McGill

UPPAAL

Eugene Syriani

Ph.D. Student in the Modelling, Simulation and Design Lab

School of Computer Science

McGill University

OVERVIEW

- In the context
- In Theory: Timed Automata
  - The language: Definitions and Semantics
  - Model Checking and Implementation
- In Practice: UPPAAL
  - Language Extensions
  - Simulation and Verification
- Case Study
- Conclusion on the tool and on the language

McGill

UPPAAL

IN THE CONTEXT

*Upp*sala University (Sweden)

+

*Aal*borg University (Denmark)

==================================

UPPAAL (SweDen)

Paul Petterson

Uppsala

Wang Yi

Uppsala

Kim G. Larsen

Aalborg

IN THE CONTEXT

- First released in 1995
- Power Tool: environment for modelling, simulation and verification of real-time systems
- Types of System: non-deterministic processes with finite control structure and real-valued clocks
- Typical Applications: real-time controllers and communication protocols, where time is critical

IN THE CONTEXT

The Technology

- Efficient model-checker with *on-the-fly* searching technique
- Efficient verification with *symbolic* technique manipulation and solving of constraints
- Facilitate modelling and debugging with automatic generation of *diagnostic traces* explaining the satisfaction of a property
- Visual (graphical) tracing through the simulator

OVERVIEW

- In the context
- In Theory: Timed Automata
  - The language: Definitions and Semantics
  - Model Checking and Implementation
- In Practice: UPPAAL
  - Language Extensions
  - Simulation and Verification
- Case Study
- Conclusion on the tool and on the language

McGill

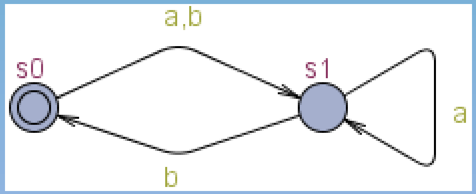
IN THEORY: TIMED AUTOMATA [1]

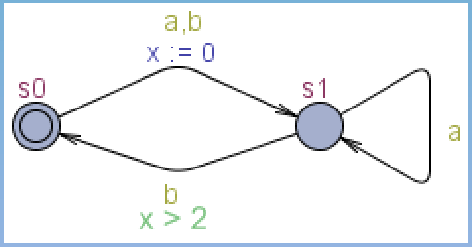
- Theory for modeling and verification of real time systems
- Other formalisms:
  - Timed Petri Nets [5]
  - Timed Process Algebras [6,7,8]
  - Real Time Logics [9,10]
- Model checkers built with timed automata:
  - UPPAAL
  - Kronos [11]

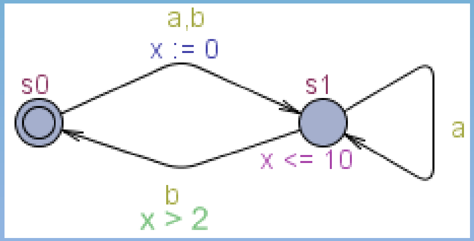
[1] R. Alur and D. L. Dill. A theory of timed automata. *Journal of Theoretical Computer Science, 126(2):183–235, 1994.*

IN THEORY: TIMED AUTOMATA

Evolution







Büchi Automata [2]

- Infinite alphabet
- Initial and accepting states
- Accept execution if pass through accepting state infinitely many times

Büchi Timed Automata

- Büchi-accepting
- Real-valued variables: modelling clock
- Constraints on clock variables and resets

Timed Safety Automata

- Clock variables
- Local invariant conditions
- Accept when invariant is satisfied

typedef TimedSafetyAutomata TimedAutomata

[2] W. Thomas. Automata on infinite objects, in Van Leeuwen, *Handbook of Theoretical Computer Science*, pp. 133-164, Elsevier, 1990.

