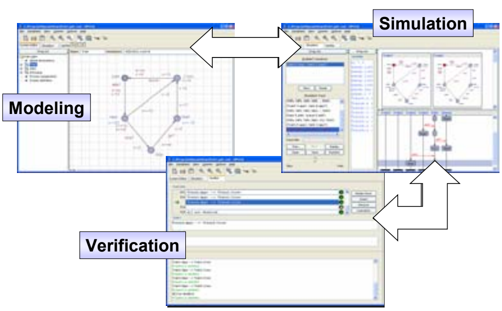
OUTLINE

- A Brief Introduction
  - Motivation ... what are the problems to solve
  - CTL, LTL and basic model-checking algorithms
- Timed Systems
  - Timed automata and verification problems
  - UPPAAL tutorial (1): data stuctures and algorithms
  - UPPAAL tutorial (2): input languages
  - TIMES: From models to code ”guaranteeing” timing constraints
- Further topics/Recent Work
  - Systems with buffers/queues [CAV 2006]

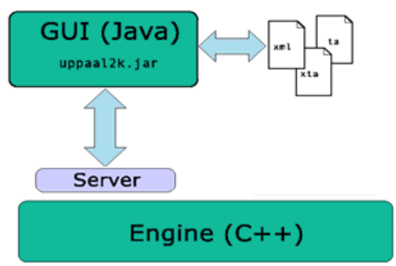
UPPAAL tool

- Developed jointly by Uppsala & Aalborg University
- >>20,000 downloads since 1995

UPPAAL Tool



Architecture of UPPAAL



Linux, Windows, Solaris, MacOS

Lecture 4 & 5 & 6

UPPAAL Tutorial (1)

What’s inside UPPAAL: algorithms and data structures

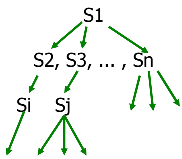
What’s inside UPPAAL

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

All Operations on Zones

(needed for verification)

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



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)

Datastructures for Zones in UPPAAL

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

Canonical Datastructures for Zones

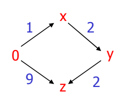
Difference Bounded Matrices Bellman 1958, Dill 1989

Inclusion

Z1

$x<=1$
$y-x<=2$
$z-y<=2$
$z<=9$

Graph

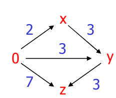


? ⊆ ?

Z2

$x<=2$
$y-x<=3$
$y<=3$
$z-y<=3$
$z<=7$

Graph



Canonical Dastructures for Zones

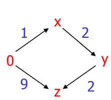
Difference Bounded Matrices Bellman 1958, Dill 1989

Inclusion

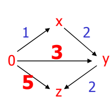
Z1

$x<=1$
$y-x<=2$
$z-y<=2$
$z<=9$

Graph



Shortest Path Closure



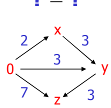
? ⊆ ?

Z1 ⊆ Z2 !

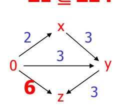
Z2

$x<=2$
$y-x<=3$
$y<=3$
$z-y<=3$
$z<=7$

Graph



Shortest Path Closure



Canonical Datastructures for Zones

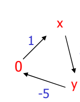
Difference Bounded Matrices Bellman 1958, Dill 1989

Emptiness

Z

$x<=1$
$y>=5$
$y-x<=3$

Graph

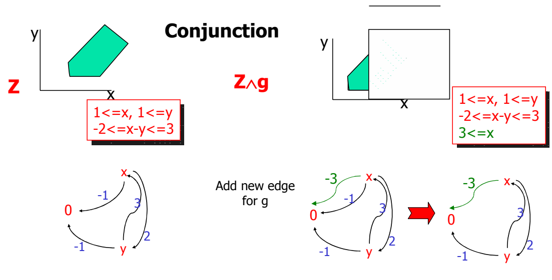


Negative Cycle
iff
empty solution set

Compact

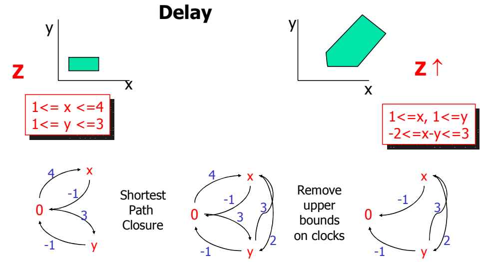
Canonical Datastructures for Zones

Difference Bounded Matrices



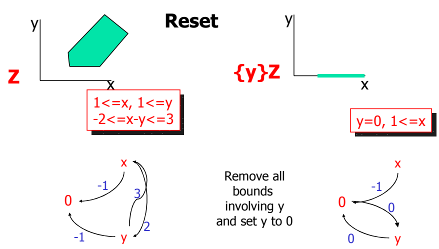
Canonical Dastructures for Zones

Difference Bounded Matrices



Canonical Datastructures for Zones

Difference Bounded Matrices



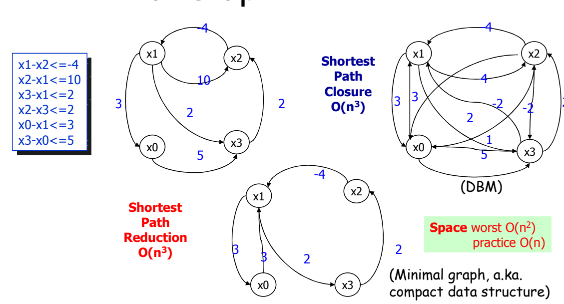
COMPLEXITY

- 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)

