The present work was submitted to the LuFG Theory of Hybrid Systems

MASTER OF SCIENCE THESIS

APPROXIMATE MODEL CHECKING FOR PROBABILISTIC RECTANGULAR AUTOMATA WITH CONTINUOUS-TIME PROBABILITY DISTRIBUTIONS ON JUMPS

Mengzhe Hua

Examiners:
Prof. Dr. Erika Ábrahám
Prof. Dr. Thomas Noll

Additional Advisor:
Stefan Schupp

Aachen, May 05. 2021

Abstract

A system consists of both discrete and continuous components is hybrid system. Most systems in our daily life are hybrid systems. A model to present the hybrid system is hybrid automata. We model the systems into hybrid automata to do mathematical analysis. Rectangular automata is an important subclass of hybrid automata by adding restrictions to labeling functions, and this thesis focus on rectangular automata. In oder to avoid nondeterministic behaviors, we introduce probabilistic rectangular automata with continuous-time probability distributions.

A popular approach to verify certain properties is to do reachability analysis. We introduce forward and backward analysis to get the reachability of request states. For each reachable state we can calculate the probability to reach it from the initial states according to corresponding probability distributions.

Contents

Chapter 1

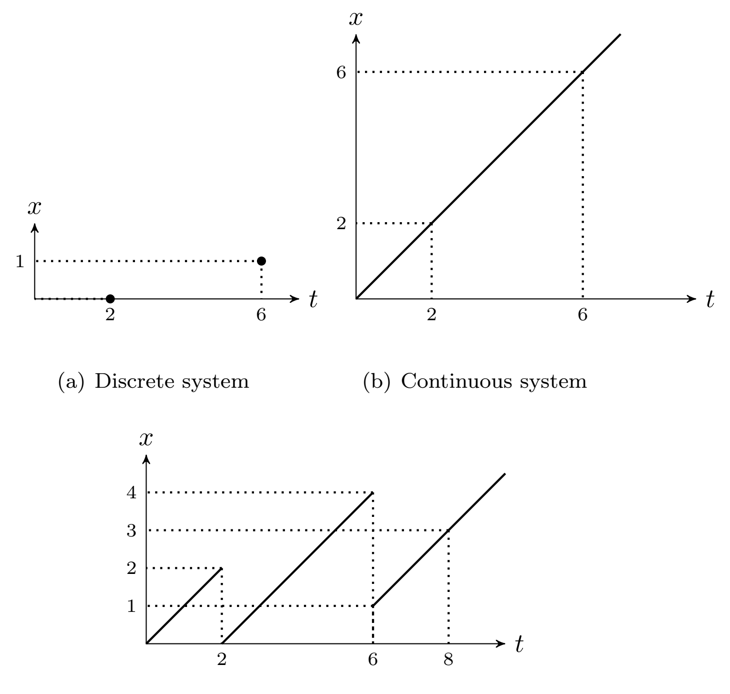
Introduction

Systems with discrete state changing are called discrete system [HMU01] and systems, whose state variables change over time continuously, are called continuous systems. A system that consists of both discrete and continuous components is a hybrid system. Most systems in our daily life are hybrid systems, like computers, mobile-phones, airplanes and so on. Figure 1.1(a) shows how variable x changes over time in a discrete system, and Figure 1.1(b) shows how variable x changes over time in a continuous system. If we combine the two systems, we will get a hybrid system and the evolution of variable x shown in Figure 1.1(c). In oder to analyze the hybrid systems mathematically, hybrid automata is introduced to model those systems. An important subclass of hybrid automata is rectangular automata, for which the model-checking problem is decidable [HM00]. Examples such as Billiard Balls [PDMV15], which models the motion of a billiard ball and its collisions with the wall, is a rectangular automaton.

Billiard balls system is a rectangular automaton without nondeterminism. However many systems contain nondeterministic behaviors, such as IEEE 802.11 WLAN protocol [KNS02] which describes a two-way handshake mechanism to achieve communication between two devices, in the meanwhile introduce a randomized exponential back-off rule to handle with possible transmission conclusions. In oder to deal with the nondeterminism caused by the randomized behavior, we consider to model the system as a probabilistic automaton.

