones in UPPAAL

- Difference Bounded Matrices [Bellman58, Dill89]
- Minimal Constraint Form [RTSS97]
- Clock Difference Diagrams [CAV99]



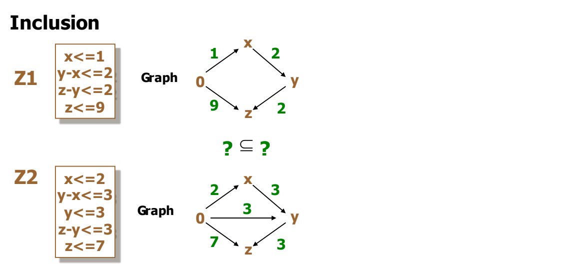


102

Canonical Datastructures for Zones

Difference Bounded Matrices

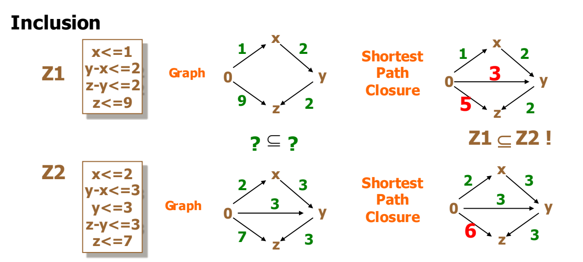
Bellman 1958, Dill 1989



Canonical Dastructures for Zones

Difference Bounded Matrices

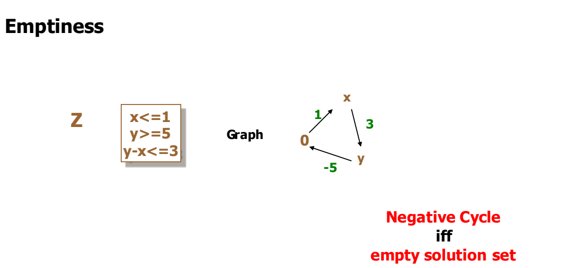
Bellman 1958, Dill 1989



Canonical Datastructures for Zones

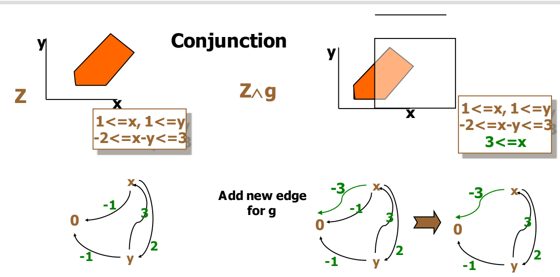
Difference Bounded Matrices

Bellman 1958, Dill 1989



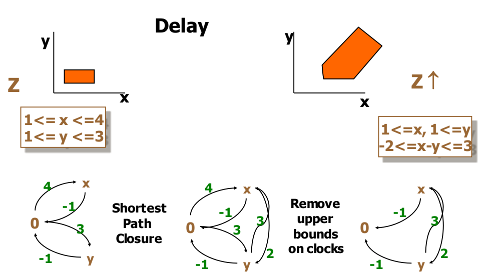
Canonical Datastructures for Zones

Difference Bounded Matrices



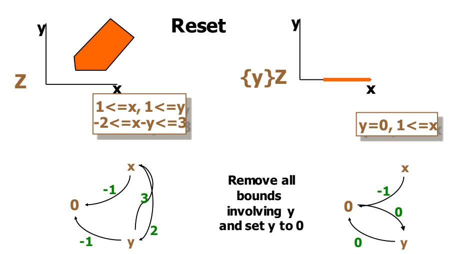
Canonical Dastructures for Zones

Difference Bounded Matrices



Canonical Datastructures for Zones

Difference Bounded Matrices

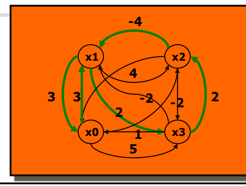


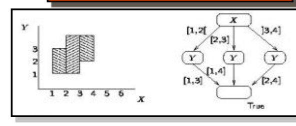
COMPLEXITY

- Computing the shortest path closure, the cannonical form of a zone: O(n^3) [Dijkstra's alg.]
- Run-time complexity, mostly in O(n)
  (when we keep all zones in cannonical form)

