UPPAAL tutorial

Architecture of UPPAAL

Inside the UPPAAL tool

What’s inside UPPAAL

All Operations on Zones (needed for verification)
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
  $\{x_i - x_j \sim b_{ij} \mid \sim \text{ is } \langle \text{ or } \leq \}^{i,j=1\ldots n}$
- 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)
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
  [Bellman58, Dill89]
- 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 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]

Other Symbolic Datastructures

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

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

\[
\begin{align*}
E<> p & | A[] p & | E[] p & | A<> p & | p - -> q \\
P ::= A | g | g_u | not p | p or p | p and p | p imply p
\end{align*}
\]

Process Location (a location in automaton A)
Clock constraint
Predicate over data variables
denotes
A[] (p imply A<> q)

SAFETY PROPERTIES
Timed CTL (a simplified version)

Syntax
\[ \phi ::= p \mid \neg \phi \mid \phi \lor \phi \mid EX \phi \mid E[\phi U \psi] \mid A[\phi U \psi] \]

where \( p \in AP \) (atomic propositions)

Derived Operators

We have a search problem

Forward Reachability

Init -> Final ?

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

REPEAT
- pick \( (n, Z) \) in Waiting
- if for some \( Z' \) \( Z \ni Z' \) in Passed then 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

Passed

Waiting

Final

n, Z

m, U

n, Z

m, U

Init -> Final ?

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

REPEAT
- pick \( (n, Z) \) in Waiting
- if for some \( Z' \) \( Z \ni Z' \) in Passed then 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

Passed

Waiting

Final

n, Z

m, U
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

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 \( \delta \)
- Update Cost whenever goal state with \( \min(C) < \text{Cost} \) is found:
- Terminates when entire state-space is explored.

Problem: The search may never terminate!

Example (min delay to reach \( G \))

Problem: How to symbolically represent the zone \( C \).

Priced-Zone

- Cost = minimal total time
- \( C \) can be represented as the zone \( Z_1 \), where:
  - \( Z_1 \): original (ordinary) DBM plus...
  - \( \delta \): 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 \( Z \).
- But inclusion-checking will be different

Solution: \( \top^1 \)-widening operation

- \( \top^1 \) removes upper bound on the \( \delta \)-clock:
  - \( C_1 \subseteq C_2 \subseteq C_1 \)
  - Then: \( C_1 \subseteq C_2 \subseteq C_1 \)
  - But: \( C_1 \nsubseteq C_2 \subseteq C_1 \)
  - It is sufficient to apply \( \top^1 \) to the initial state \((x_0, C_2)\).
### Example (widening for Min)

\[ Z \in Z \]

\[ Z \sqsupseteq Z \]

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

\[ Z_1 \subseteq Z_2 \]

\[ Z_1 \in Z_2 \]

### Example (widening for Min)

\[ Z \in Z \]

\[ Z \sqsupseteq Z \]

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

\[ Z_1 \subseteq Z_2 \]

\[ Z_1 \in Z_2 \]

### An Algorithm (Min)

\[
\begin{align*}
\text{Cost} & := \infty, \ Pass := \{\}, \ Wait := \{(l_0, C_0)\} \\
\text{while} \ Wait \neq {} \ \text{do} & \\
& \quad \select \ (l, C) \ \text{from} \ Wait \\
& \quad \text{if} \ (l, C) = P \ \text{and} \ \text{Min}(C) < \text{Cost} \ \text{then} \ Cost := \text{Min}(C) \\
& \quad \text{if} \ (l, C) \notin (l, C') \ \text{for some} \ (l, C') \ \text{in} \ Pass \ \text{then} \ \text{skip} \\
& \quad \text{otherwise} \ \text{add} \ (l, C) \ \text{to} \ Pass \\
& \quad \text{and} \ \forall (m, C') \ \text{such that} \ (l, C) \not\sqsubseteq (m, C') \\
& \quad \quad \text{add} \ (m, C') \ \text{to} \ Wait \\
\text{Return} \ Cost
\end{align*}
\]

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

### Inside the UPPAAL tool

- **Data Structures**
  - DBMs (Difference Bounds Matrices)
  - Canonical and Minimal Constraints