One of the most popular modeling language to deal with probability is Continuous-Time Markov Chain (CTMC for short). A CTMC is a stochastic process, in which the probability of moving from one state to another state is dependent on a transition rate matrix. However, a huge amount of probabilistic systems may allow different types of probability distributions, such as discrete probability distribution. In this case we consider to analyze models containing both continuous and discrete probability distributions.

Stochastic Petri Nets (SPN for short) is one of the approaches which consist not only stochastic process. In a SPN model, the probability of taking a transition is associated with the probability distribution over delays on the state, i.e. the probability of taking a transition is related to the time spent on the source state [BK02]. Another approach is Stochastic Timed Automata (STA for short). In a STA model, the probability of taking a transition is independent on the delays, which means that the combined continuous and discrete probability distribution is associated to transitions [BBB^{+}14].



(a) Discrete system (b) Continuous system

(c) Hybrid system

Figure 1.1: Variable changing over time in three types systems

Since timed automata is a subclass of rectangular automata [Kop96], we extend the stochastic timed automata to stochastic rectangular automata and apply reachability analysis on stochastic rectangular automata.

1.1 Structure of the Thesis

In oder to introduce stochastic rectangular automata properly, we introduce necessary preliminaries in Chapter 2. Firstly we give the formal definition of hybrid automata, which builds hybrid systems mathematically. By giving rectangular region restrictions we introduce a subclass of hybrid automata, rectangular automata. In oder to check some essential properties, we introduce reachability analysis consisting of forward and backward methods.

After we get a clear view of rectangular automata, we define stochastic rectangular automata in Chapter 3. Discrete-time probabilistic rectangular automata model the systems which only associate discrete probability distribution over transitions. After given the definition of probability distributions over delays and transition, we extend it to the definition of stochastic rectangular automata.

The implementation is introduced in Chapter 4. The most critical part is the interval calculation, because the probability to take a transition out-going of the state and calculation of enabled transitions are dependent on the interval in which a given transition may be taken from the state. Finally we introduce the brief idea of how to achieve reachability analysis by forward and backward method in stochastic rectangular automata.

At the end we present the experimental results in Chapter 5, starting with introduction of the tool HyPro. The discussion of our future work is in Chapter 6.

Chapter 2

Preliminaries

Rectangular automata is a subclass of hybrid automata [HKPV98]. We present hybrid automata first and then introduce rectangular automata based on the definition of hybrid automata.

2.1 Hybrid Automata

Hybrid systems are systems which combine the continuous activities and discrete transitions together [Bra05]. A bouncing ball system [JELS99] is a typical hybrid system. An elastic ball is dropped from a certain height h_0 and bounces with energy loss after reaching the ground. The height and the velocity of the ball change continuously between each bounce, and the velocity of the ball changes discretely when bouncing. In oder to model hybrid systems we introduce hybrid automata which are graph specifications of hybrid systems. Hybrid automata specify discrete transitions by edges and continuous activities by vertices.

Definition 2.1.1 (Syntax of Hybrid Automata [ACH+95])**.** *A hybrid automaton (HA) $\mathcal{H}$ is a tuple $\mathcal{H} = (Loc, Var, Lab, Edge, Act, Inv, Init)$ with*

- *Loc is a finite set of locations.*
- *Var is a finite set of real-valued variables. A valuation ν for the variables is a function $\nu : Var \to \mathbb{R}$ assigning a real value to each variable $x \in Var$. We use V to denote the set of valuations.*

  A state is a pair $s = (l,\nu)$ where $l \in Loc$ and $\nu \in V$. We use Σ to denote the set of states.
- *Lab is a finite set of synchronization labels, including the stutter label $\tau \in Lab$.*
- *Edge is a finite set of transitions. Each transition $e = (l, a, \mu, l')$ consists of a source location $l \in Loc$, a target location $l' \in Loc$, a synchronization label $a \in Lab$, and a transition relation $\mu \subseteq V^2$. If $(\nu,\nu') \in \mu$ we call ν the predecessor and ν' the successor. For all transitions with label τ we call τ-transitions.*
- *Act is a labeling function that assigns to each location $l \in Loc$ a set of activities. Each activity is a continuous function $f : \mathbb{R} \to V$ that assigns nonnegative reals to a valuation in V. The activities of each location are time-invariant: for all*

locations $l \in Loc$, all activities $f \in Act(l)$, and nonnegative reals $t \in \mathbb{R}_{\geq 0}$, $(f+t) \in Act(l)$ holds where $(f+t)(t') = f(t+t')$ for all $t' \in \mathbb{R}_{\geq 0}$.