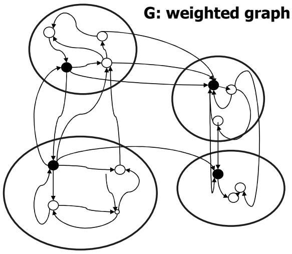
Datastructures for Zones in UPPAAL

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

Minimal Graph



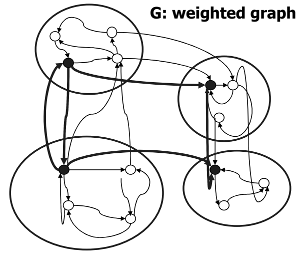
Graph Reduction Algorithm



G: weighted graph

1. Equivalence classes based on 0-cycles.

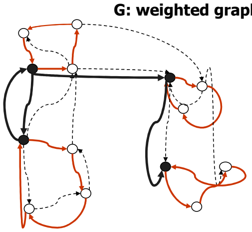
Graph Reduction Algorithm



G: weighted graph

1. Equivalence classes based on 0-cycles.
2. Graph based on representatives.
   Safe to remove redundant edges

Graph Reduction Algorithm



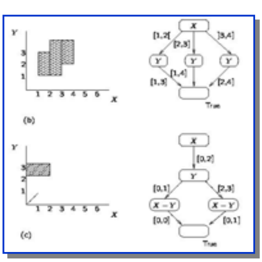
G: weighted graph

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]

CDD: Clock Decision Diagrams



Goals:

Compact representation of
- disjunctive formulas or
- union of DBM's

Best use of sharing

Other Symbolic Datastructures

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

What's inside UPPAAL

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

Timed CTL in UPPAAL

EF p | AG p | EG p | AF 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

p leads to q *denotes* **AG (p imply AF q)**

Timed CTL in UPPAAL

EF p | AG p | EG p | AF 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)

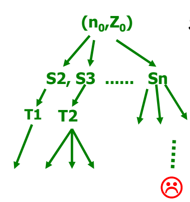
Clock constraint

predicate over data variables

SAFETY PROPERTIES

p leads to q *denotes* **AG (p imply AF q)**

We have a search problem



Symbolic state

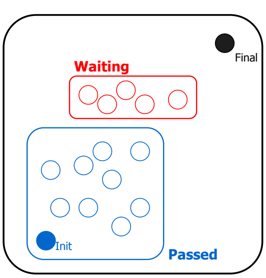
Symbolic transitions

Reachable?

EF ☹

Forward Reachability

Init -> Final ?



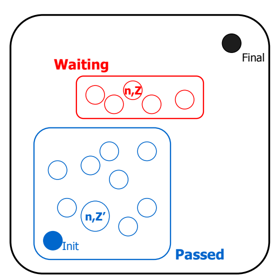
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

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 ?



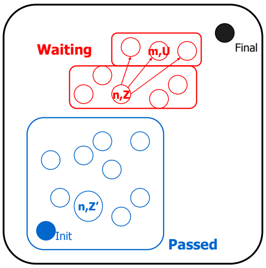
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

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 ?



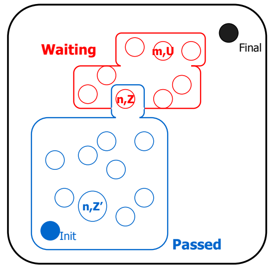
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

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 ?



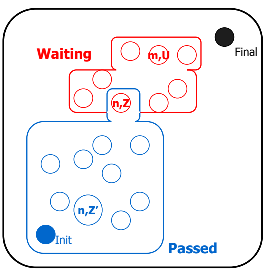
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

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**

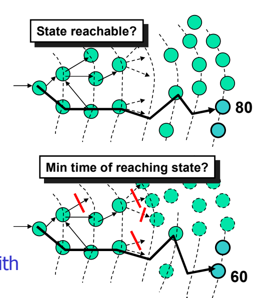
Further question

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

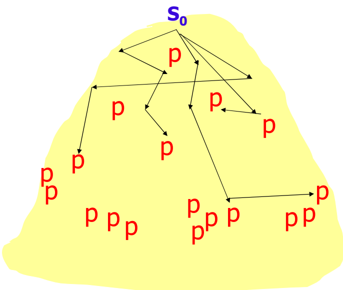
- Verification Algorithms:
  - Checks a logical property of the entire state-space of a model.
  - Efficient Blind search.
- Optimization Algorithms:
  - Finds (near) optimal solutions.
  - Uses techniques to avoid non-optimal parts of the state-space (e.g. Branch and Bound).
- Goal: solve opt. problems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



OPTIMAL REACHABILITY

The maximal and minimal delay problem

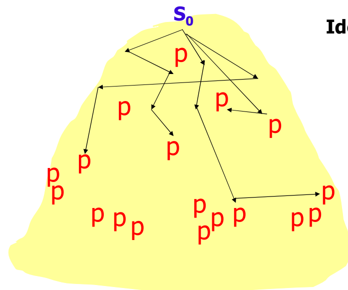
Find the trace leading to P with min delay



There may be a lot of pathes leading to P

Which one with the shortest delay?