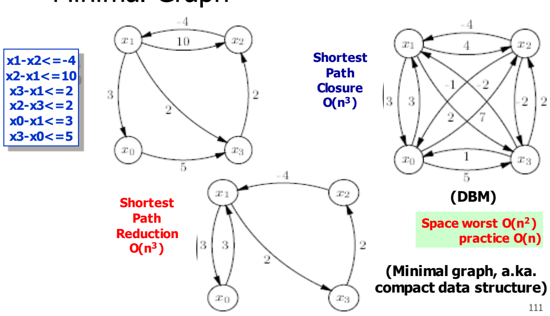
Datastructures for Zones in UPPAAL

- Difference Bounded Matrices
  [Bellman58, Dill89]
- Minimal Constraint Form
  [RTSS97]
- Clock Difference Diagrams
  [CAV99]

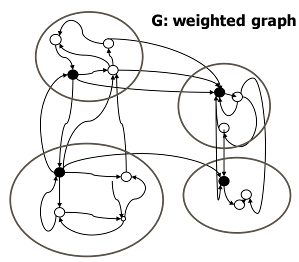




Minimal Graph



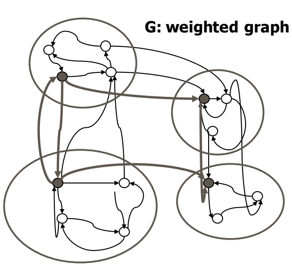
Graph Reduction Algorithm



G: weighted graph

1. Equivalence classes based on 0-cycles.

Graph Reduction Algorithm

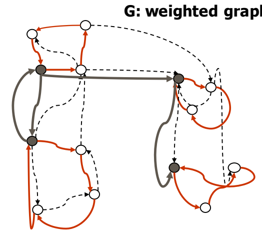


G: weighted graph

1. Equivalence classes based on 0-cycles.

2. Graph based on representatives.
Safe to remove redundant edges

Graph Reduction Algorithm



G: weighted graph

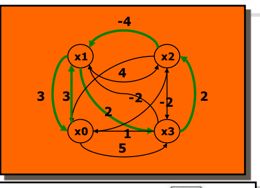
1. Equivalence classes based on 0-cycles.

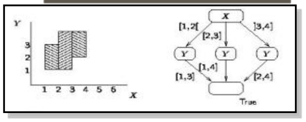
2. Graph based on representatives.
Safe to remove redundant edges

3. Shortest Path Reduction
=
One cycle pr. class
+
Removal of redundant edges between classes

Datastructures for Zones in UPPAAL

- Difference Bounded Matrices
  [Bellman58, Dill89]
- Minimal Constraint Form
  [RTSS97]
- Clock Difference Diagrams
  [CAV99]



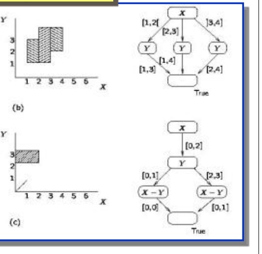


115

Other Symbolic Datastructures

- NDD's Maler et. al.
- CDD's UPPAAL/CAV99
- DDD's Møller, Lichtenberg
- Polyhedra HyTech
- ......

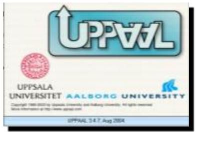
CDD-representations



116

Inside the UPPAAL tool

- Data Structures
  - DBM's (Difference Bounds Matrices)
  - Canonical and Minimal Constraints
- Algorithms
  - Reachability analysis
  - Liveness checking
- Verification Options



117

Timed CTL in UPPAAL

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

P ::= A.l | g_c | g_d | not p | p or p | p and p | p imply p

Process Location (a location in automaton A)

Clock constraint

predicate over data variables

SAFETY PROPERTIES

denotes
A[] (p imply A<> q)

118

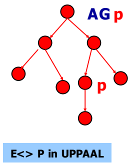
Timed CTL (a simplified version)

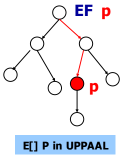
Syntax

ϕ **::=** **p** | $\neg\phi$ | $\phi \vee \phi$ | **EX** ϕ | **E[** ϕ **U** ϕ **]** | **A[** ϕ **U** ϕ **]**

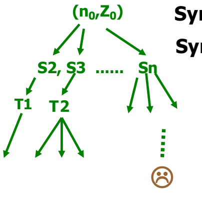
where p $\in$ **AP (atomic propositions) or Clock constraint**

Derived Operators





We have a search problem



Symbolic state

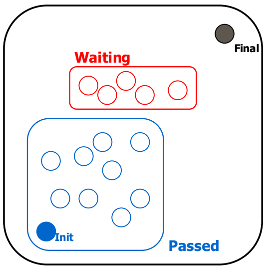
Symbolic transitions

Reachable?
E<> ☹

120

Forward Reachability

Init -> Final ?