- *Inv is a labeling function, assigning an invariant Inv(l) $\subseteq V$ to each location $l \in Loc$.*
- *Init is a set of initial states with Init $\subseteq \Sigma$.*

Intuitively, variable $x \in Var$ represents a continuous activity of the hybrid system and $f \in Act(l)$ with $l \in Loc$ shows how x changes according to time in state (l,ν) where $\nu \in V$. A transition $e = (l, a, \mu, l') \in Edge$ represents a discrete transition of the hybrid system. In the rest of this thesis we use $\rightarrow$ to denote both continuous activities and discrete transitions.

Definition 2.1.2 (Semantics of Hybrid Automata [ACH+95])**.** *As hybrid systems contain discrete transitions and continuous activities, the semantics of a hybrid automaton $\mathcal{H} = (Loc, Var, Lab, Edge, Act, Inv, Init)$ consist of two parts that are discrete instantaneous steps (also called jump) and continuous time steps (also called flow):*

1. *Discrete step semantics*

$$\frac{(l, a, \mu, l') \in Edge \quad (\nu, \nu') \in \mu \quad \nu' \in Inv(l')}{(l, \nu) \xrightarrow{a} (l',\nu')} Rule_{discrete}$$

for state $s = (l, \nu) \in \Sigma$ and for each discrete transition $e = (l, a, \mu, l') \in Edge$ with source location l we say that e is enabled if and only if some $(\nu,\nu') \in \mu$ is satisfied and synchronization label a occurs.

2. *Time step semantics*

$$\frac{f \in Act(l) \quad f(0) = \nu \quad f(t) = \nu' \quad t \geq 0 \quad f([0,t]) \subseteq Inv(l)}{(l,\nu) \xrightarrow{t} (l,\nu')} Rule_{time}$$

for state $s = (l, \nu) \in \Sigma$ and for each continuous activity $f \in Act(l)$ we say that the system stay at location l for t time step and the valuation changes to $f(t)$ only if no valuation in $[0,t]$ violates the invariant of the location l.

For a jump $e = (l, a, \mu, l') \in Edge$ with $(\nu,\nu') \in \mu$ we say that the predecessor ν satisfies the guard g of e and the successor ν' equals to the reset r of e. In the rest of this chapter we denote $\mu = (g,r)$ for jumps.

Example 2.1.1. *Figure 2.1 shows a simple hybrid automaton $\mathcal{H}_1 = (Loc_1, Var_1, Lab_1, Edge_1, Act_1, Inv_1, Init_1)$ with*

- $Loc_1 = \{l_0,l_1\}$,
- $Var_1 = \{x,y\}$,
- $Lab_1 = \{\tau,a\}$,
- $Edge_1 = \{e_0,e_1,e_2,e_3,e_4\}$ *where*
  - $e_0 = (l_0, \tau, (\nu,\nu), l_0)$ *with* $\nu \in V_{l_0}$,
  - $e_1 = (l_1, \tau, (\nu',\nu'), l_1)$ *with* $\nu' \in V_{l_1}$,

  - $e_2 = (l_0, a, (\{y \geq 5\}, \{x := 0, y := 10\}), l_1)$,
  - $e_3 = (l_0, a, (\{y \geq 5\}, \{x := 0, y := 0\}), l_0)$,
  - $e_4 = (l_1, a, (\{y \leq 5\}, \{x := 0, y := 0\}), l_0)$,

  e_0 and e_1 are τ-transitions, e_2, e_3 and e_4 are jumps,

- *activity functions are represented by differential equations: $Act_1(l_0) = \{\dot{x} = 1, \dot{y} = x\}$ and $Act_1(l_1) = \{\dot{x} = -1, \dot{y} = x\}$,*
- *$Inv_1(l_0) = \{y \leq 10\}$ and $Inv_1(l_1) = \{true\}$,*
- *$Init_1 = (l_0, \{x = 0, y = 0\})$.*