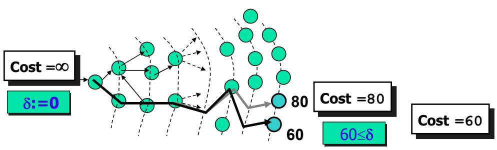
Find the trace leading to P with min delay



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

An Simple Algorithm for minimal-cost reachability

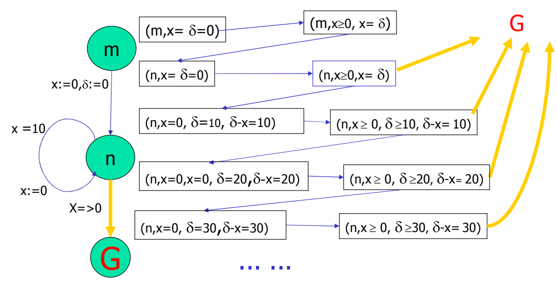
- State-Space Exploration + Use of global variable `Cost` and global clock δ
- Update `Cost` whenever goal state with **min(C)** < `Cost` is found:



- Terminates when entire state-space is explored.

Problem: The search may never terminate!

Example (min delay to reach G)



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

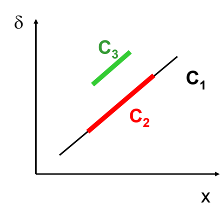
Problem: How to **symbolically** represent the zone **C.**

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 original 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.



Then: $C_3 \sqsubseteq C_2 \sqsubseteq C_1$

But: $C_3 \nsubseteq C_2 \subseteq C_1$

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

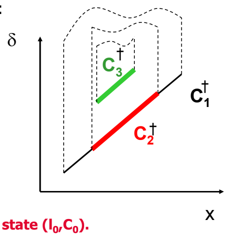
- $()^{\dagger}$ removes upper bound on the δ–clock:

  $C_3 \sqsubseteq C_2 \sqsubseteq C_1$

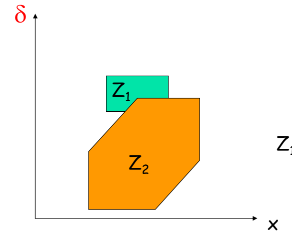
  $C_3^{\dagger} \subseteq C_2^{\dagger} \subseteq C_1^{\dagger}$

- In the Algorithm:
  - $Delay(C^{\dagger}) = (Delay(C^{\dagger}))^{\dagger}$
  - $Reset(x,C^{\dagger}) = (Reset(x,C^{\dagger}))^{\dagger}$
  - $C_1^{\dagger} \wedge g = (C_1^{\dagger} \wedge g)^{\dagger}$

  - **It is suffices to apply $()^{\dagger}$ to the initial state (l_0,C_0).**

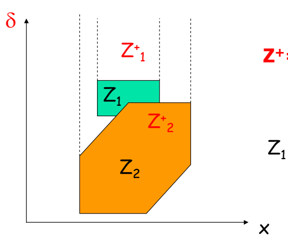


Example (widening for Min)



$Z_1 \not\subseteq Z_2$

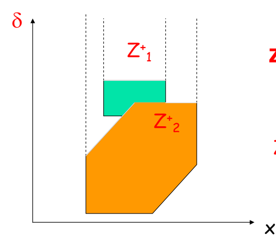
Example (widening for Min)



Z^{+}= Widen(Z)

$Z_1 \not\subseteq Z_2$

Example (widening for Min)



Z^{+}= Widen(Z)

$Z^{+}_1 \subseteq Z^{+}_2$!