- **Algorithms**
  - Reachability analysis
  - Liveness checking
  - Verification Options

### Timed CTL in UPPAAL

\[
\begin{align*}
\end{align*}
\]

- **Process Location**
  - (a location in automaton A)

- **Clock Constraint**

- **Predicate over data variables**

- **Liveness Properties**

- **Safety Properties**
Timed CTL (a simplified version)

Syntax

\[ \phi ::= p \mid \neg \phi \mid \phi \lor \phi \mid \text{EX} \phi \mid \text{E}[\phi \cup \psi] \mid \text{A}[\phi \cup \psi] \]

where \( p \) = AP (atomic propositions) OP Clock constraint

Derived Operators

```
\begin{align*}
E[\phi] & \text{ in UPPAAL} \\
A[\phi \cup \psi] & \text{ in UPPAAL}
\end{align*}
```

Question

\( A<> P \) \quad "P will be true for sure in future"

\begin{itemize}
\item \( m \)
\item \( x \leq 5 \)
\item \( p \)
\end{itemize}

?? Does this automaton satisfy \( AF P \)

Note that

\( A<> P \) \quad "P will be true for sure in future"

\begin{itemize}
\item \( m \)
\item \( x \leq 5 \)
\item \( p \)
\end{itemize}

This automaton satisfies \( AF P \)

Note that

\( A<> P \) \quad "P will be true for sure in future"

\begin{itemize}
\item \( m \)
\item \( x \leq 5 \)
\item \( p \)
\end{itemize}

This automaton satisfies \( AF P \)

Algorithm for checking \( A<> P \) \quad Eventually \( P \)

"There is no cycle containing only states where \( p \) is false: not \( E[] (not \ p) \)"

Bouajjani, Tripakis, Yovine '97

On-the-fly symbolic model checking of TCTL
Question: Time bound synthesis

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

Assume $AP$ is satisfied by an automaton A. 
Can we calculate the $\text{Max}$ time bound?
OBS: we know how to calculate the $\text{Min}$ !

An Algorithm (Max)  -- not supported by UPPAAL

\begin{verbatim}
Cost:=0, Pass := {}, Wait := {{l_0,C_0}}
while Wait \# {} do
  select (l,C) from Wait
  if (l,C) = P and Max(C)\geq Cost then Cost:= Max(C)
  else if forall (l,C') in Pass: C \leq C' then
    add (l,C) to Pass
    forall (m,C') such that (l,C) \rightarrow (m,C'):
      add (m,C') to Wait

Return Cost
\end{verbatim}

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

Zone-Widening operation for $\text{Max}$

Inside the UPPAAL tool
Global Reduction
(When to store symbolic state)

No Cycles: Passed list not needed for termination

Inactive (passive) Clock Reduction

Definition
x is only active in location S1

Global Reduction
(When to store symbolic state)

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

To Store Or Not To Store?

Reuse of State Space

A[1] prop1
A[1] prop2
A[1] prop4
- - -
A[1] propn

Search in existing Passed list before continuing search

Which order to search?

Propositions: A[]prop1, A[]prop2, A[]prop3, A[]prop4, A[]prop5

Reusing State Space

States: Passed

51 states
Waiting

Passes

117 states saved
81 states, i.e., 9 states

Time: On less than 10%
(need to re-explore some states)
Reuse of State Space

A[1] prop1
A[1] prop2
A[1] prop4

- 
A[1] propn

Search in existing Passed list before continuing search

Which order to search?

 Hashtable

Swapped to secondary memory

Under-approximation

Bitstate Hashing (Holzman, SPIN)

n,Z'

m,U

Passed

Waiting

REVERSE CREATION ORDER

generation order

Swapped to secondary memory

Bit-state Hashing

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

REPEAT
    - pick (n,Z) in Waiting
    - if for some (m0,U0) in Passed then STOP
      - else add (n,Z) to Passed
      - to Waiting;
    Add (n,Z) to Passed
UNTIL Waiting = Ø

U PPAAL 8 Mbits

Partially

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

Passed(F(n,Z)) := 1
Under Approximation
(good for finding Bugs quickly, debugging)

- **Positive 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)

- **Positive 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