# PROJECT REPORT

## ESE601
## SPRING - 2006


## AKSHAY H. RAJHANS
## NIKHIL K. SHALIGRAM

**Abstract**:

The aim of this report is to document the term project done towards the completion of ESE 601 Hybrid Systems course. The goal of the term project was to learn UPPAAL tool and apply it in a practical example.

The report is organized as follows:
Section 1 introduces the concepts of hybrid automata, linear hybrid automata and timed automata. Section 2 provides an overview of UPPAAL, its structure, architecture, and simulation and verification using UPPAAL. Section 3 talks about the application of UPPAAL in a practical example, and Section 4 concludes the report with a summary.

1. **Introduction**:

- **Definition of terms used**:
    - **Hybrid Systems:**

Very generally speaking, hybrid systems can be defined as the systems which have a combination of continuous dynamics and discrete events. These continuous and discrete dynamics not only coexist, but interact. Changes occur both in response to discrete, instantaneous events as described by difference equations and in response to dynamics as described by differential euqations.

- **Hybrid Automata:**

Hybrid systems can be best expressed as hybrid automata. A hybrid automaton is described by a septuple (L, X, A, W, E, Inv, Act) where the symbols have the following meaning:

- L is a finite set, called the set of discrete states or locations. They are the vertices of a graph
- X is the continuous state space of of the hybrid automaton in which the continuous state variables 'x' take their values. For our purposes $X \in R^n$ or X is an n-dimensional manifold.
- A is a finite set of symbols whixh serve to label the edges.
- $W = R^n$ is a continuous communication space in which the continuous external variables 'w' take their values.
- E is a finite set of edges called transitions (or events). Every edge is defined by a five-tuple (l, a, $Guard_{ll}$', $Jump_{ll}$',l'), where $l, l' \in L$, $a \in A$, $Guard_{ll}$' is a subset of X and $Jump_{ll}$' is a relation defined by a subset of $X \times X$. The transition from the discete state l to l'is enabled when the continuous state 'x' is in $Guard_{ll}$', while during the transition the continuous state 'x' jumps to a value x' given by the relation $(x,x') \in Jump_{ll}$'.
- Inv is mapping from the locations L to the set of subsets of X, that is $Inv(l) \in X$ for all $l \in L$. Whenever the system is at location l, the continuous state x must satisfy $x \in Inv(l)$. The subset Inv(l) for $l \in L$ is called location invariant of location l.
- Act is a mapping that assigns each location $l \in L$ a set of differential algebraic equations $F_l$, relating the continuous state variables x with their time derivatives $\dot{x}$ and the continuous external variables w: $F_l(x, \dot{x}, w) = 0$

The solutions to these differential algebraic equations are called the activities of the location. This definition of hybrid automata could be found in [R6].

- **Linear Hybrid Automata:**

Linear hybrid automata could be defined as a special class of hybrid automata, where the continuous part is linear, i.e. has a constant slope.

- **Timed Automata:**

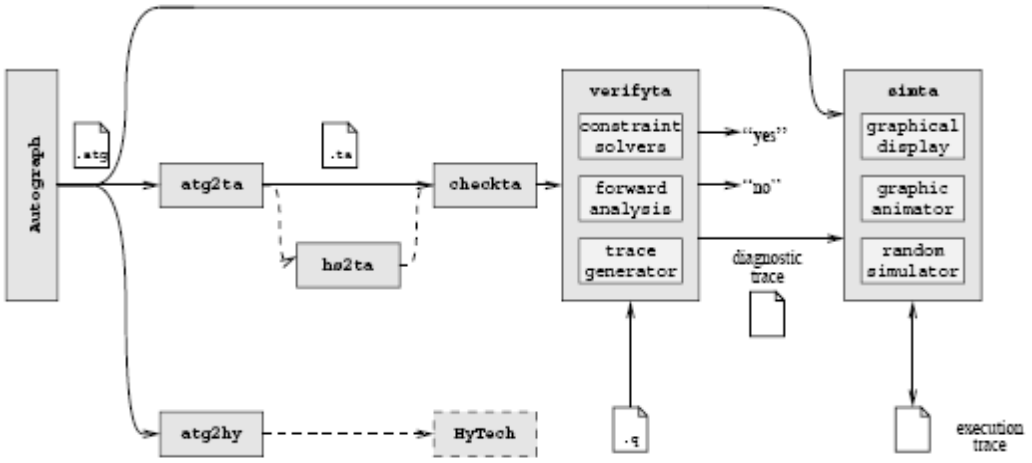
The term 'timed automata' was introduced by Rajeev Alur and David Dill in 1990. The formal definition could be found in [R8]. The basic idea is that the notion of time can be introduced via the continuous variables called clocks, which have a constant slope equal to 1, i.e. increase monotonically, and can be reset (or set) after each discrete transitions. Clock constraints *i.e.* guards on edges are used to restrict the behavior of the automata. A transition represented by an edge can be taken when the clocks values satisfy the guard labeled on the edge. Clocks may be reset to zero when a transition is taken.