$Z_1 \sqsubseteq Z_2$

An Algorithm (Min)

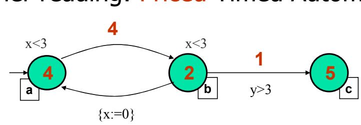
```
Cost:=∞, Pass := {}, Wait := {(l0,C0)}
while Wait ≠ {} do
   select (l,C) from Wait
   if (l,C) ⊨ P and Min(C)<Cost then Cost:= Min(C)
   if (l,C) ⊑(l,C') for some (l,C') in Pass then skip
       otherwise add (l,C) to Pass
       and forall (m,C') such that (l,C)⇝(m,C'):
         add (m,C') to Wait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

One-step reachability relation

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

Problem: How to **symbolically** represent the zone **C.**

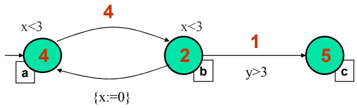
Further reading: Priced Timed Automata [Larsen et al]



- Timed Automata + Costs on transitions and locations.
- Uniformly Priced = Same cost in all locations (edges may have different costs).

- Cost of performing transition: Transition cost.
- Cost of performing delay **d**: (**d** x location cost).

Priced Timed Automata



Trace:

(**a**,x=y=0) —4→ (**b**,x=y=0) —ε(2.5), 2.5 x 2→ (**b**,x=y=2.5) —0→ (**a**,x=0,y=2.5)

Cost of Execution Trace:

Sum of costs: **4 + 5 + 0 = 9**

Problem: Finding the minimum cost of reaching c !

Inside the UPPAAL tool

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

Timed CTL in UPPAAL

EF p | AG p | EG p | AF 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

LIVENESS PROPERTIES

SAFETY PROPERTIES

p leads to q *denotes* **AG (p imply AF q)**

LIVENESS Properties

in UPPAAL

F ::= EG p | AF p | p - -> q

Possibly always P is equivalent to (¬ AF ¬ P)

Eventually P is equivalent to (¬ EG ¬ P)

P **leads to** Q is equivalent to AG (P imply AF Q)

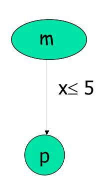
Algorithm for checking AF P

Eventually P

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

Question

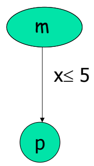
AF P — *"P will be true for sure in future"*



?? Does this automaton satisfy AF P

Note that

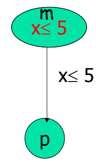
AF P "P will be true for sure in future"



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

Note that

AF P "P will be true for sure in future"



This automaton satisfies AF P

Liveness Algorithm

Bouajjani, Tripakis, Yovine, 97

```
proc Eventually(S0, φ) ≡
   ST := ∅
   Passed := ∅
   Search(delay(S0, ¬φ))
   exit(true)
end
● proc Search(S) ≡   if empty(S) then exit(true) fi
   if loop(S, ST) then exit(false) fi
   S := S ∧ ¬φ
   push(ST, S)
   if unbounded(S) ∨ deadlocked(S) then
                        exit(false) fi
   if ∀S' ∈ Passed : S ⊈ S'
     then foreach S' : S ⇒ S' do
              Search(delay(S', ¬φ))
           od
   fi
   Passed := Passed ∪ {pop(ST)}
end
```


Question: Time bound synthesis

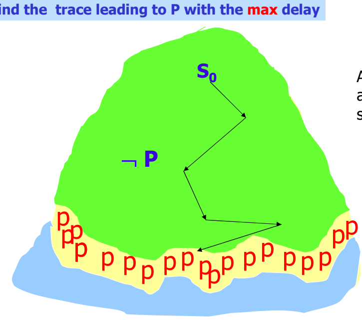
AF P "P will be true eventually"
But no time bound is given.

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

OBS: we know how to calculate the Min !

Assume AF 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

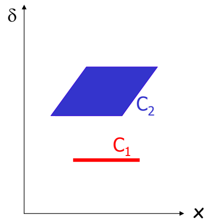
An Algorithm (Max)

```
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

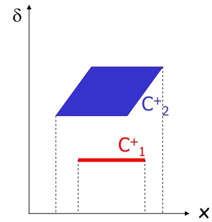
Output: Cost = the min cost of a found trace satisfying P.
BUT: ⊑ is defined on zones where the lower bound of "cost" is removed

Zone-Widening operation for Max



$C_1 \not\subseteq C_2$

Zone-Widening operation for Max



$C_1 \not\subseteq C_2$

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

$C_1 \sqsubseteq C_2$!

End of Basic Algorithms ....

How about termination?

What's inside UPPAAL

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

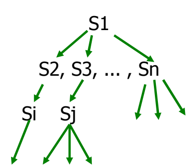
Lecture 5

Zone Normalization

Operations on Zones

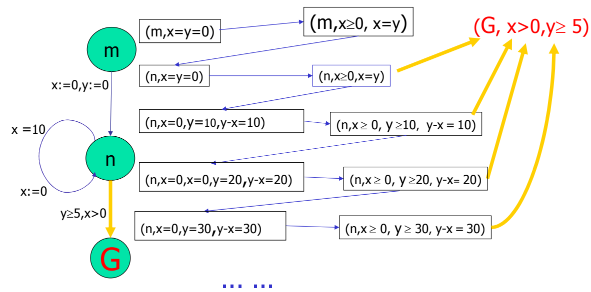
(needed for verification)

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



We need one more zone operation:
normalization to terminate the searching process

Example: is G reachable?