IN THEORY: TIMED AUTOMATA

Behaviour

- Variables model logical clocks in the system
  - Initialized to 0
  - Increase synchronously at the same rate
- Taking transition (delay or action)
  - Necessary condition: clocks values satisfy guard on edge
  - Action: clocks may be reset to 0

IN THEORY: TIMED AUTOMATA

Formal Definition

A timed automaton is a tuple $\langle L, l_0, \mathrm{E}, \mathrm{I} \rangle$ where:

- L is a finite set of locations
- $l_0 \in L$ is the initial location
- $E \subseteq L \times \mathfrak{B}(C) \times \Sigma \times 2^C \times L$ is the set of edges
- $I: L \rightarrow \mathfrak{B}(C)$ is the function mapping locations to invariants on the clock elements

IN THEORY: TIMED AUTOMATA

Formal Semantics

Operational Semantics of a timed automaton is:

- **If** $u, u + d \in I(l)$ **and** $d \in \mathbb{R}^+$,
  then $\langle l, u \rangle \overset{d}{\rightarrow} \langle l, u + d \rangle$
- **If** $l \xrightarrow{\tau,\alpha,r} l', u \in g, u' = [r \mapsto 0]u$ **and** $u' \in I(l)$,
  then $\langle l, u \rangle \overset{\alpha}{\rightarrow} \langle l', u' \rangle$

- **Notation:** $\langle l, u \rangle$ is a state

  $\langle l, u \rangle \overset{\alpha}{\rightarrow} \langle l', u' \rangle$ is a transition

OVERVIEW

- In the context
- In Theory: Timed Automata
  - The language: Definitions and Semantics
  - Model Checking and Implementation
- In Practice: UPPAAL
  - Language Extensions
  - Simulation and Verification
- Case Study
- Conclusion on the tool and on the language

IN THEORY: TIMED AUTOMATA

Model Checking

- Reachability analysis:
  - Safety: “something bad never happens”
  - Liveness: “something good will eventually happen”
    - loop detection

IN THEORY: TIMED AUTOMATA

Model Checking

- The state space of a timed model can be represented by a *zone graph* (efficient region graph)
- A zone is the maximal set of clock assignment solution of clock constraints
- Zone graphs can be infinite: *widening operation*
- Zone graphs can be normalized to a canonical representation

McGill

IN THEORY: TIMED AUTOMATA

Model Checking and Implementations

- Zones can be efficiently represented in memory as *Difference Bound Matrices* (DBM) [3]
- DBM store clock constraints in canonical form
- Clock $g \in \mathfrak{B}(C)$ constraint is

  $g ::= x \sim m | x - y \sim n | g \wedge g$

  where $x, y \in C, m, n \in \mathbb{N}$ and $\sim \in \{\leq, <, =, >, \geq\}$

[3] J. Bengtsson and W. Yi . Timed Automata: Semantics, Algorithms and Tools. In *Lecture Notes on Concurrency and Petri Nets*. W. Reisig and G. Rozenberg (eds.), LNCS 3098, Springer-Verlag, 2004.

IN THEORY: TIMED AUTOMATA

Model Checking and Implementations

- DBM will represent any clock constraint of a zone as:
  - If $x_i - x_j \sim n \in D$, then $D_{ij} = (\sim, n)$
  - If $x_i - x_j$ is unbounded, then $D_{ij} = \infty$
  - Add $D_{ii} = (\leq, 0)$ and $D_{0i} = (\leq, 0)$

McGill

IN THEORY: TIMED AUTOMATA

Model Checking and Implementations

$$D = x - 0 < 20 \wedge y - 0 \leq 20 \wedge y - x \leq 10$$
$$\wedge\, x - y \leq -10 \wedge 0 - z < 5$$

$$M(D) = \begin{pmatrix} (0, \leq) & (0, \leq) & (0, \leq) & (5, <) \\ (20, \leq) & (0, \leq) & (-10, \leq) & \infty \\ (20, \leq) & (10, \leq) & (0, \leq) & \infty \\ \infty & \infty & \infty & (0, \leq) \end{pmatrix}$$

IN THEORY: TIMED AUTOMATA

Model Checking and Implementations

- Operations on DBMs:

  1. $consistent(D)$: checks if a DBM is consistent, a non-empty solution set. Used for removing inconsistent states from an exploration (negative cycles).
  2. $relation(D, D')$: checks if $D \subseteq D'$. Used for combined inclusion checking.
  3. $satisfied(D, x_i - x_j \leq m)$: checks if a zone satisfies a certain condition.
  4. $up(D)$: computes the strongest post-condition of a zone.
  5. $down(D)$: computes the weakest pre-condition of a zone.
  6. $and(D, x_i - x_j \leq m)$: add a constraint to a zone.
  7. $free(D, x)$: remove all conditions on a clock in a zone.
  8. $reset(D, x \coloneqq m)$: set the clock to a specific value.
  9. $copy(D, x \coloneqq y)$: copy the value of one clock into another.
  10. $shift(D, x \coloneqq x + m)$: add or subtract a clock with an integer value.

