UPPAAL tutorial

- What's inside UPPAAL
- The UPPAAL input languages

UPPAAL tool

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

UPPAAL Tool

- Modeling
- Simulation
- Verification

Architecture of UPPAAL

- GUI (Java)
- Engine (C++)
- Linux, Windows, Solaris, MacOS

What's inside UPPAAL

OUTLINE

- Data Structures
  - DBMs (Difference Bounds Matrices)
  - Canonical and Minimal Constraints
- Algorithms
  - Reachability analysis
  - Liveness checking
  - Verification Options
All Operations on Zones
(needed for verification)
- Transformation
- Conjunction
- Post condition (delay)
- Reset
- Consistency Checking
- Inclusion
- Emptiness

Zones = Conjunctive constraints
- A zone Z is a conjunctive formula:
  \[ g_1 \land g_2 \land \ldots \land g_n \]
  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
\[ Z_1 \subseteq Z_2 \]

Emptiness
Negative Cycle
iff
empty solution set
Canonical Datastructures for Zones

Difference Bounded Matrices

Conjunction

Add new edge for y

Delay

Remove upper bounds on clocks

COMPLEXITY

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

Datastructures for Zones in UPPAAL

- Difference Bounded Matrices
  [Bultan08, Dill99]
- Minimal Constraint Form
  [RTSS97]
- Clock Difference Diagrams
  [CAV99]
Graph Reduction Algorithm

1. Equivalence classes based on 0-cycles.
2. Graph based on representatives. Safe to remove redundant edges.
3. Shortest Path Reduction
   One cycle per 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]

Other Symbolic Datastructures

- NDD’s Maler et al.
- CDD’s UPPAAL/CAV99
- DDD’s Holst, Lichtenburg
- Polyhedra HyTech
- ......
Timed CTL in UPPAAL

\[ P ::= A.l | gc | gd | \neg p | p \lor p | p \land p | p \implies p \]

Process Location (a location in automaton A)

Clock constraint

Predicate over data variables

\[ p \text{ leads to } q \quad \text{denotes} \quad AG \ (p \implies AF q) \]

SAFETY PROPERTIES

\[ F ::= EF \ P \ | \ AG \ P \]

Reachability

Invariant = \neg EF \neg P

Thus, AG P is also checked by reachability analysis!

We have a search problem

Symbolic state

Symbolic transitions

Reachable?

EF

Forward Reachability

\[ \text{Init} \rightarrow \text{Final} \ ? \]

\[ \text{INITIAL} \quad \text{Passed} := \emptyset; \quad \text{Waiting} := \{(n_0,Z_0)\} \]

\[ \text{REPEAT} \]

- pick \ (n,Z) in Waiting
  - if for some \ z \geq 2 \ (n,z) in Passed then STOP
  - else (explore) add \ \{(m,U) : (n,Z) \Rightarrow (m,U)\} \ to Waiting
  - Add \ (n,Z) \ to Passed

\[ \text{UNTIL} \quad \text{Waiting} = \emptyset \]

or

Final is in Waiting

Forward Reachability

\[ \text{Init} \rightarrow \text{Final} \ ? \]

\[ \text{INITIAL} \quad \text{Passed} := \emptyset; \quad \text{Waiting} := \{(n_0,Z_0)\} \]

\[ \text{REPEAT} \]

- pick \ (n,Z) in Waiting
  - if for some \ z \geq 2 \ (n,z) in Passed then STOP
  - else (explore) add \ \{(m,U) : (n,Z) \Rightarrow (m,U)\} \ to Waiting
  - Add \ (n,Z) \ to Passed

\[ \text{UNTIL} \quad \text{Waiting} = \emptyset \]

or

Final is in Waiting
**Forward Reachability**

**Init -> Final ?**

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

REPEAT

- pick \((n,Z)\) in Waiting
  - if for some \(Z' \supseteq Z\)
    \((n,Z')\) in Passed
    STOP
  - else /explore/ add \\{(m,U) : (n,Z) \Rightarrow (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 paths leading to P.

Which one with the shortest 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 \( \text{min}(C) < \text{Cost} \).

Terminates when entire state-space is explored.

Problem: The search may never terminate!

Example (min delay to reach G)

Cost = min total time

C can be represented as the zone \( Z_\delta \), where:
- \( Z_\delta \) is the original zone \( Z \) extended with the global clock \( \delta \) keeping track of the cost/time.
- Delay, Reset, Conjunction etc. on \( Z \) are the standard DBM-operations.
- But inclusion-checking will be different.

Priced-Zone

- Cost = minimal total time
- C can be represented as the zone \( Z_\delta \), where:
  - \( Z_\delta \) is the original zone \( Z \) extended with the global clock \( \delta \) keeping track of the cost/time.
  - Delay, Reset, Conjunction etc. on \( Z \) are the standard DBM-operations.
- But inclusion-checking will be different.
Solution: \( \varphi^+ \)-widening operation

- \( \varphi^+ \) removes upper bound on the \( \delta \)-clock:
  \[
  C_1 \subseteq C_2 \subseteq C_1^+
  \]

- In the Algorithm:
  - \( \text{Delay}(C) = \{ \text{Delay}(C') \}^+ \)
  - \( \text{Reset}(x,C) = \{ \text{Reset}(x,C') \}^+ \)
  - \( C_1^+ - \varphi = \{ C_1^+ - \varphi \}^+ \)

- It is sufficient to apply \( \varphi^+ \) to the initial state \((s_0, C_0)\).