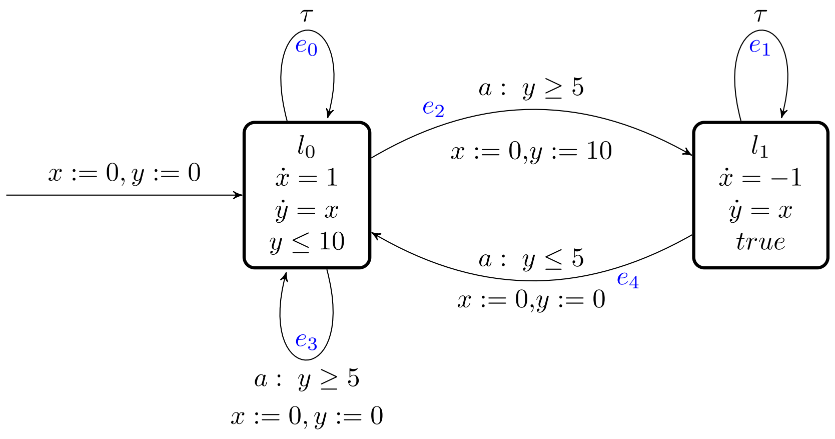


Figure 2.1: A simple hybrid automaton $\mathcal{H}_1$

We introduce flows and jumps in detail separately. Firstly, we take flows in location l_0 as an example: if invariant $Inv_1(l_0) = \{y \leq 10\}$ is satisfied for each time point in $[0,t]$, then the system may stay in location l_0 for t time step. Next, we take e_2 as an example of jumps: If synchronization label a occurs and guard $g = \{y \geq 5\}$ is satisfied at location l_0, then e_2 is enabled; If e_2 is taken, then the system will set the variables to the values given in effect $r = \{x := 0, y := 10\}$, i.e. x will be set to 0 and y will be set to 10, finally the system will reach location l_1 from l_0. However, we notice that e_2 and e_3 have the same source location l_0, and they could be enabled at same time points, we say that the hybrid automaton $\mathcal{H}_1$ is non-deterministic, which means that after e_2 and e_3 enabled the system choose one of enabled edges to take non-deterministically.

To simplify the graphical representations, in the rest of this thesis we omit τ-transition, non-synchronization labels and trivial invariant in graphical representations.

A finite run [AD94] ϱ of hybrid automata $\mathcal{H} = (Loc, Var, Lab, Edge, Act, Inv, Init)$ with state set Σ is a sequence of transitions

$$\varrho = s_0 \xrightarrow{t_1, e_1} s_1 \xrightarrow{t_2, e_2} \ldots \xrightarrow{t_n, e_n} s_n$$

where $n \in \mathbb{N}$, $(s_i) = (l_i,\nu_i) \in \Sigma$ with $0 \leq i \leq n$, $(t_i)_{1 \leq i \leq n} \in \mathbb{R}_{\geq 0}$, $t_0 = 0$ and $e_i = (l_{i-1},a_i,(g_i,r_i),l_i) \in \mathit{Edge}$ with $1 \leq i \leq n$, $a_i \in \mathit{Lab}$, $g_i \in \mathcal{R}^n$ and $r_i \in \mathcal{R}^n$. Each transition $s_{i-1} \xrightarrow{t_i,e_i} s_i$ with $1 \leq i \leq n$ satisfies the following requirements:

- $\nu_{i-1} + f_{i-1}(t) \models g_i$,
- $\nu_{i-1} + f_{i-1}([0,t]) \subseteq \mathit{Inv}(i-1)$,
- $\nu_i = r_i$,

where $t = t_i - t_{i-1}$ and $f_i \in \mathit{Act}(l_i)$. We say that n is the length of the run ϱ. We use $\mathit{Run}_f(\mathcal{H},s_0)$ to denote the set of all finite runs from s_0 in $\mathcal{H}$. Similarly an infinite run [AD94] ϱ of $\mathcal{H}$ is an infinite sequence of transitions.

$$\varrho = s_0 \xrightarrow{t_1,e_1} s_1 \xrightarrow{t_2,e_2} s_2 \xrightarrow{t_3,e_3} \ldots$$

with $(s_i)_{i\in\mathbb{N}} \in \Sigma$, $(t_i)_{i\in\mathbb{N}^+} \in \mathbb{R}_{\geq 0}$ and $(e_i)_{i\in\mathbb{N}^+} \in \mathit{Edge}$, satisfying the above requirements. We use $\mathit{Run}(\mathcal{H},s_0)$ to denote the set of all infinite runs from s_0.