2. **UPPAAL:**

- **An overview of UPPAAL**:

UPPAAL is an integrated tool for environment for modeling, simulation and verification of real time systems. The tool developed jointly by Uppsala University in Sweden and Aalborg University in Denmark, derives its name coined from an acronym containing the first three letters of the two developer universities. UPPAAL finds its applications in the field of real time systems, and is of particular help in case of timed automata.

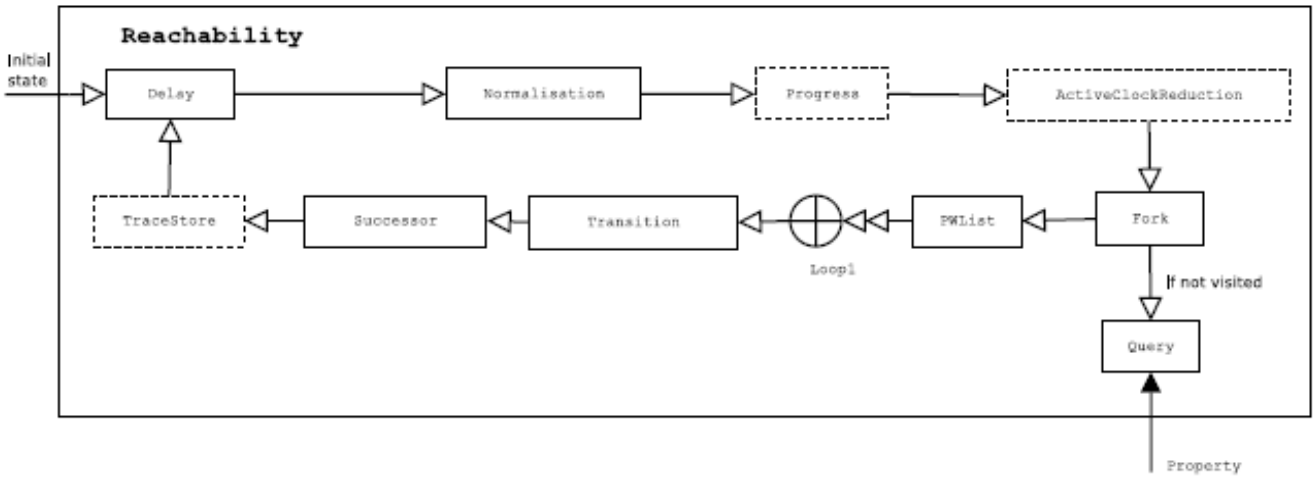
- **Architecture of UPPAAL:**




**UPPAAL Architecture**

UPPAAL consists of three main parts: a **description language**, a **simulator** and a **model-checker**. The description language is a command language, which serves as a modeling or design language to describe system behavior. The simulator is a validation tool which enables examination of possible dynamic executions of a system during early design or modeling stages. The model checker is useful in doing the reachability analysis, with the given set of constraints.