Example (widening for Min)

\[
\begin{align*}
Z_1 \subseteq Z_2 \\
Z_2 \subseteq Z_1^+ \\
Z_1^+ \xrightarrow{\delta} Z_2 \\
Z_1 \xrightarrow{\delta} Z_2^+ \\
Z_1^+ \subseteq Z_2
\end{align*}
\]

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

\( Z_1 \not\subseteq Z_2 \)

An Algorithm (Min)

Cost:=\( \infty \), Pass := \{ \}, Wait := \{ (l_0, C_0) \}
while Wait \( \neq \{ \} \) do
  select \((l, C)\) from Wait
  if \((l, C) = P\) and Min(C) < Cost then
    Cost := Min(C)
  if \((l, C) \xrightarrow{\delta} (l, C')\) for some \((l, C')\) in Pass then skip
  otherwise add \((l, C)\) to Pass
  and forall \((m, C')\) such that \((l, C) \xrightarrow{\delta} (m, C')\):
    add \((m, C')\) to Wait
Return Cost

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

Further reading: Priced Timed Automata

- 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 \( d \): \( d \times \text{location cost} \).
Priced Timed Automata

Trace:
(a,x,y=0) \rightarrow (b,x,y=0, t=2.5) \rightarrow (b,x,y=2.5, t=2) \rightarrow (a,x=0, y=2.5, t=0)

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
- Verification Options

Timed CTL in UPPAAL

EF $p$ | AG $p$ | EG $p$ | AF $p$ | $p \rightarrow q$

$F ::= \text{EG } p \mid \text{AF } p \mid p \rightarrow q$

LIVENESS Properties

Possibly always $P$ is equivalent to $\neg \text{AF } \neg P$

Eventually $P$ is equivalent to $\neg \text{EG } \neg P$

P leads to $Q$ is equivalent to $\text{AG } (P \implies \text{AF } Q)$

LIVENESS Properties

SAFETY PROPERTIES

Algorithm for checking $\text{AF } P$

Eventually $P$

Bouajjani, Tripathi, Yovine '97

On-the-fly symbolic model checking of TCTL

Question

"P will be true for sure in future"

?? Does this automaton satisfy $\text{AF } P$
Note that

\[ AF \neg P \]

"P will be true for sure in future"

\[ x \leq 5 \]

\[ m \]

NO !!!! there is a path:
\[(m, x=0) \rightarrow (m,x=1) \rightarrow (m,2) \ldots (m, x=k) \ldots \]
Idling forever in location m

This automaton satisfies \[ AF \neg P \]

\[ x \leq 5 \]

\[ m \]

\[ p \]

Liveness Algorithm

Passed ST Unexplored

AF \phi

\neg \phi

S

Bouajjani, Tripakis, Yovine, 97

Liveness Algorithm

Passed ST Unexplored

AF \phi

\neg \phi

S

Liveness Algorithm

Passed ST Unexplored

AF \phi

\neg \phi

S

Liveness Algorithm

Passed ST Unexplored

AF \phi

\neg \phi

S

Liveness Algorithm

Passed ST Unexplored

AF \phi

\neg \phi

S
Liveness Algorithm

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```

```
if empty(S) then exit(true) fi
```

```
if ST = 0 goto ST
```
Question: Time bound synthesis

\[ \text{AF P} \quad \text{“P will be true eventually”} \]
But no time bound is given.

Assume \( \text{AF P} \) is satisfied by an automaton \( A \).
Can we calculate the Max time bound?

OBS: we know how to calculate the Min!

An Algorithm (Max)

\[
\text{Cost} := 0, \text{Pass} := \emptyset, \text{Wait} := \{(l_0, C_0)\}
\]
while \( \text{Wait} \neq \emptyset \) do

select \((l, C)\) from \( \text{Wait} \)

if \((l, C) = P \text{ and Max}(C) > \text{Cost}\) then \(\text{Cost} := \text{Max}(C)\)
else if \(\forall (l, C') \in \text{Pass}: C \subseteq C'\) then

add \((l, C)\) to \( \text{Pass} \)

\(\forall (m, C') \text{ such that } (l, C) \not\rightarrow (m, C'):\)

add \((m, C')\) to \( \text{Wait} \)

Return \(\text{Cost}\)

Output: \(\text{Cost}\) = the min cost of a found trace satisfying \( P \).

BUT \(\subseteq\) is defined on zones where the lower bound of "cost" is removed.

Zone-Widening operation for Max

\[
C_1 \subseteq C_2 \quad C_1 \not\subseteq C_2
\]

Inside the UPPAAL tool

- Data Structures
  - DBM’s (Difference Bounds Matrices)
  - Canonical and Minimal Constraints
- Algorithms
  - Reachability analysis
  - Liveness checking
  - Termination
- Verification Options
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 $S_1$
- $x$ is inactive at $S_i$ if on all paths from $S_i$, $x$ is always reset before being tested.

Global Reduction

(When to store symbolic state)

No Cycles: Passed list not needed for domination

However, Passed list useful for efficiency

No Cycles: Passed list not needed for domination

Global Reduction

(When to store symbolic state)

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

Reuse of State Space

- $A[1] \text{ prop1}$
- $A[1] \text{ prop2}$
- $A[1] \text{ prop3}$
- $A[1] \text{ prop4}$
- $A[1] \text{ prop5}$
- ...$
- A[1] \text{ propn}$

Which order to search?

To Store Or Not To Store?

- 117 states, 81 states entrypoint
- 81 states
- 9 states
- Time OH less than 10%
- (need to re-explore some states)
Reuse of State Space

Under-approximation

Bitstate Hashing (Holzman, SPIN)

Under-approximation

Bitstate Hashing

Reuse of State Space
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
(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)