2.2 Rectangular Automata

Rectangular automata is one of the particularly important subclasses of hybrid automata. Rectangular automata inherit all components of hybrid automata and restrict their initializations, invariants, activities, and transition relations by rectangular regions.

Definition 2.2.1 (Rectangular Region)**.** *Given a region $R \subseteq \mathbb{R}^n$ with $n > 0$, R is called rectangular if it is a cartesian product of (possibly unbounded) intervals, all of whose finite endpoints are rational:*

$$\begin{aligned} R &= I_0 \times I_1 \times \ldots \times I_{n-1} \\ &= \{(x_0,x_1,\ldots,x_{n-1}) \in \mathbb{R}^n \mid \forall 0 \leq i \leq n-1.\ x_i \in I_i\}. \end{aligned}$$

We use $\mathcal{R}^n$ to denote the set of all rectangular regions in $\mathbb{R}^n$.

Definition 2.2.2 (Syntax of Rectangular Automata [HKPV98])**.** *An n-dimensional rectangular automaton (shortly rectangular automaton or RA) is a hybrid automaton $\mathcal{H} = (\mathit{Loc}, \mathit{Var}, \mathit{Lab}, \mathit{Edge}, \mathit{Act}, \mathit{Inv}, \mathit{Init})$ with restricted labeling function:*

- *Edge is a finite set of transitions. Each transition $e = (l, a, \mu, jump, l')$ consists of a source location $l \in Loc$, a target location $l' \in Loc$, a synchronization label $a \in Lab$, and a transition relation $(\mu, jump) \subseteq (\mathcal{R}^n \times \mathcal{R}^n) \times 2^{\{1,\ldots,n\}}$ with $\mu = (g,r)$. For each $i \in jump$ the value of variable $x_i \in Var$ will be changed to a value $v \in r_i$ non-deterministically and for each $i \notin jump$ the value of variable $x_i \in Var$ does not change.*
- *$Act : Loc \to \mathcal{R}^n$.*
- *$Inv : Loc \to \mathcal{R}^n$.*
- *$Init : Loc \to \mathcal{R}^n$.*

Definition 2.2.3 (Semantics of Rectangular Automata [HKPV98]). *The semantics of a rectangular automaton $\mathcal{H} = (Loc, Var, Lab, Edge, Act, Inv, Init)$ also consist of two parts that are discrete instantaneous steps (also called jump) and continuous time steps (also called flow):*

1. *Discrete step semantics*

$$\frac{\begin{array}{c}(l, a, (g, r), jump, l') \in Edge \\ \nu \in g \quad \nu' \in r \quad \forall i \notin jump.\nu'_i = \nu_i \quad \nu' \in Inv(l')\end{array}}{(l, \nu) \xrightarrow{a} (l', \nu')} Rule_{discrete}$$

for state $s = (l, \nu) \in \Sigma$ and for each discrete transition $e = (l, a, (g, r), jump, l') \in Edge$ of location l we say that e is enabled if and only if the valuation ν satisfies the guard g and synchronization label a occurs, the system may take the transition e and in the meanwhile update the value of variables in jump to r only if updated valuation ν' does not violate the invariant of target location.

2. *Time step semantics*

$$\frac{(t = 0 \wedge \nu = \nu') \vee (t > 0 \wedge (\nu' - \nu)/t \in Act(l)) \quad \nu' \in Inv(l)}{(l,\nu) \xrightarrow{t} (l,\nu')} Rule_{time}$$

for state $s = (l, \nu) \in \Sigma$ the system may stay at location for t time step and update valuation to ν' only if either t is zero and valuation does not change or the ratio of difference between ν' and ν to t meets the continuous activities of l and ν' does not violate the invariant of l.

We say that the rectangular automaton $\mathcal{H}$ is initialized, if for every edge $e = (l, a, (g, r), jump, l')$ of $\mathcal{H}$, every variable index $i \in \{1, \dots, n\}$ with $Act(l)_i \neq Act(l')_i$ we have $i \in jump$.