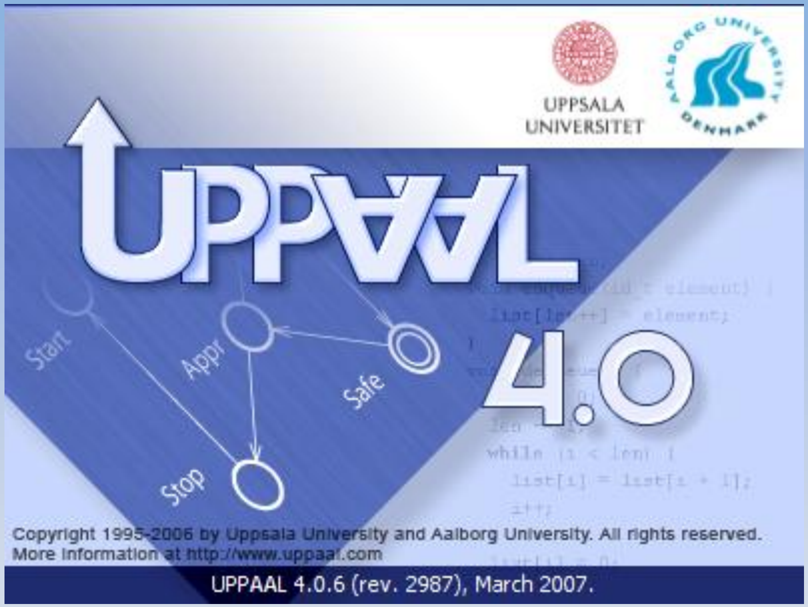
McGill

OVERVIEW

- In the context
- In Theory: Timed Automata
  - The language: Definitions and Semantics
  - Model Checking and Implementation
- In Practice: UPPAAL
  - Language Extensions
  - Simulation and Verification
- Case Study
- Conclusion on the tool and on the language

IN PRACTICE: UPPAAL

UPPAAL, The Tool [4,5]



[4] G. Behrmannet al. Uppaal Implementation Secrets. In *Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems*, 2002.
[5] G. Behrmann, A. David, and K. G. Larsen. A Tutorial on Uppaal. *In proceedings of the 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems*. LNCS 3185.

IN PRACTICE: UPPAAL

Language Extensions

- Typed variables:
  - Integer
  - Clock
  - Channel
  - Constant
  - Scalar (set)
  - Array
  - Meta-variable
  - Record variable: structure

IN PRACTICE: UPPAAL

Language Extensions: A C syntax

- **Functions (typed and untyped)**
- **For/While/Do loops, If-Else statements**
- **Operators**
  - **All** C **operators: comparison, mathematical, assignment**
  - **Wrapper operators: min, max,** and**,** or**,** not**,** imply
  - **Quantifier:** forall**,** exists

IN PRACTICE: UPPAAL

Language Extensions

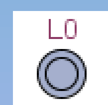
- Template: extended time automaton
  - Locations (extended)
  - Edges (extended)
  - Declarations
  - Parameters

IN PRACTICE: UPPAAL

Location

- Invariant

- Initial



- Urgent

  - Atomic: freeze time

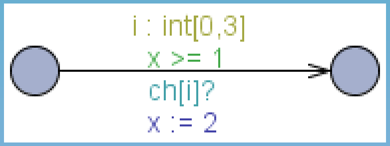
- Committed

  - Urgent + Highest priority

McGill

IN PRACTICE: UPPAAL

Edge



- Guard
  - Edge is enabled iff its guard is true
- Update
  - Assignment
  - State of the system changed only on transition execution

- Synchronization
  - Over channel with the same name
- Selection
  - Non-deterministic binding of variable over a range

McGill

IN PRACTICE: UPPAAL

Synchronization

- Edge labelled *ch!* (emitter) synchronizes with edge labelled *ch?* (receiver)
- Binary: pair of channels chosen non-deterministically
- Broadcast: emitter channel synchs with all receiver channels. Not blocking
- Urgent: no delay, no time constraint

McGill

IN PRACTICE: UPPAAL

System Description

- Global and local declarations
  - Variables, functions and types
- Automata templates
  - Parameterizable extended timed automata
    ≡ Behavioural classes
- System definition
  - System model: concurrent processes, channels and local and global variables