To provide a system that is both efficient, easy to use and portable, UPPAAL is split into two components, a graphical user interface written in Java and a verification engine written in C++. The engine and the GUI communicate using a protocol, allowing the verification to be performed either on the local workstation or on a powerful server in a network.
The verification engine for analysing the reachability of the system is structured as a pipeline that incarnates the natural dataflow in the algorithm. This architecture simplifies both activating and deactivating optimizations at runtime by inserting and removing stages dynamically, and introducing new optimizations and features in the tool by implementing new or changing existing stages. The pictorial view of the same is as given below:




**Pictorial view of UPPAAL's reachability engine**

- **Modeling with UPPAAL:**

The core of the UPPAAL modeling language is networks of timed automata. A typical UPPAAL model consists of a set of timed automata, a set of clocks, global variables, and synchronizing channels. A node in an automaton may be associated with an invariant, which is a set of clock constraints, for enforcing transitions out of the node. An arc may be associated with a guard for controlling when this transition can be taken. On any transition, local clocks may get reset and global variables may get re-assigned. A trace in UPPAAL is a sequence of states, each of which containing a complete specification of a node from each automata, such that each state is the result of a valid transition from the previous state.

Apart from the above basics derived from the definition of timed automata, the UPPAAL language has been further extended with features to ease the modeling task and to guide the verifier in state space exploration. UPPAAL has been proven to be a useful model checking tool for many domains including distributed multimedia and power controller applications. The most important features are shared integer variables, urgent channels and committed locations.

- **Networks of Timed Automata:**

A network of timed automata is the parallel composition of a set of timed automata called processes, combined into a single system by the CCS parallel composition operator with all external actions hidden. Synchronous communication between the processes is by hand-shake synchronization using input and output actions; asynchronous communication is by shared variables as described later. To model hand-shake synchronization, the action alphabet is assumed to consist of symbols for input actions denoted, output actions denoted and internal actions represented by the distinct symbol.

The semantics of networks is given as for single timed automata in terms of transition systems. A state of a network is a pair <l,u> where 'l' denotes a vector of current locations of the network, one for each process, and 'u' is the clock assignment remembering the current values of the clocks in the system. A network may perform two types of transitions, delay transitions and discrete

transitions. The rule for delay transitions is similar to the case of single timed automaton where the invariant of a location vector is the conjunction of the location invariants of the processes. There are two rules for discrete transitions defining (a) **local actions** where one of the processes makes a move on its own, and (b) **synchronizing actions** where two processes synchronize on a channel and move simultaneously.

- **Synchronization of the networked automata:**

For the purpose of synchronization, the transition labels need to be assigned as 'channels' in UPPAAL. There could be a potential problem with variable updating in a synchronizing transition where one of theprocesses participating in the transition updates a variable used by the other. In UPPAAL, however, for a synchronization transition, the resets on the edge with an output-label is performed before the resets on the edge with an input-label. The disctinction of the input-labels and the output-labels is introduced by adding a '?' or a '!' at the end of the label. This destroys the symmetry of input and output actions. But it gives a natural and clear semantics for variable updating. The transition rule for synchronization is modified accordingly:

$\langle l, u\rangle \xrightarrow{\tau} \langle l[l_i'/l_i][l_j'/l_j], u'\rangle$ if there exist $i \neq j$ such that

1. $l_i \xrightarrow{g_i, a?, r_i} l_i'$, $l_j \xrightarrow{g_j, a!, r_j} l_j'$ and $u \in g_i \wedge g_j$, and
2. $u' = [r_i \mapsto 0]([r_j \mapsto 0]u)$ and $u' \in I(l[l_i'/l_i][l_j'/l_j])$

- **Urgent Channels:**

To support the urgent transitions, which should be taken as soon as they are enabled, UPPAAL supports a notion of urgent channels. An urgent channel works much like an ordinary channel, but with the exception that if a synchronization on an urgent channel is possible the system may not delay. Interleaving with other enabled action transitions, however, is still allowed. In order to keep clock constraints representable using convex zones, clock guards are not allowed on edges synchronizing on urgent channels.