Example 2.2.1. *Figure 2.2 shows a simple rectangular automaton $\mathcal{H}_2 = (Loc_2, Var_2, Lab_2, Edge_2, Act_2, Inv_2, Init_2)$. $\mathcal{H}_2$ is quite similar to $\mathcal{H}_1$, the main difference between $\mathcal{H}_2$ and $\mathcal{H}_1$ is the labeling function. Act_1 in $\mathcal{H}_1$ contains linear differential equation $\dot{y} = x$, which is not restricted to rectangular regions. However all activity functions in $Act_2 = \{\{\dot{x} = 1, \dot{y} = 2\}, \{\dot{x} = -1, \dot{y} \in [-2, -1]\}\}$ are obviously restricted to rectangular regions. Another difference is that if e_4 is taken, then the value of y in $\mathcal{H}_2$ will be changed to a value $v \in [0,10]$ non-deterministically. We notice that e_2 and e_3 have the same source location, and they could be enabled at same time points. Therefore $\mathcal{H}_2$ is also non-deterministic.*

If we remove edge e_3 in $\mathcal{H}_2$, we will get an initialized rectangular automaton, because for each $e_i = (l,a,(g,r),jump,l') \in Edge_2$ with $i \in \{2,4\}$ we have $Act_2(l)(x) \neq Act_2(l')(x)$ and $Act_2(l)(y) \neq Act_2(l')(y)$.

Given a rectangular automaton $\mathcal{H} = (Loc, Var, Lab, Edge, Act, Inv, Init)$ with state set Σ, a finite run [AD94] ϱ of $\mathcal{H}$ is a sequence of transitions

$$\varrho = s_0 \xrightarrow{t_1, e_1} s_1 \xrightarrow{t_2, e_2} \dots \xrightarrow{t_n, e_n} s_n$$

where $n \in \mathbb{N}$, $(s_i) = (l_i, \nu_i) \in \Sigma$ with $0 \leq i \leq n$, $(t_i)_{1 \leq i \leq n} \in \mathbb{R}_{\geq 0}$, $t_0 = 0$ and $e_i = (l_{i-1}, a_i, (g_i, r_i), jump_i, l_i) \in Edge$ with $1 \leq i \leq n$, $a_i \in Lab$, $g_i \in \mathcal{R}^n$ and $r_i \in \mathcal{R}^n$. Each transition $s_{i-1} \xrightarrow{t_i, e_i} s_i$ with $1 \leq i \leq n$ satisfies the following requirements:

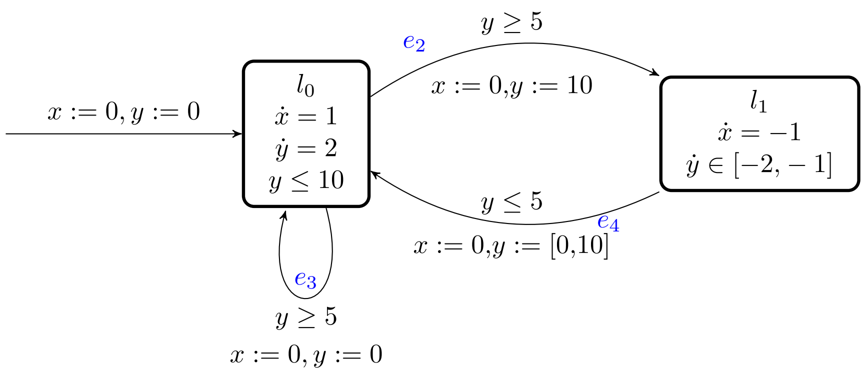


Figure 2.2: A simple rectangular automaton $\mathcal{H}_2$

- $\nu_{i-1} + f_{i-1}(t) \models g_i$,
- $\nu_{i-1} + f_{i-1}([0,t]) \subseteq Inv(i-1)$,
- $\forall x \notin jump_i.\ \nu_i(x) = \nu_{i-1} + f_{i-1}(t)$,
- $\forall x \in jump_i. \nu_i(x) = r_i(x)$,

where $t = t_i - t_{i-1}$ and $f_i \in Act(l_i)$. We say that n is the length of the run ϱ. We use $Run_f(\mathcal{H},s_0)$ to denote the set of all finite runs from s_0 in $\mathcal{H}$. Similarly an infinite run [AD94] ϱ of $\mathcal{H}$ is an infinite sequence of transitions.

$$\varrho = s_0 \xrightarrow{t_1,e_1} s_1 \xrightarrow{t_2,e_2} s_2 \xrightarrow{t_3,e_3} \dots$$

with $(s_i)_{i\in\mathbb{N}} \in \Sigma$, $(t_i)_{i\in\mathbb{N}^+} \in \mathbb{R}_{\geq 0}$ and $(e_i)_{i\in\mathbb{N}^+} \in Edge$, satisfying the above requirements. We use $Run(\mathcal{H},s_0)$ to denote the set of all infinite runs from s_0.