IN PRACTICE: UPPAAL

Synchronization revisited

- Concurrent processes synchronize via channels (*ch!* and *ch?*)
- CCS parallel composition:
  - Action interleaving
  - Hand-shake synchronization
- Computationally extremely expensive (product automaton): *on-the-fly* verification

IN PRACTICE: UPPAAL

More language extensions

- Parameterized templates
- Operations on processes (re-use)
- Priorities
  - Channels
  - Processes
- Graphical and textual syntax for automata
- More…

OVERVIEW

- In the context
- In Theory: Timed Automata
  - The language: Definitions and Semantics
  - Model Checking and Implementation
- In Practice: UPPAAL
  - Language Extensions
  - Simulation and Verification
- Small Case Study
- Conclusion on the tool and on the language

IN PRACTICE: UPPAAL

Verification

- A model checker verifies whether a model respects a requirement
- UPPAAL uses a simplified version of CTL [5] (temporal first-order logic)
- State formulae
- Path formulae: reachability, safety, liveness

[5] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Systems*, 8(2):244–263, April, 1986.

IN PRACTICE: UPPAAL

Verification

- **State formula**
  - **Complex boolean expression, similar to guards but disjunction is allowed**
  - deadlock**: no action transition going out of a state or of its delay successors**

IN PRACTICE: UPPAAL

Verification

- Reachability property
  - Sanity check: “something will possibly happen”
    Does not mean it will !
  - $E <> \ \varphi$: there is a path that, starting from an initial state, reaches a state where φ is eventually satisfied

IN PRACTICE: UPPAAL

Verification

- Safety property
  - Invariantly check: “something bad will never happen”
  - $A[\,]\ \varphi$: φ should be true for all reachable states
  - $E[\,]\ \varphi$: there is a maximal path along which φ is always true (the last state is infinite or a leaf)

IN PRACTICE: UPPAAL

Verification

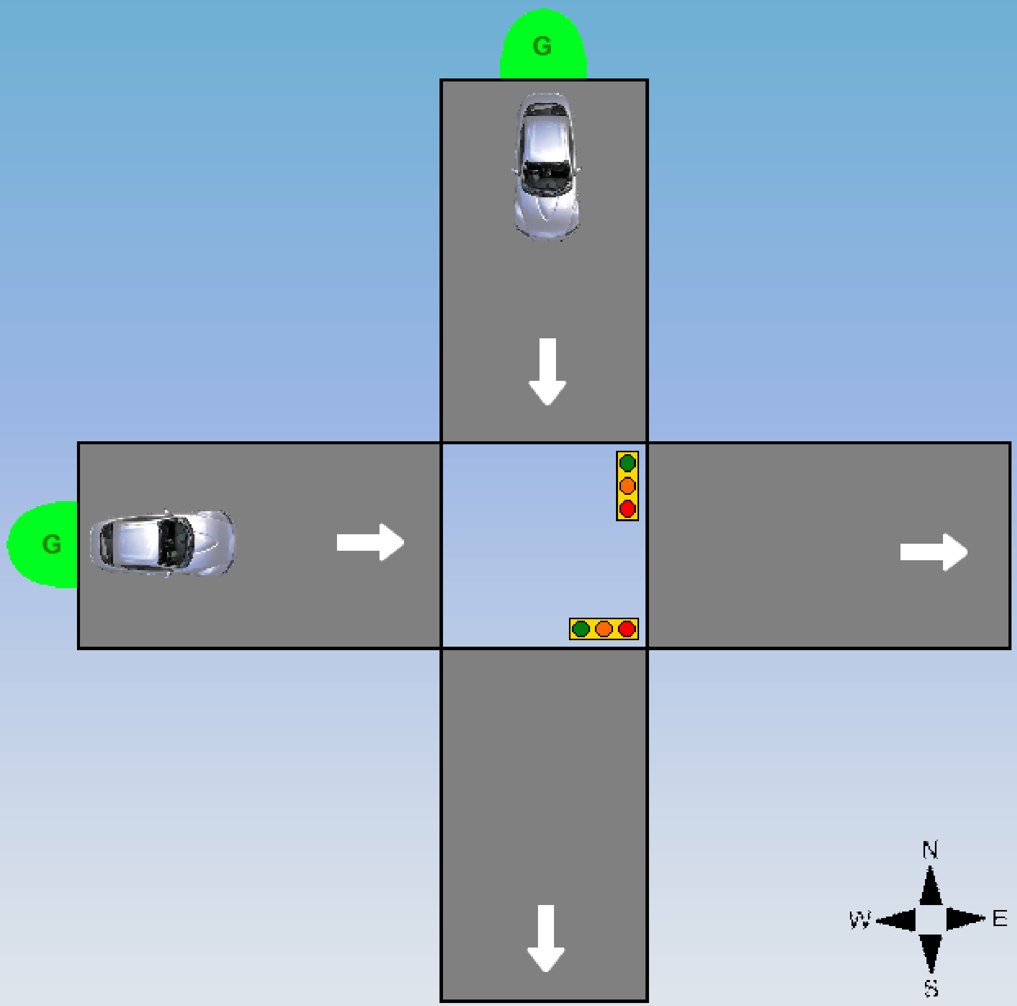
- **Liveness property**
  - "something will eventually happen"
  - $A <> \varphi$: all transitions eventually reach a state where φ is true
  - $\varphi \dashrightarrow \psi$: whenever φ is satisfied, ψ will eventually be satisfied