In such situations, system may go into no-convex zone which is not desired. The problem can be solved by splitting the non-convex zone into two convex ones. But in general, the splitting is a computationally vary complex operation. Hence, in UPPAAL it is decided to avoid such operations for the sake of efficiency. So only integerguards are allowed on edges involving synchronizations on urgent channels.

Similar to the Urgent Channels being a special class of channels, there are two special classes of locations that can be defined in UPPAAL.
i. Urgent Locations and ii. Committed Locations

- **Urgent Locations:**

Urgent locations are a locations where time is not allowed to increment. Semantically, urgent locations are equivalent to:

- Adding an extra clock x which is reset on every incoming egde
- Adding an invariant x<=0 to the location

- **Committed Locations:**

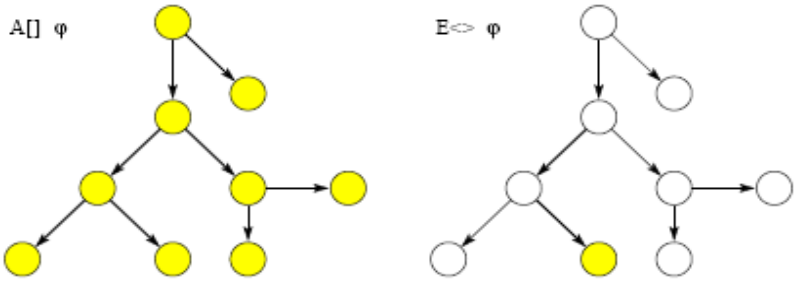
A committed location is a stricted notion than an urgent location. In a committed location, no delay is allowed and in the given network, if any process is in a committed location then only action transitions starting from such a committed location are allowed. Thus, processes in committed locations may be interleaved only with processes in a committed location. Commited locations are useful for creating atomic sequences and for encoding synchronization between two or more components, e.g. in atomic broadcast or multicast.

- **Verification in UPPAAL:**

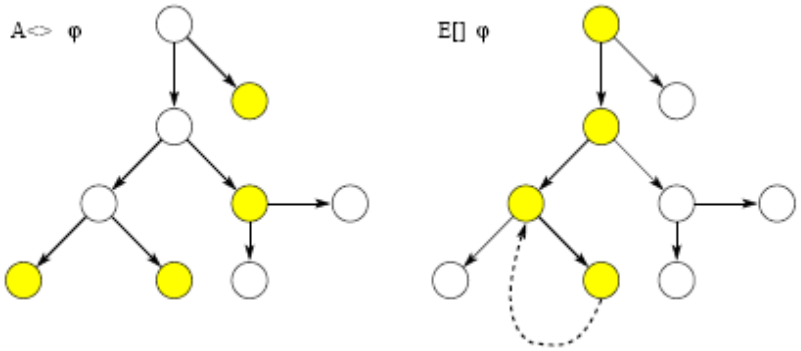
The verifier in UPPAAL is equipped to check the reachability, safety and liveness properties of the system, by on-the-fly exploration of the state space of the system in terms of symbolic states represented by constraints. It also provides a requirement specification editor for specifying and documenting the system requirements.

Verification in UPPAAL is also refered to as model checking. The model checking engine of UPPAAL is designed to check a subset of TCTL formula for networks of timed automata. The formulae for verification are listed as follows as an example:

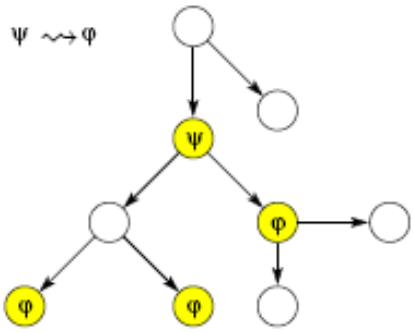
- A [ ] ϕ = invariantly or always ϕ
- E <> ϕ = possibly or eventually ϕ




- A <> ϕ = always eventually ϕ
- E [ ] ϕ = eventually always ϕ




- ϕ —> ψ = ϕ always leads to ψ




Here ϕ and ψ are any local properties that can be checked locally on a state, such as boolean expressions over predicates on locations and integer variables, and clock constraints.