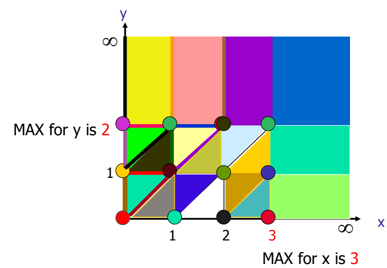


Normalization of Zones

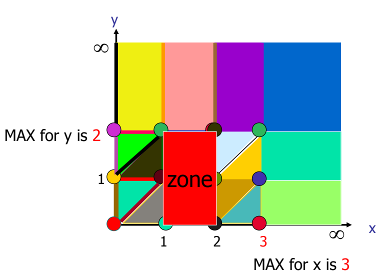
To guarantee termination

Region Equivalence:

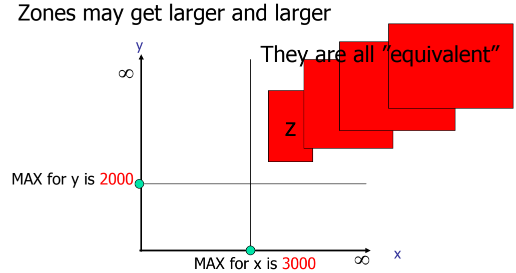
The same color means "equivalent" Alur&Dill 1990



Zone = "set of regions"



Zones may get larger and larger



Normalization of Zones

1. To have a canonical representation for the equivalent zones
2. Any guard g, not enabled by Z, should not be enabled by the normalized Z