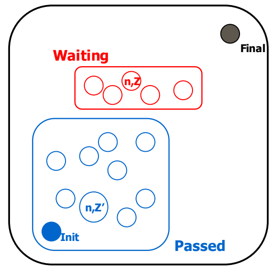
INITIAL Passed := Ø;
Waiting := {(n0,Z0)}

REPEAT
- pick (n,Z) in Waiting
- if for some Z' Z
(n,Z') in Passed then STOP
- else /explore/ add
{ (m,U) : (n,Z) => (m,U) }
to Waiting;
Add (n,Z) to Passed

UNTIL Waiting = Ø
or
Final is in Waiting

Forward Reachability

Init -> Final ?



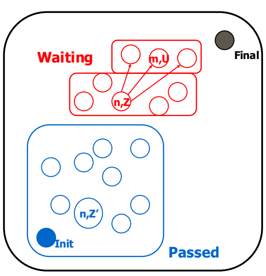
INITIAL Passed := Ø;
Waiting := {(n0,Z0)}

REPEAT
- pick (n,Z) in Waiting
- if for some Z' Z
(n,Z') in Passed then STOP
- else (explore) add
{ (m,U) : (n,Z) => (m,U) }
to Waiting;
Add (n,Z) to Passed

UNTIL Waiting = Ø
or
Final is in Waiting

Forward Reachability

Init -> Final ?



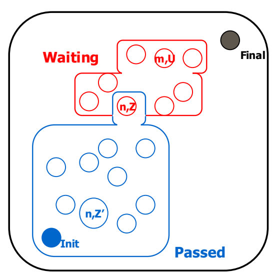
INITIAL Passed := Ø;
Waiting := {(n0,Z0)}

REPEAT
- pick (n,Z) in Waiting
- if for some Z' Z
(n,Z') in Passed then STOP
- else /explore/ add
{ (m,U) : (n,Z) => (m,U) }
to Waiting;
Add (n,Z) to Passed

UNTIL Waiting = Ø
or
Final is in Waiting

Forward Reachability

Init -> Final ?



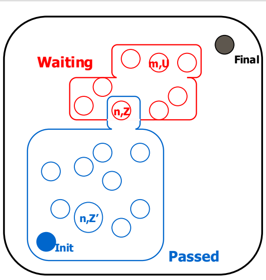
INITIAL Passed := Ø;
Waiting := {(n0,Z0)}

REPEAT
- pick (n,Z) in Waiting
- if for some Z' Z
(n,Z') in Passed then STOP
- else /explore/ add
{ (m,U) : (n,Z) => (m,U) }
to Waiting;
Add (n,Z) to Passed

UNTIL Waiting = Ø
or
Final is in Waiting

Forward Reachability

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
{ (m,U) : (n,Z) => (m,U) }
to Waiting;
Add (n,Z) to Passed

UNTIL Waiting = Ø
or
Final is in Waiting

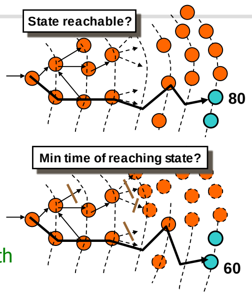
Further question

Can we find the path with shortest delay, leading to P ?
(i.e. a state satisfying P)

OBSERVATION:
Many scheduling problems can be phrased naturally as reachability problems for timed automata.

Verification vs. Optimization

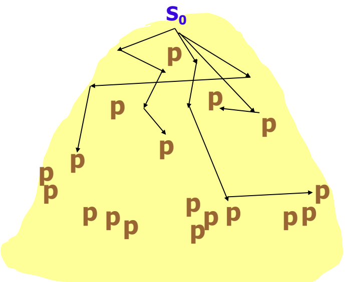
- Verification Algorithms:
  - Checks a logical property of the entire state-space of a model.
  - Efficient Blind search.
- Optimization Algorithms:
  - Finds (near) optimal solutions.
  - Uses techniques to avoid non-optimal parts of the state-space (e.g. Branch and Bound).
- Goal: solve opt. problems with verification.