The two types of properties most commonly used in verification of timed systems are E<>ϕ and A[ ] ϕ. They are dual in the sense that E<>ϕ is satisfied if and only if A[]¬ϕ is not satisfied. These properties are often termed as safety properties, i.e. the system is safe in the sense that a specified hazard can not occur. It is also possible to transform so called bounded liveness properties, i.e. properties stating that some desired state will be reached within a given time, into safety properties using observer automata or by annotating the model.

The other three types of properties are commonly classified as unbounded liveness properties, i.e. they are used to express and check for global progress. These properties are not commonly used in UPPAAL case-studies. It seems to be the case that bounded liveness properties are more important for timed systems.

- **Other algorithms deployed by UPPAAL:**

    - Minimal constraint systems and CCDs to reduce memory consumption by removing redundant information in zones before storing them.
    - Selective storing of states in 'Passed', where static analysis is used to detect states that can be omitted safely from 'Passed' without losing terminaion.
    - Compression and sharing of state data, to reduce memory consumption of 'Passed' and 'Wait'
    - Active clock reduction, to determine when the value of clock is irrelevent. This reduces the size of individual states as well as the perceived state space.
    - Supertrace and hash compaction where already visited states are stored only as hash signatures and convex hull approximation, where convex hulls are used to approximate unions of zones, for reducing memory consumption at a risk of inconclusive results.

- **Applications of UPPAAL:**

UPPAAL finds its applications in all real time practical problems. Since we can formulate those systems as hybrid finite timed automata, all the system which involve time can be modeled in UPPAAL. Some of such real time systems could be :

  - Gearbox controller in automotive vehicles
  - Railway gate control
  - Automatic cruise control and Anti-lock braking system in automobiles
  - Verification in robotics
  - Reachability analysis in robotics
  - In systems which require optimization in switching modes such as one explained by Jim and Sameer
  - Airbag control in automobiles
  - Collision avoidance protocol
  - Multimedia streaming etc.

- **Advantages and disadvantages UPPAAL:**

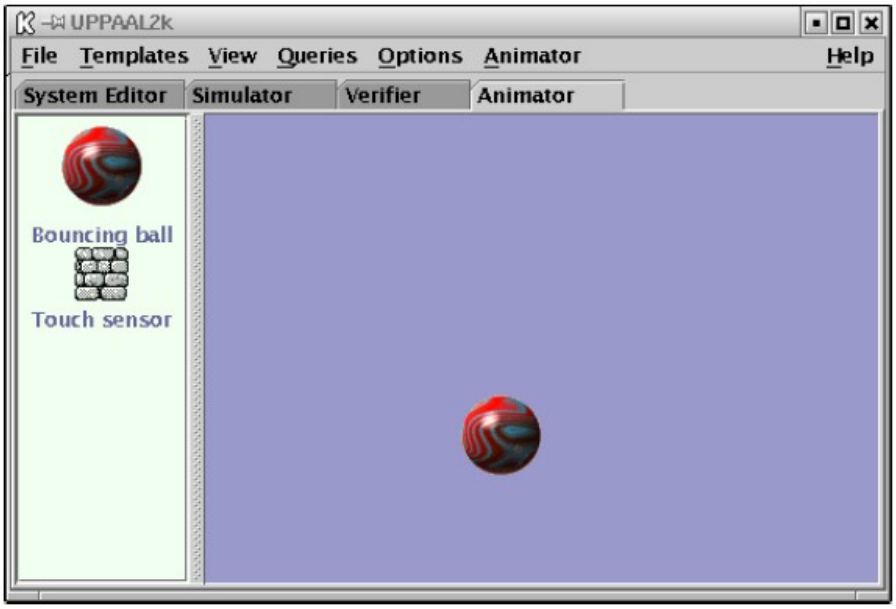
  - As we can see from its syntax, UPPAAL model language is small, simple and remarkably easy to use. The graphical interface of the UPPAAL toolbox also puts its user-friendliness years ahead of many other formal method research tools.
  - For non real-time systems, data variable constraints and temporal logic properties can usually represent most desired properties. However, for real-time systems, relaxed temporal properties like temporal logic are insufficient as precise timing constraints are needed for their correctness. Being a formal method designed for real-time systems, UPPAAL's inclusion of timing information in the modeling and the verification languages is preferable considering that many other formal methods do not provide this feature.
  - Taking into consideration its negative side, it can not open two projects simultaneously in the same window, i.e. We have to close one of the projects and reopen it later when we are finished with the other project.

