The Uppaal Tool

Formal Methods for Concurrent and Real-Time Systems, A.Y. 2024/2025

Andrea Manini, Pierluigi San Pietro

9th April 2025

Introduction

- Integrated tool environment for modeling, simulation and verification of **Timed Automata** (TA).

- Developed jointly by Uppsala University (UPP) and Aalborg University (AAL).

- Uppaal website

  Download latest academic version.

  Available for Windows, Linux and MacOS.

  Installation instructions are available on the Download page.

Uppaal Models: Templates

- Uppaal allows to build models as **Networks of Timed Automata** (NTA).
- Users can model the **Template** of a TA, and define the system as a composition of Template **instances**.
- A Template can have formal **parameters** whose value is specified when an instance is created:
  - e.g., assume you have a template `Car` with parameter `int vel`, the creation of an instance (a "process" in Uppaal) will look like the following:
    ```
    myCar = Car(50);
    ```
- For each Template, it is necessary to model locations and edges.

Uppaal Models: Locations

- For each Template, there must be one **initial** location.
- Locations can be endowed with **invariants**, i.e., logic conditions that must be verified as long as the process is in that location.
- Locations can be **urgent** or **committed**. Time cannot pass in urgent or committed locations: it is like resetting a clock *y* upon entering them, and endowing them with the invariant $y \leq 0$.

Uppaal Models: Edges

- Edges can have **guard** conditions, i.e., logic conditions that must evaluate to true for the edge to be enabled (otherwise, it cannot fire).

- Synchronization among TA is achieved through channels. Edges can be labelled with:
  - *c!*: if the automaton **sends** a message through channel c;
  - *c?*: if the automaton **receives** a message through channel c.

- Edges can also be endowed with **update instructions** (assignment instructions or function calls) that get executed when the edge fires.

Case Study: Coffee Machine

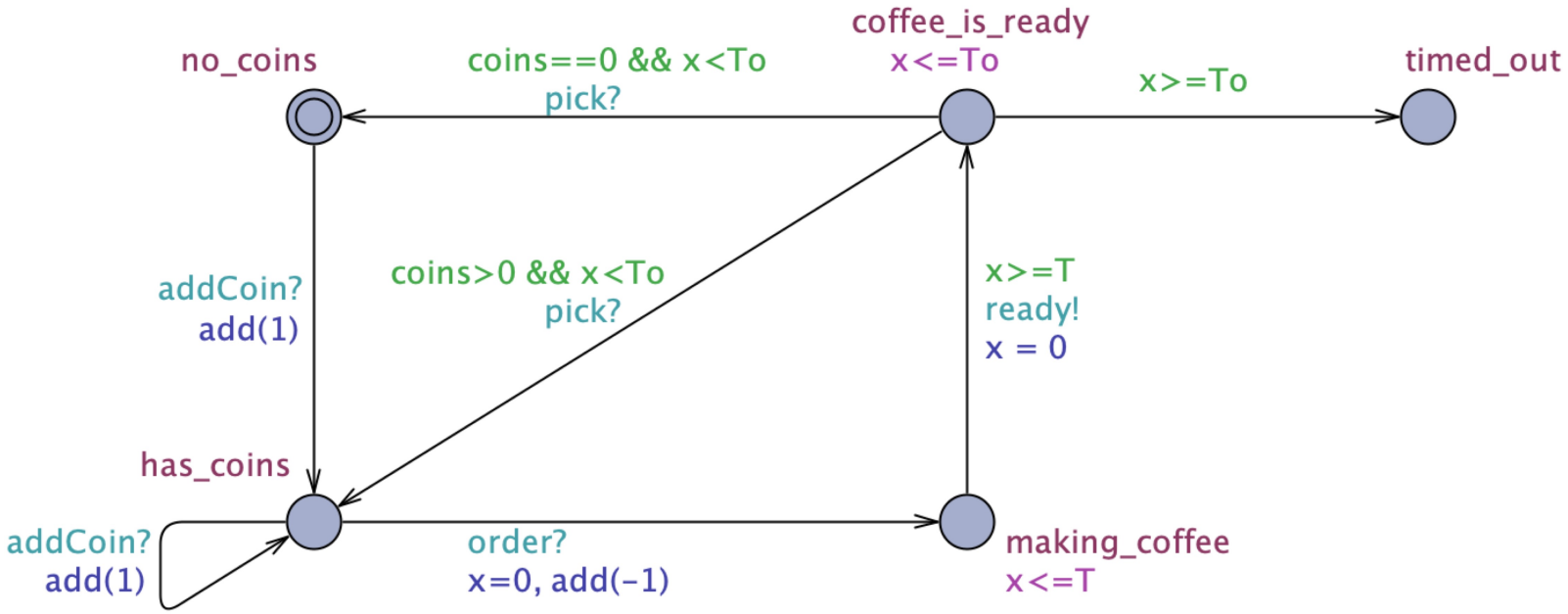
We want to model the behavior of a coffee machine and a user. The user can either insert coins or order a coffee, if the machine credit is ≥ 1. Assume the machine starts with 0 coins and ordering a coffee consumes 1 coin. The machine takes T time instants to brew a coffee. As soon as the coffee is ready , the user can pick it up . If the coffee is not picked up within T_0 time instants, the machine will time out.