g∧Z = empty iff g∧Normalized(Z)=empty

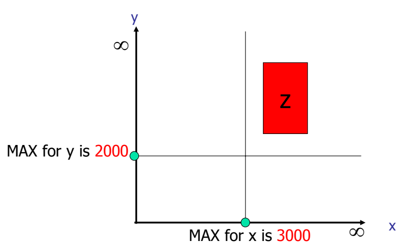
The K-Normalization in UPPAAL

based on maximal constants

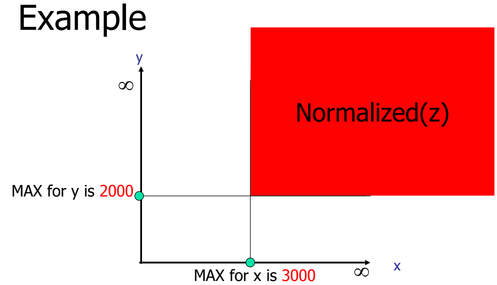
K-Normalized(Z)= {u| vϵZ, u ~ v}

Easy to compute this via constraints

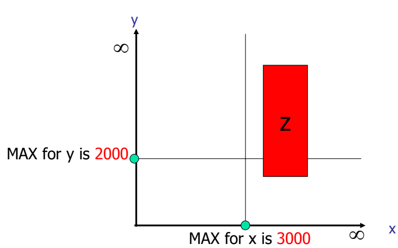
Example



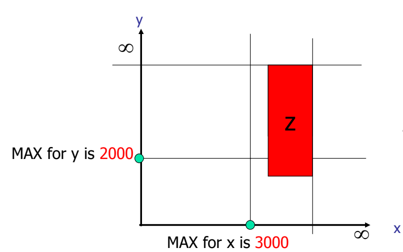
Example



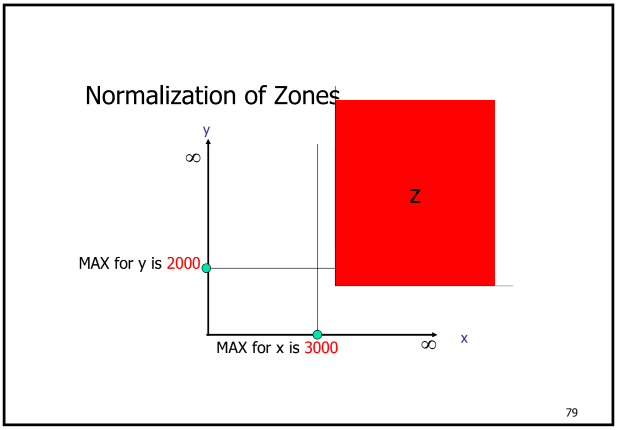
Normalization of Zones

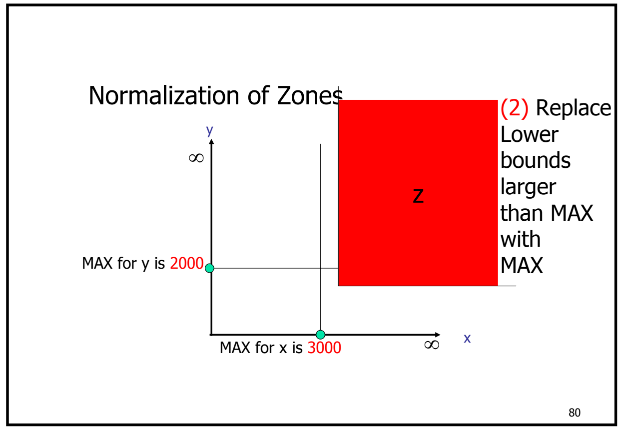


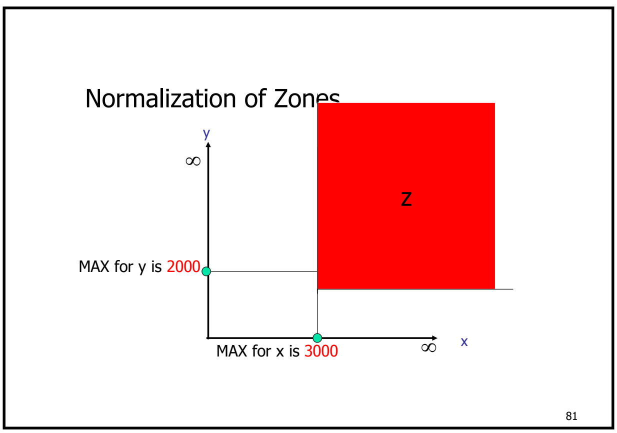
Normalization of Zones

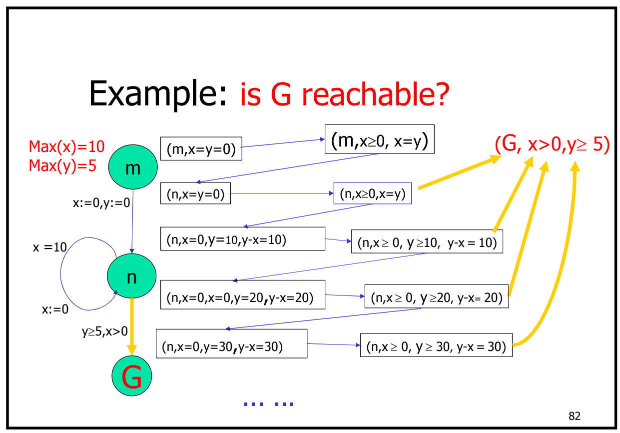


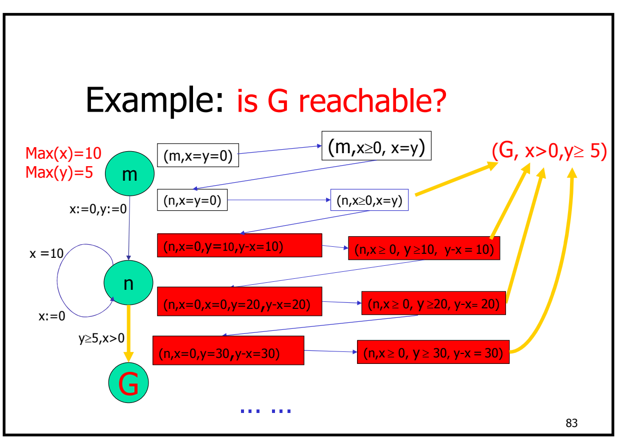
(1) Remove upper bounds larger than MAX's

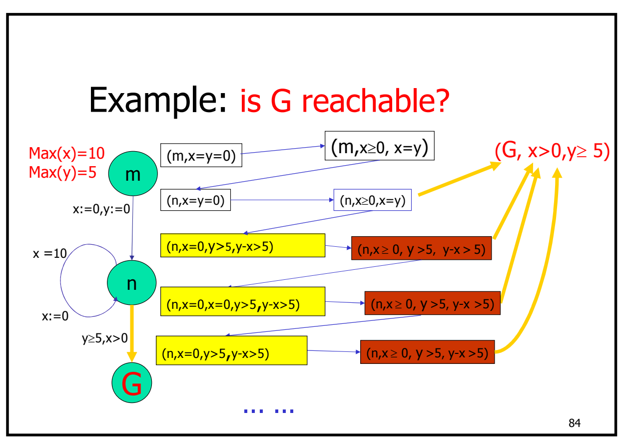

Normalization of Zones
y
∞
Z
MAX for y is 2000
MAX for x is 3000
∞
x
79


Normalization of Zones
y
∞
Z
(2) Replace Lower bounds larger than MAX with MAX
MAX for y is 2000
MAX for x is 3000
∞
x
80

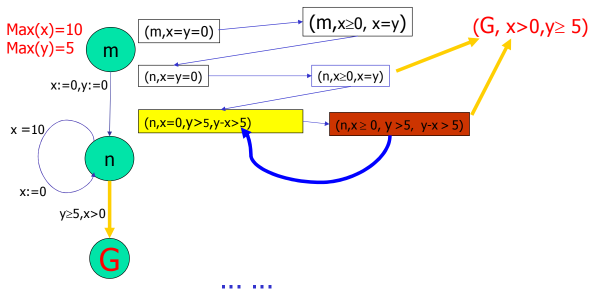

Normalization of Zones
y
∞
Z
MAX for y is 2000
MAX for x is 3000
∞
x
81


Example: is G reachable?
Max(x)=10
Max(y)=5
m
x:=0,y:=0
x =10
n
x:=0
y≥5,x>0
G
(m,x=y=0)
(m,x≥0, x=y)
(n,x=y=0)
(n,x≥0,x=y)
(n,x=0,y=10,y-x=10)
(n,x≥ 0, y ≥10, y-x = 10)
(n,x=0,x=0,y=20,y-x=20)
(n,x ≥ 0, y ≥20, y-x= 20)
(n,x=0,y=30,y-x=30)
(n,x ≥ 0, y ≥ 30, y-x = 30)
(G, x>0,y≥ 5)
… …
82