- Location dependent clock integrators and parameters are not employed

- **Future developments of UPPAAL:**

The future developments as listed by the developers on the UPPAAL website are as follows:

- COUPPAAL : Involves cost optimal search
- PARAMETRIC UPPAAL : Finding solutions to parameterized reachability problems
- STOPWATCH UPPAAL : From modeling of timed automata to hybrid systems
- PR UPPAAL : Probabilistic timed automata
- HUPPAAL : Supports hierarchical structures for modeling
- EX UPPAAL : Executable timed automata
- Hybrid automata animation

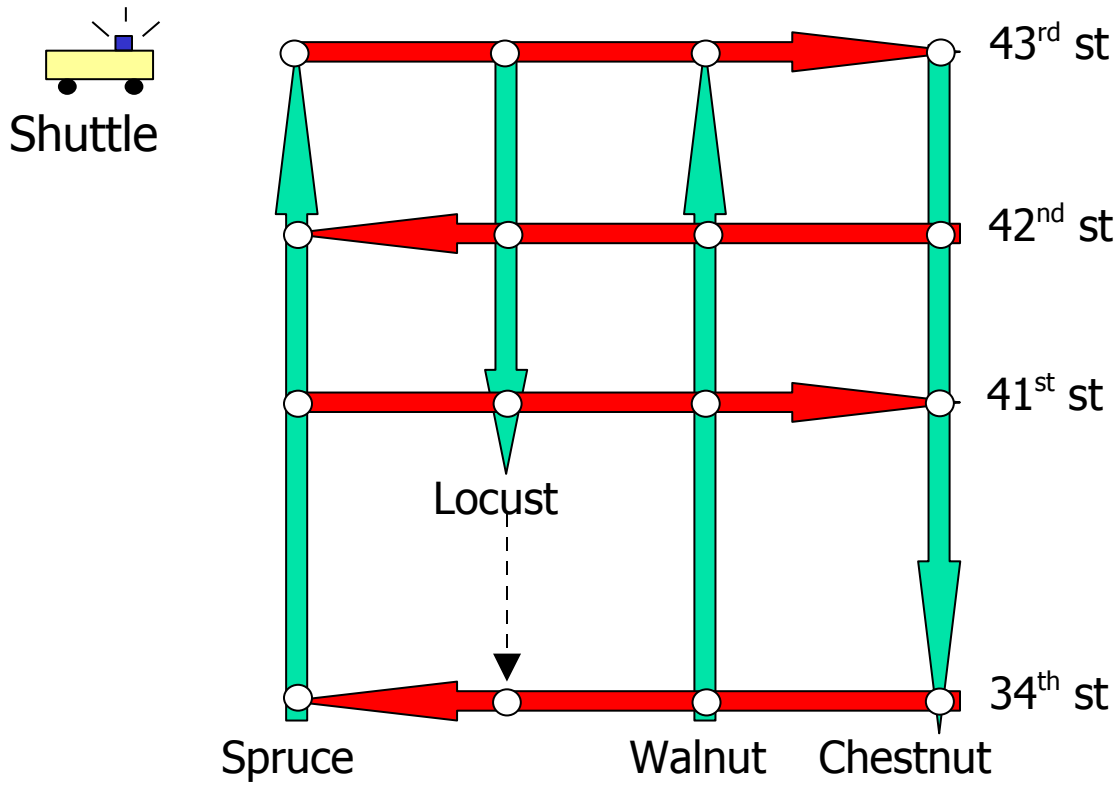



**Bouncing ball example as would be shown in hybrid automata animation**

3. **Implementation of UPPAAL in a practical problem:**

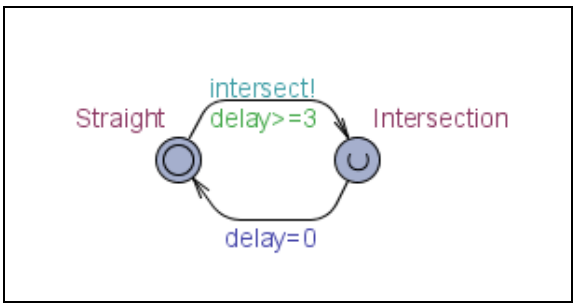
To implement UPPAAL in a practical problem, a practical example was assumed. This example consisted of a Penn shuttle traveling through the streets of West Philadelphia as a timed automata, with each intersection of any two roads being considered as the discrete locations. The East West running streets that were considered were Spruce street, Locust Street, Walnut street and Chestnut street, with stricter direction constraints than actual. The North- South streets that were considered, were 34th street, from where the shuttle leaves, and 41st, 42nd and 43rd streets, where the destination locations in West Philadelphia could be located.

A schematic of the problem space could be shown as follows:

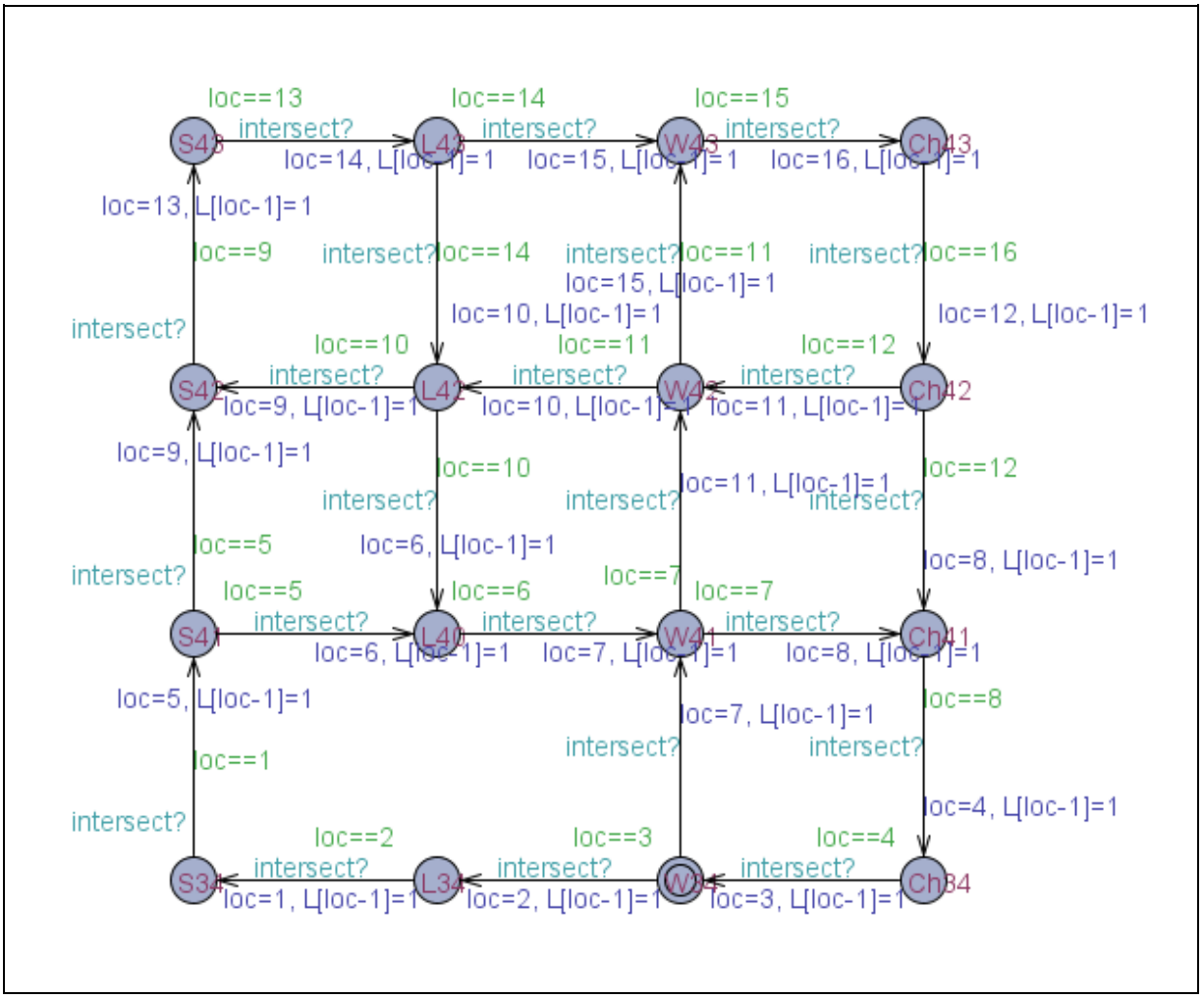


Tramslating into the UPPAAL language, the shuttle was modeled as a template '`drive`', while the area of West Philadelphia under consideration was modeled as a template '`city`'.

The two templates were as shown below:




'`drive`' **template**




'`city`' **template**

- **Findings and observations**:
  - Reachability, safety and liveness properties were verified
  - Urgent location feature was used
  - Diagnostic trace was used
  - Deadlock checking feature was used
  - A[], E<>, --> types of CTL formulae were used while verification
  - A basis was created for further use

## 4. Conclusions:

In this project we studied UPPAAL as a programming software for designing, simulating and verifying a hybrid system expressed as timed hybrid automata. We were also able to use some of its tools/key features to simulate a Penn shuttle routing problem as a timed automata.

While using UPPAAL, we found that, it is an easy to use formal method that is suitable for modeling systems with real-time constraints. Concurrent systems that are not data-dependent are bound to find UPPAAL appealing. Intuitive tools and modeling language make UPPAL ideal for beginners in the formal method realm.

---

**References:**

**[R1] - UPPAAL in a Nutshell**
Kim G. Larsen1, Paul Pettersson2, and Wang Yi2