The Uppaal model should include:

- The machine template, with parameters T and T_0;
- The user template;
- A system with instances *m*: $T = 3$, $T_0 = 10$ and u.

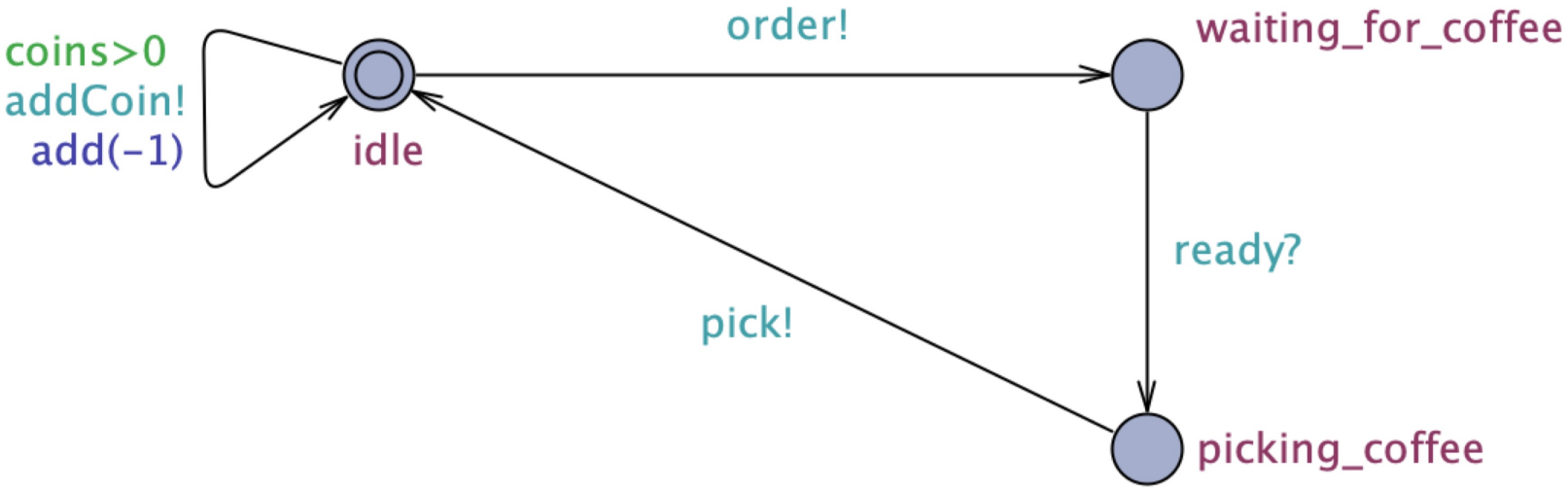
Case Study: Coffee Machine

Coffee Machine Template:



Case Study: Coffee Machine

User Template:



Uppaal Verifier

Properties in Uppaal are expressed using a query language based on a simplified version of **TCTL** (notice that ϕ and ψ are always atomic propositions, expressions cannot be nested):

- E<>(ϕ): a state in which property ϕ holds is reachable from the initial state;
- A<>(ϕ): all paths from the initial state are such that a state in which ϕ holds is reachable;
- A[ ](ϕ): all states reachable from the initial state are such that ϕ holds;
- E[ ](ϕ): there exists a path from the initial state such that ϕ holds in all states along the path;
- ϕ –––> ψ: all paths from the initial state are such that, if ϕ holds in a state, eventually ψ holds.

You can find the whole documentation about the query language in the Web-Help.

Case Study: Coffee Machine

We can check some properties about our Coffee Machine example:

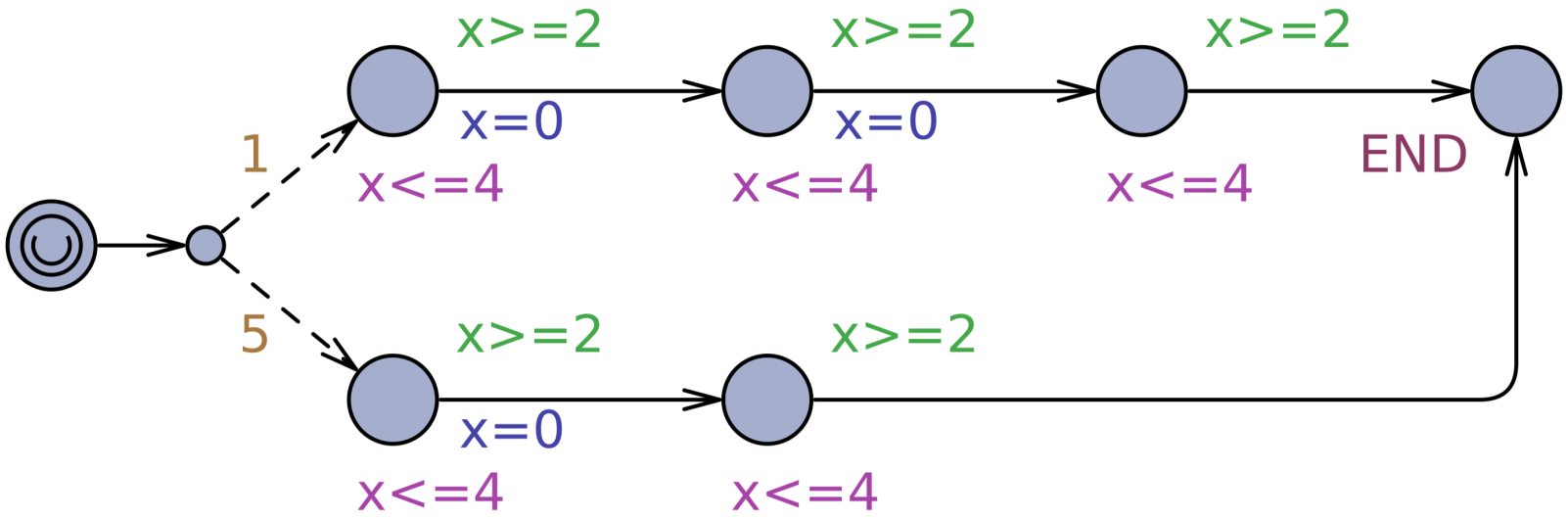
- E<>(*m.timed_out*): there exists a path along which the machine eventually times out;
- E<>(*m.making_coffee && m.x > m.T*): there exists a path in which the machine takes longer than *T* to make a coffee;
- *m.making_coffee* −−> *m.coffee_is_ready*: making the coffee always leads to the coffee being ready;
- E<>(*m.has_coins && m.coins <= 0*): there exists a path in which the machine is in state *has_coins* and there is no credit;
- A[ ](*deadlock imply* (*m.timed out* || (*m.coins <= 0 && u.coins <= 0*))): the system is in a deadlock state only if the machine has timed out or the machine has no credit and the user is out of coins.