OPTIMAL REACHABILITY

The maximal and minimal delay problem

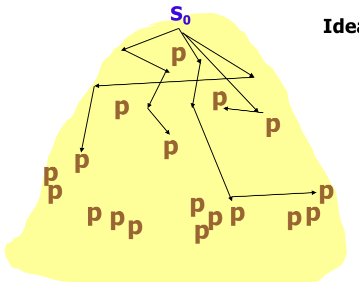
Find the trace leading to P with min delay



There may be a lot of pathes leading to P

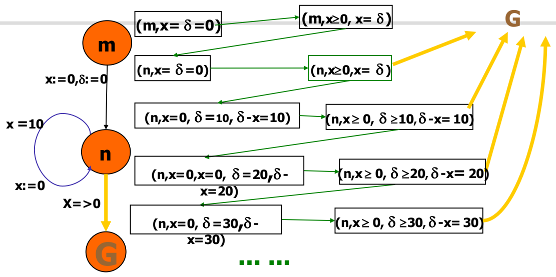
Which one with the shortest delay?

Find the trace leading to P with min delay



Idea: delay as "Cost" to reach a state, thus cost increases with time at rate 1

Example (min delay to reach G)

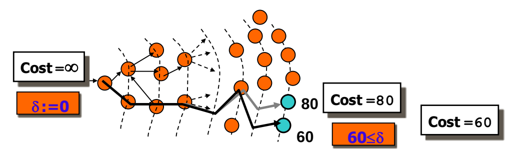


The minimal delay = 0 but the search may never terminate!

Problem: How to symbolically represent the zone C.

An Simple Algorithm for minimal-cost reachability

- State-Space Exploration + Use of global variable `Cost` and global clock δ
- Update `Cost` whenever goal state with **min(C) <** `Cost` is found:



- Terminates when entire state-space is explored.

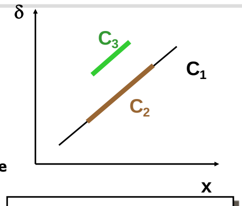
Problem: The search may never terminate!

Priced-Zone

- Cost = minimal total time
- C can be represented as the zone Z^δ, where:
  - Z^δ original (ordinary) DBM plus…
  - δ clock keeping track of the cost/time.
- Delay, Reset, Conjunction etc. on Z are the standard DBM-operations
- Delay-Cost is incremented by Delay-operation on Z^δ.

Priced-Zone

- Cost = min total time
- C can be represented as the zone Z^δ, where:
  - Z^δ is the original zone Z extended with the global clock δ keeping track of the cost/time.
  - Delay, Reset, Conjunction etc. on C are the standard DBM-operations
- But inclusion-checking will be different

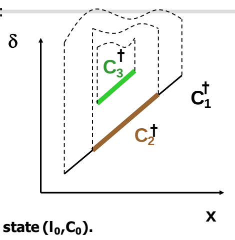


Then: $C_3 \sqsubseteq C_2 \sqsubseteq C_1$
But: $C_3 \not\subseteq C_2 \subseteq C_1$

Solution: $()^\dagger$-widening operation

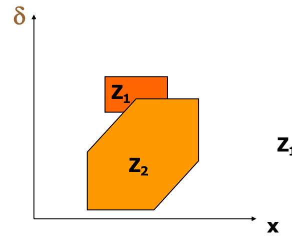
- $()^\dagger$ removes upper bound on the δ–clock:

  $C_3 \sqsubseteq C_2 \sqsubseteq C_1$
  $C_3^\dagger \subseteq C_2^\dagger \subseteq C_1^\dagger$



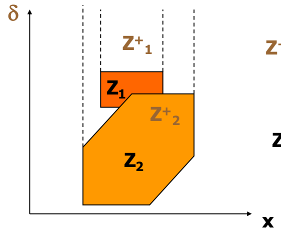
- In the Algorithm:
  - $\text{Delay}(C^\dagger) = (\text{Delay}(C^\dagger))^\dagger$
  - $\text{Reset}(x,C^\dagger) = (\text{Reset}(x,C^\dagger))^\dagger$
  - $C_1^\dagger \wedge g = (C_1^\dagger \wedge g)^\dagger$
- **It is suffices to apply $()^\dagger$ to the initial state (l_0,C_0).**

Example (widening for Min)