Definition 2.2.4 (Direct Successors [BK08])**.** *Let $\mathcal{H} = (Loc, Var, Lab, Edge, Act, Inv, Init)$ be a rectangular automaton. For $s \in \Sigma$ and $e \in Edge$, the set of direct e-successors of s is defined as:*

$$Post(s,e) = \{s' \in \Sigma \mid \exists t \in \mathbb{R}_{\geq 0}.s \xrightarrow{t,e} s'\},$$

and the set of direct successors of s is defined as:

$$Post(s) = \bigcup_{e\in Edge} Post(s,e).$$

Each state $s' \in Post(s,e)$ is a direct e-successor of s, and each state $s' \in Post(s)$ is a direct successor of s.

Given a rectangular automaton $\mathcal{H} = (Loc, Var, Lab, Edge, Act, Inv, Init)$ with state set Σ, a finite path $\widehat{\pi}(s_0, e_1 \dots e_n)$ of $\mathcal{H}$ is a set of all finite runs starting from s_0 by taking $e_1,e_2,\dots,e_n$, we define $\widehat{\pi}(s_0, e_1 \dots e_n) = \{\varrho = s_0 \xrightarrow{t_1,e_1} s_1 \xrightarrow{t_2,e_2} \dots \xrightarrow{t_n,e_n} s_n \in Run_f(\mathcal{H},s_0) \mid \forall i \in \mathbb{N}.(1 \leq i \leq n \wedge t_i \in \mathbb{R}_{\geq 0} \wedge s_i \in Post(s_{i-1},e_i))\}$. An infinite path $\pi(s_0, e_1e_2\dots)$ of $\mathcal{H}$ is a set of all infinite runs that start from s_0 and take $e_1,e_2,\dots$ in sequence, we define $\pi(s_0, e_1e_2\dots) = \{\varrho = s_0 \xrightarrow{t_1,e_1} s_1 \xrightarrow{t_2,e_2} s_2 \xrightarrow{t_3,e_3} \dots \in Run(\mathcal{H},s_0) \mid \forall i \in \mathbb{N}.(i \geq 1 \wedge t_i \in \mathbb{R}_{\geq 0} \wedge s_i \in Post(s_{i-1},e_i))\}$.

Example 2.2.2. *We take a rectangular automaton $\mathcal{H}_2$ from Example 2.2.1. Assume $s_0 = (l_0,\{x = 0,y = 0\})$ and $s_1 = (l_1,\{x = 0,y = 10\})$ are states of $\mathcal{H}_2$. A possible finite run of $\mathcal{H}_2$ is $\varrho_1 = s_0 \xrightarrow{3,e_3} s_0 \xrightarrow{7,e_3} s_0 \xrightarrow{12,e_2} s_1$, accordingly $\widehat{\pi}(s_0, e_3e_3e_2)$ is the finite path fragment with $\varrho_1 \in \widehat{\pi}(s_0, e_3e_3e_2)$. A possible infinite run of $\mathcal{H}_2$ is $\varrho_2 = s_0 \xrightarrow{3,e_3} s_0 \xrightarrow{6,e_3} s_0 \xrightarrow{9,e_3} \ldots$, accordingly $\pi(s_0, e_3e_3 \ldots)$ is the infinite path fragment with $\varrho_2 \in \pi(s_0, e_3e_3 \ldots)$.*

Given a rectangular automaton $\mathcal{H} = (\mathit{Loc}, \mathit{Var}, \mathit{Lab}, \mathit{Edge}, \mathit{Act}, \mathit{Inv}, \mathit{Init})$, a state $s \in \Sigma$ and a transition $e \in \mathit{Edge}$ of $\mathcal{H}$, we define $I(s,e) = \{t \in \mathbb{R}_{\geq 0} \mid \exists s' \in \Sigma_{\mathcal{H}}.s \xrightarrow{t,e} s'\}$ an interval that may be spent on s before e taken and $I(s) = \bigcup_{e \in \mathit{Edge}} I(s,e)$ an union of intervals that may be spent on s before any transition taken.