Example: is G reachable?
Max(x)=10
Max(y)=5
m
x:=0,y:=0
x =10
n
x:=0
y≥5,x>0
G
(m,x=y=0)
(m,x≥0, x=y)
(n,x=y=0)
(n,x≥0,x=y)
(n,x=0,y=10,y-x=10)
(n,x≥ 0, y ≥10, y-x = 10)
(n,x=0,x=0,y=20,y-x=20)
(n,x ≥ 0, y ≥20, y-x= 20)
(n,x=0,y=30,y-x=30)
(n,x ≥ 0, y ≥ 30, y-x = 30)
(G, x>0,y≥ 5)
… …
83


Example: is G reachable?
Max(x)=10
Max(y)=5
m
x:=0,y:=0
x =10
n
x:=0
y≥5,x>0
G
(m,x=y=0)
(m,x≥0, x=y)
(n,x=y=0)
(n,x≥0,x=y)
(n,x=0,y>5,y-x>5)
(n,x≥ 0, y >5, y-x > 5)
(n,x=0,x=0,y>5,y-x>5)
(n,x ≥ 0, y >5, y-x >5)
(n,x=0,y>5,y-x>5)
(n,x ≥ 0, y >5, y-x >5)
(G, x>0,y≥ 5)
… …
84

Example: is G reachable?



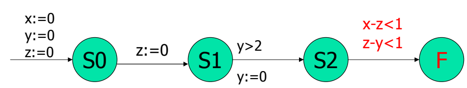
The K-normalization

- First compute the shortest path closure of a zone
- Remove all constraints in the form:
  x <(≤) m or x-y <(≤) n
  where m, n>Cx
- Replace all constraints in the form:
  x >(≥) m or x-y >(≥) n
  where m, n>Cx
  with x > Cx or x-y >Cx

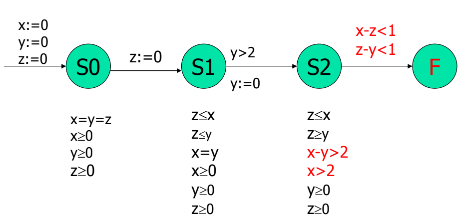
This is the normalization

- Implemented in UPPAAL, and
- Works for automata with guards like x ~ c
- Over-approximation for automata with guards like x – y ~ c

The counter example



The example (cont.)



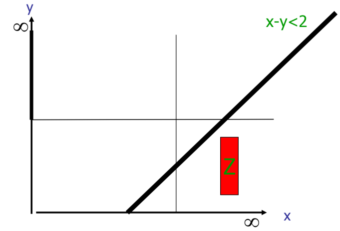
F is not reachable!

Before 2000, UPPAAL would have told you: *F is reachable*

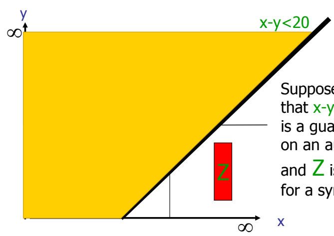
This was a bug unfortuately

Why doesn't this work?

The Problem with Clock Difference Constraints

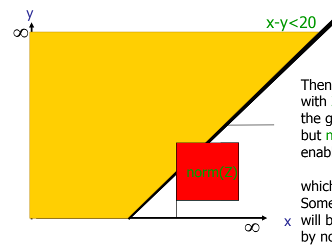


The Problem with Clock Difference Constraints



Suppose that x-y<20 is a guard on an automaton and Z is a zone for a symbolic state

The Problem with Clock Difference Constraints

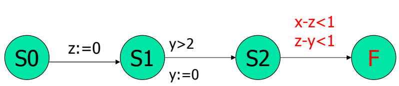


Then the state with Z disables the guard x-y<2 but norm(Z) will enable it,

which means Some transitions will be enabled by normalization

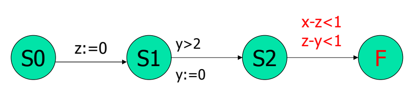
This violates the second condition on normalization of zones

The example revisited



at S2, we have x-y>2 and x>2 which disables the guard (x-z<1 & z-y<1) implying x-y<2; thus F is not reachable

Why the tools would tell: F is reachable



at S2, we have x-y>2 and x>2 which disables the guard

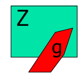
x-z<1 & z-y<1 i.e. x-y<2; thus F is not reachable

As the maximal const for x is 1, x-y>2 and x>2 is normalized to x-y>1 and x>1 which enables x-z<1 & z-y<1 and F is reachable (a wrong answer from the tool !)