1. Department of Computer Science and Mathematics, Aalborg University, Denmark.
Email: kgl@cs.auc.dk
2. Department of Computer Systems, Uppsala University, Sweden.
Email: fpaupet,yig@docs.uu.se

**[R2] - A Tutorial on Uppaal** (Updated 17th November 2004)
Gerd Behrmann, Alexandre David, and Kim G. Larsen
Department of Computer Science, Aalborg University, Denmark

{behrmann,adavid,kgl}@cs.auc.dk

**[R3] - UPPAAL – Now, Next, and Future**
Tobias Amnell, Gerd Behrmann, Johan Bengtsson, Pedro R. D'Argenio, Alexandre David, Ansgar Fehnker, Thomas Hune, Bertrand Jeannet, Kim G. Larsen, M. Oliver Möller, Paul Petterson, Carsten Weise, Wang Yi
www.uppaal.com

**[R4] - UPPAAL: Status & Developments**
Kim G Larsen, Paul Pettersson, Wang Yi
www.uppaal.com

**[R5] - Timed Automata: Semantics, Algorithms and Tools**
Johan Bengtsson and Wang Yi
Uppsala University
Email: {johanb,yi}@it.uu.se

**[R6] - An Introduction to Hybrid Dynamical Systems**
Arjan van der Schaft, Hans Schumacher
ESE 601 Reference Reading papers

**[R7] - UPPAAL2K : Small Tutorial**
www.uppaal.com

**[R8] - A Theory of Timed Automata**
Rajeev Alur, David Dill, 1990

**[R9] - UPPAAL introduction**
Alexandre David, Paul Petterson
Real Time Systems Symposium (RTSS) ’05

**[R10] - Beyond UPPAAL**
Alexandre David, Paul Petterson
Real Time Systems Symposium (RTSS) ’05