Use Uppaal to check which of these properties are verified: some of them correspond to undesirable situations, so they should not hold if the model has been properly designed.

Statistical Model Checking

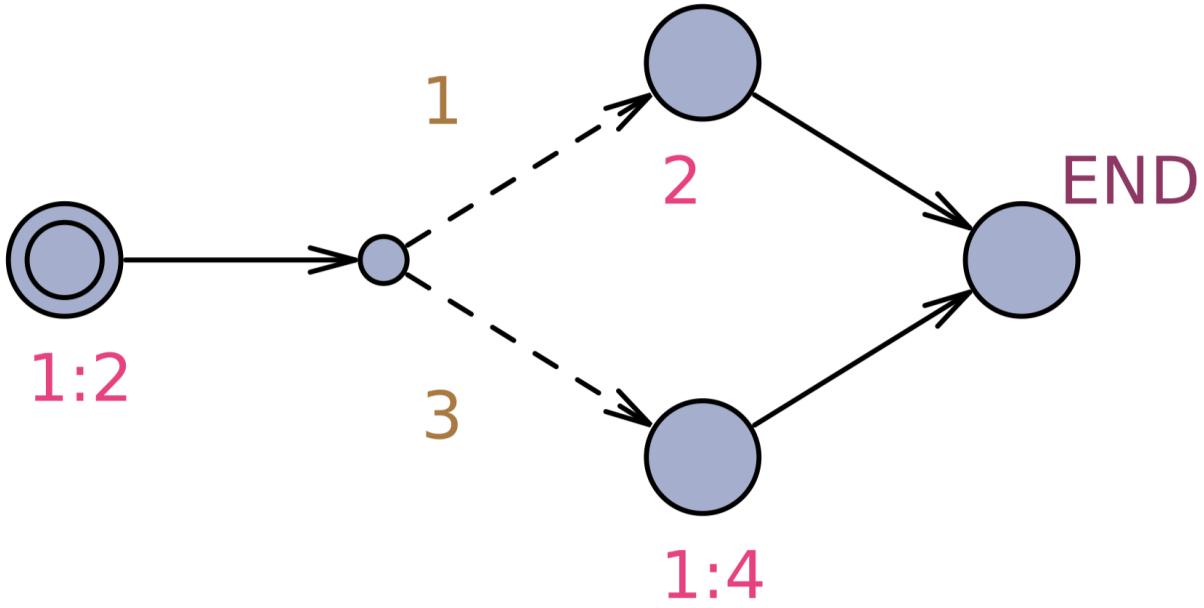
- Templates can be enriched with **stochastic** features, obtaining so-called Stochastic Timed Automata (STA).
- In this case, Uppaal models are built as a **Network of STA** (NSTA).
- Stochastic features are available in **Uppaal SMC**, which is directly integrated in the main tool.
- In NSTA, communication is restricted to **BROADCAST CHANNELS** to keep a clean semantics of only non-blocked components which are racing against each other with their corresponding local distributions.

It is possible to specify the probability of taking some edges:



If an edge has probabilistic branches, then the probability of taking a branch i is determined by the ratio w_i/W, where w_i is the weight of the branch i and W is the sum of all branch weights: $W = \Sigma_j w_j$.

It is possible to specify a probabilistic invariant:



Delays are chosen according to exponential distributions with user-supplied rates (here 1/2, 2, and 1/4).

Statistical Model Checking

The query language is extended with statistical operators:

- *simulate N* [< = *bound*] { E_1, ..., E_k }
  - where *N* is a natural number indicating the number of simulations to be performed, *bound* is the time bound on the simulations, and E_1, ..., E_k are the *k* (state-based) expressions that are to be monitored and visualized.

- *Pr* [*bound*] (*ap*)
  - Where *bound* is the time bound on the simulations and *ap* is an Uppaal expression preceded by either <> or [].