The normalization based on the MAXIMAL constants doesn't work for clock difference constraints

Some observations

- $Z \subseteq$ norm(Z)
  - It is at least an "over-approximation"
  - Thus a reply with the form: a state is not reachable can be trusted
  - But a reply saying that a state is reachable may be wrong
- a guard g is not enabled by Z, i.e. $Z \wedge g$ is empty should not be enabled either by norm(Z)



Z enables g

Normalization of Zones

1. To have a canonical representation for the equivalent zones
2. Any guard g, not enabled by Z, should not be enabled either by the normalized Z, that is:

$g \wedge Z$ = empty iff $g \wedge$Normalized(Z)=empty

We need more care to guarantee the 2nd condition when difference constraints involved

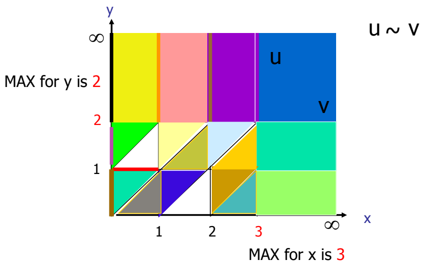
Normalization of Zones

1. To have a canonical representation for the equivalent zones
2. Any guard g, not enabled by Z, should not be enabled either by the normalized Z, that is:

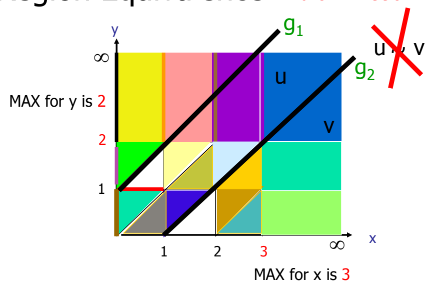
$g \wedge Z$ = empty iff $g \wedge$Normalized(Z)=empty

SOLUTION

Region Equivalence Alur&Dill 1990

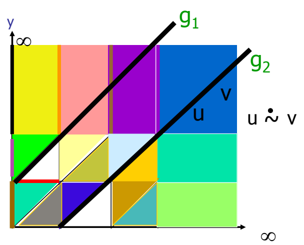


Region Equivalence Alur&Dill 1990



Refined Region Equivalence

Finite many g_i's



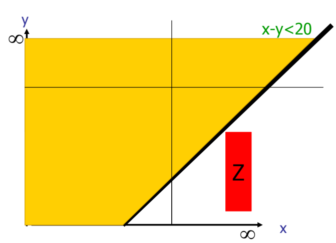
$u \dot{\sim} v$ if (1) $u \sim v$ and (2) $g_i(u)$ iff $g_i(v)$

New Normalization

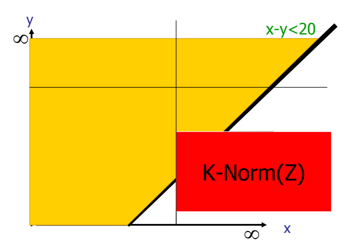
Normalized(Z)= {u| v∈Z, $u \dot{\sim} v$ }

The question is how to compute this via constraints

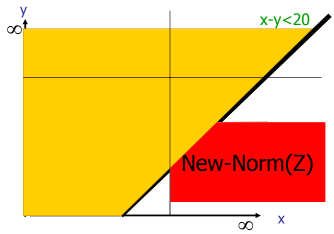
Example



Example

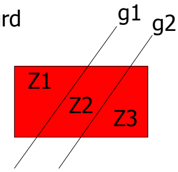


Example



In general, splitting is needed

Z: a zone to be normalized
g: a difference constraint in a gaurd



Split(Z) = {Z1, Z2, ... Zn}

so that either $g \wedge Zi$ is empty
or $g \wedge Zi$ = Zi

We should split Z for ALL g

New-Normalized(Zi)

- If $g \wedge Zi$ = empty then
  New-Normalized(Zi) = k-Norm(Zi)$\wedge \neg$g
- Otherwise
  New-Normalized(Zi) = k-Norm(Zi)

The normalization algorithm

- Collect all the maximal constants K
- Collect all the difference constraints G
- For any Z, the normalized version is computed as follows:
  - Split(Z) = {Z1 ... Zn} for all g in G
    such that $Zi \wedge g$=empty or $Zi \wedge g$=Zi
  - Norm(Zi)=K-Norm(Zi)
  - Repeat for all g in G such that $Zi \wedge g$ = empty
    Norm(Zi)=Norm(Z) $\wedge \neg$g

New-Norm(Z) = { Norm(Z1) ... Norm(Zn) }

The number of "Normalized Zones" is bounded

By the number of regions !

FACTS

- The refined region equivalence induces only FINITE many regions
  - Therefore, finite many Normalized zones
  - This guarantees termination
- No guards (difference constraints) will be enabled by the new normalization operator
  - This guarantees soundness