$Z_1 \not\sqsubseteq Z_2$

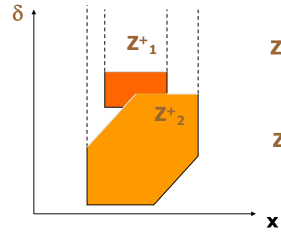
Example (widening for Min)



$Z^+ = \text{Widen}(Z)$

$Z_1 \not\sqsubseteq Z_2$

Example (widening for Min)



$Z^+ = \text{Widen}(Z)$

$Z^+_1 \subseteq Z^+_2$!
$Z_1 \sqsubseteq Z_2$

An Algorithm (Min)

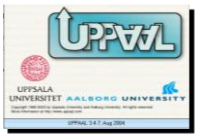
```
Cost:=∞, Pass := {}, Wait := {(l0,C0)}
while Wait ≠ {} do
   select (l,C) from Wait
   if (l,C) |= P and Min(C)<Cost then Cost:= Min(C)
   if (l,C) ⊑ (l,C') for some (l,C') in Pass then skip
       otherwise add (l,C) to Pass
       and forall (m,C') such that (l,C) ⇝ (m,C'):
         add (m,C') to Wait
Return Cost
```

One-step reachability relation

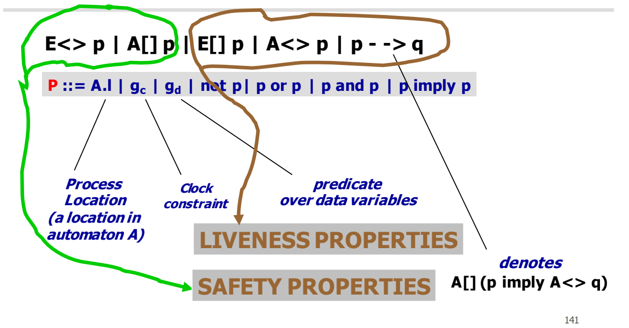
Output: `Cost` = the min cost of a found trace satisfying `P`.

Inside the UPPAAL tool

- Data Structures
  - DBM's (Difference Bounds Matrices)
  - Canonical and Minimal Constraints
- Algorithms
  - Reachability analysis
  - Liveness checking
- Verification Options



Timed CTL in UPPAAL



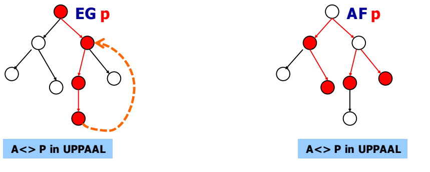
Timed CTL (a simplified version)

Syntax

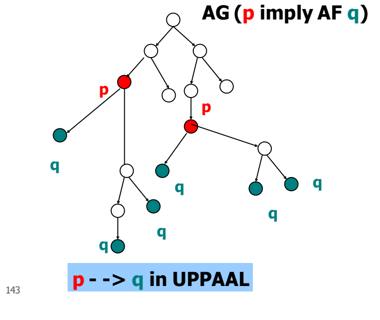
ϕ ::= p | $\neg\phi$ | $\phi \vee \phi$ | EX ϕ | E[ϕ U ϕ] | A[ϕ U ϕ]

where p ∈ AP (atomic propositions) or Clock constraint

Derived Operators



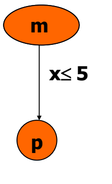
Derived Operators (cont.)



Question

A<> P

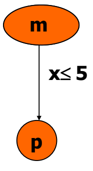
"P will be true for sure in future"



?? Does this automaton satisfy AF P

Note that

A<> P ***"P will be true for sure in future"***

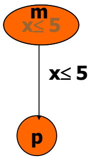


NO !!!! **there is a path:**
(m, x=0) → (m,x=1) → (m,2) ... (m,x=k) ...
Idling forever in location m

145

Note that

A<> P ***"P will be true for sure in future"***



This automaton satisfies AF P

146

Algorithm for checking A<> P **Eventually P**

Bouajjani, Tripakis, Yovine'97
On-the-fly symbolic model checking of TCTL

There is no cycle containing only states where p is false

147

Question: Time bound synthesis

A<> P **"P will be true eventually"**
But no time bound is given.

Assume AF P is satisfied by an automaton A.
Can we calculate the Max time bound?