OVERVIEW

- In the context
- In Theory: Timed Automata
  - The language: Definitions and Semantics
  - Model Checking and Implementation
- In Practice: UPPAAL
  - Language Extensions
  - Simulation and Verification
- Case Study
- Conclusion on the tool and on the language

UPPAAL

CASE STUDY



CASE STUDY

- Close to DEVS assignment
- Automaton (statechart-like) version
- More analysis than with Petri-Nets

CASE STUDY

Usage

1. Graphical Model Edition
2. Graphical Simulation with recording of dynamic behaviour
3. Interface for Requirement Specification
4. Model-Checking of safety and liveness
    a. Graphical trace debugging

OVERVIEW

- In the context
- In Theory: Timed Automata
  - The language: Definitions and Semantics
  - Model Checking and Implementation
- In Practice: UPPAAL
  - Language Extensions
  - Simulation and Verification
- Case Study
- Conclusion on the tool and on the language

CONCLUSION ON THE TOOL

- UPPAAL simulator is a process algebra tool
  - Process behaviour defined by a timed automaton
  - Allow process synchronization
- UPPAAL verifier is a model checker
  - Models can be queried for safety and liveness properties
- UPPAAL is an editor for real-time models
  - Visual traces for debugging

CONCLUSION ON THE TOOL

- Cost-UPPAAL
  - Minimal cost reachability analysis
- Distributed-UPPAAL
  - Run on multi-processors and clusters
- T-UPPAAL
  - Test case generator for black box conformance testing
- World-wide used
  - Sweden, Denmark, Belgium, England, Germany, USA

CONCLUSION ON THE LANGUAGE

- Template $\hat{=}$ *composite state* in Statechart but more scalable with system description
- System $\hat{=}$ group of *orthogonal components* with synchronisation possibility
- Process-Oriented *¿Kiltera?*

  Processes and channels
- Super-porcess? process composition
- Inheritance?

MORE REFERENCES

6. B. Berthomieu and M. Diaz. Modeling and verification of timed dependent systems using timed petri nets. *IEEE Transactions on Software Engineering,* 17(3):259–273, 1991.

7. G. M. Reed and A. W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science, 58(1-3):249–261, 1988.*

8. W. Yi. CCS + time = an interleaving model for real time systems. In *Proceedings, 18th Intl' Colloquium on Automata, Languages and Programming, LNCS, 510. Springer-Verlag, 1991.*

9. X. Nicollin and J. Sifakis. The algebra of timed processes, ATP: Theory and application. *Journal of Information and Computation, 114(1):131–178, 1994.*

10. Z. Chaochen. Duration calculus, a logical approach to real-time systems. *LNCS, 1548:1–7, 1999.*

11. R. Alur and T. A. Henzinger. A really temporal logic. *Journal of the ACM,* 41(1):181–204, 1994.

12. S. Yovine. Kronos: a verification tool for real-time systems. *Journal on Software Tools for Technology Transfer, 1, October 1997.*

UPPAAL website: http://www.it.uu.se/research/group/darts/uppaal/documentation.shtml

UPPAAL's help manual