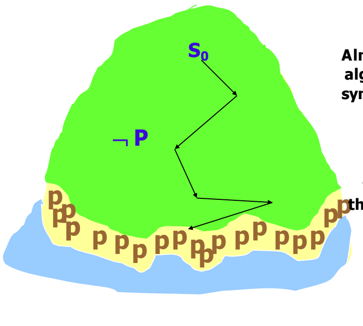
OBS: we know how to calculate the Min !

148

not available in the distributed version of UPPAAL

Assume A<> P is satisfied

Find the trace leading to P with the max delay



Almost the same algorithm as for synthesizing Min

We need to explore the Green part

149

An Algorithm (Max)

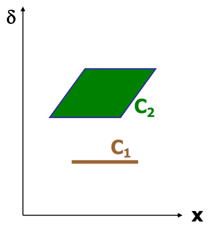
```
Cost:=0, Pass := {}, Wait := {(l0,C0)}
while Wait ≠ {} do
   select (l,C) from Wait
   if (l,C) |= P and Max(C)>Cost then Cost:= Max(C)
   else if forall (l,C') in Pass: C ⋢ C' then
      add (l,C) to Pass
      forall (m,C') such that (l,C) ⇝ (m,C'):
         add (m,C') to Wait
Return Cost
```

One-step reachability relation

Output: `Cost` = the max cost of a found trace satisfying `P`.
BUT: $\sqsubseteq$ is defined on zones where the lower bound of "cost" is removed

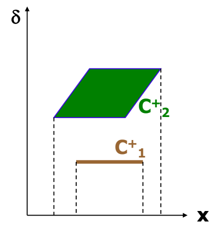
150

Zone-Widening operation for Max



$C_1 \not\subseteq C_2$

Zone-Widening operation for Max



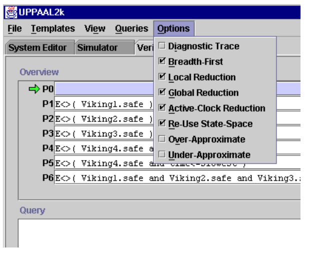
$C_1 \not\subseteq C_2$

$C^+_1 \subseteq C^+_2$

$C_1 \sqsubseteq C_2$!

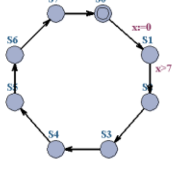
Inside the UPPAAL tool

- Data Structures
  - DBM's (Difference Bounds Matrices)
  - Canonical and Minimal Constraints
- Algorithms
  - Reachability analysis
  - Liveness checking
- Verification Options



- **Diagnostic Trace**
- **Breadth-First**
- **Depth-First**
- **Local Reduction**
- **Active-Clock Reduction**
- **Global Reduction**
- **Re-Use State-Space**
- **Over-Approximation**
- **Under-Approximation**

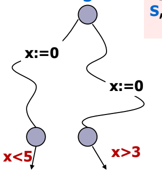
Inactive (passive) Clock Reduction



x is only *active* in location S1

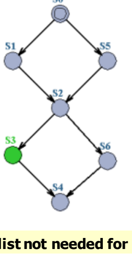
Definition

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



Global Reduction

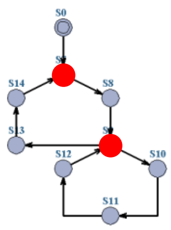
(When to store symbolic state)



However, Passed list useful for efficiency

No Cycles: Passed list not needed for *termination*

Global Reduction
(When to store symbolic state) [RTSS97]



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

157

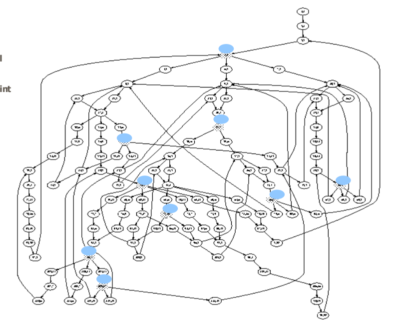
To Store Or Not To Store?

[RTSS97,CAV03]

117 $states_{total}$
↓
81 $states_{entrypoint}$
↓
9 states

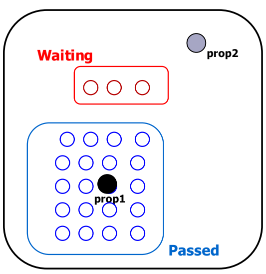
Time OH less than 10%

(need to re-explore some states)



158

Reuse of State Space



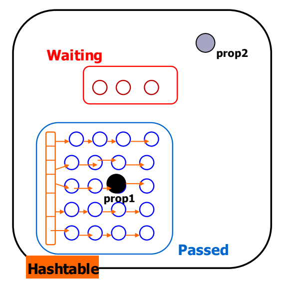

```
A[]  prop1

A[]  prop2
A[]  prop3
A[]  prop4
A[]  prop5
 .
 .
 .
A[]  propn
```

Search in existing Passed list before continuing search

Which order to search?

159

Reuse of State Space



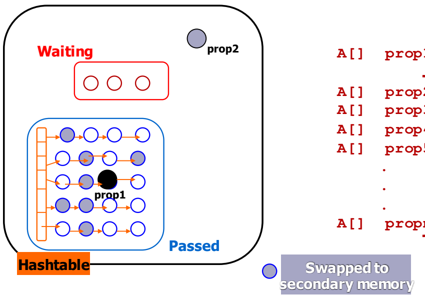

```
A[]  prop1

A[]  prop2
A[]  prop3
A[]  prop4
A[]  prop5
 .
 .
 .
A[]  propn
```

Search in existing Passed list before continuing search

Which order to search?

160

Reuse of State Space



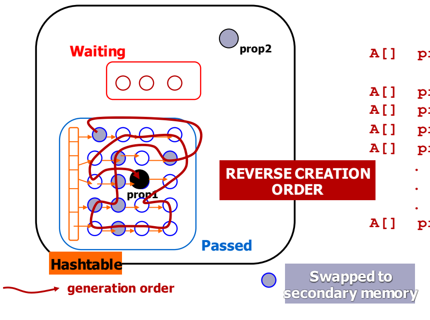

```
A[]  prop1

A[]  prop2
A[]  prop3
A[]  prop4
A[]  prop5
 .
 .
 .
A[]  propn
```

Search in existing Passed list before continuing search

Which order to search?

161

Reuse of State Space



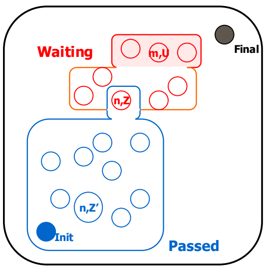

```
A[]  prop1

A[]  prop2
A[]  prop3
A[]  prop4
A[]  prop5
 .
 .
 .
A[]  propn
```

Search in existing Passed list before continuing search

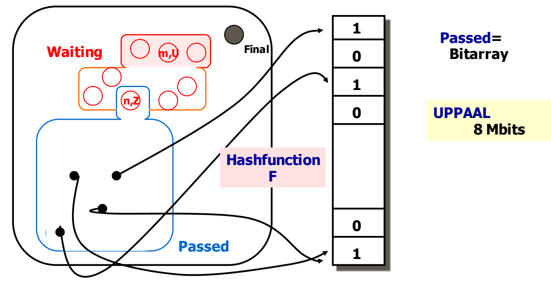
Which order to search?

162

Under-approximation
Bitstate Hashing (Holzman,SPIN)



Under-approximation
Bitstate Hashing



Bit-state Hashing

```
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
    { (m,U) : (n,Z) => (m,U) }
    to Waiting;
    Add (n,Z) to Passed

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

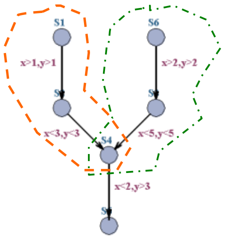
Passed(F(n,Z)) = 1

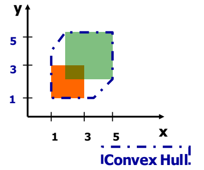
Passed(F(n,Z)) := 1

Under Approximation
(good for finding Bugs quickly, debugging)

- Possitive answer is safe (you can trust)
  - You can trust your tool if it tells:
    a state is reachable (it means Reachable!)
- Negative answer is Inconclusive
  - You should not trust your tool if it tells:
    a state is non-reachable
  - Some of the branch may be terminated by conflict (the same hashing value of two states)

Over-approximation
Convex Hull





Over-Approximation
(good for safety property-checking)

- Possitive answer is Inconclusive
  - a state is reachable means Nothing
    (you should not trust your tool when it says so)
  - Some of the transitions may be enabled by Enlarged zones
- Negative answer is safe
  - a state is not reachable means Non-reachable
    (you can trust your tool when it says so)

Now, you can go home

- Download and use UPPAAL or
- Start to implement your own model checker