Example 2.2.3. *We take a rectangular automaton $\mathcal{H}_2$ from Example 2.2.1. Assume $s_0 = (l_0,\{x = 0,y = 0\})$ and $s_1 = (l_1,\{x = 0,y = 10\})$ are states of $\mathcal{H}_2$. We have $I(s_0, e_3)$ is $[2.5,5]$ and $I(s_0, e_2)$ is also $[2.5,5]$, so $I(s_0)$ is $[2.5,5]$. As Inv(l_1) is true the system may stay in l_1 infinitely many time, we have $I(s_1,e_4) = [2.5, +\infty)$.*

Given a rectangular automaton $\mathcal{H}$, if $I(s) \neq \emptyset$ for every state $s \in \Sigma_{\mathcal{H}}$ is satisfied, we say that $\mathcal{H}$ is a non-blocking rectangular automaton. Intuitively, for each state s of a non-blocking rectangular automaton there exists at least one jump transition that may be taken at some time point. $\mathcal{H}_1$ and $\mathcal{H}_2$ above are both non-blocking. In the rest of this thesis we only consider non-blocking rectangular automata.

2.3 Reachability Analysis

Generally we need to check whether certain functional events occur during the running time, thus we verify if certain properties are satisfied by the hybrid automata. Usually we consider the following two types of properties:

1. Safety properties: state that “nothing bad will happen”. We verify if all states, which are defined as “bad states”, are not reachable.

2. Liveness properties: state that “something good will happen”. We verify if some states, which are defined as “good states”, are reached.

In order to verify *safety* properties, we have to explore all the reachable states of the hybrid automata and intersect the reachable states with “bad states”. If the intersection is an empty set we say that the *safety* properties is satisfied, otherwise not satisfied. This approach is called reachability analysis. Given a hybrid automaton $\mathcal{H} = (\mathit{Loc}, \mathit{Var}, \mathit{Lab}, \mathit{Edge}, \mathit{Act}, \mathit{Inv}, \mathit{Init})$ and s_0, s_1 are two states of $\mathcal{H}$. We call s_1 is reachable from s_0 if there is a path from s_0 to s_1, written as $s_0 \mapsto^* s_1$ [ACH+95]. The reachability problem of hybrid automata is to ask, whether $s_0 \mapsto^* s_1$ holds by given a hybrid automaton $\mathcal{H}$ with its states s_0 and s_1.

The general reachability problem for hybrid automata is undecidable [ACHH92]. Fortunately the time-bounded reachability problem for rectangular automata is decidable proved in [BDG+11]. Thus it is meaningful to apply reachability analysis on rectangular automata.

2.3.1 Forward Analysis

Informally speaking forward analysis [ACH$^+$95] is to compute the reachable state set $Reach^+(Init)$ starting from initial state set $Init$ as $Reach^+(Init) = \{s \in \Sigma \mid \exists s_0 \in Init.\ s_0 \mapsto^* s\}$. As each hybrid automaton has two types of steps which are jump and flow, we discuss one-step reachability separately.

We define the *forward time closure* $\langle P\rangle_l^{\nearrow}$ over a set of valuations $P \subseteq V$ at a location $l \in Loc$ as

$$\nu \in \langle P\rangle_l^{\nearrow} \quad \Leftrightarrow \quad \exists \nu^{pre} \in V, t \in \mathbb{R}^{\geq 0}.\ \nu^{pre} \in P \wedge \nu = Act_l[\nu^{pre}](t) \wedge Inv_l[\nu],$$

a set of states is called region $R_l = \{(l,\nu) \mid \nu \in P\}$, and we define the *region* of forward time closure $\langle R_l\rangle_l^{\nearrow}$ as

$$\langle R_l\rangle_l^{\nearrow} = (l, \langle P\rangle_l^{\nearrow}),$$

and we extend to regions of all locations $R = \bigcup_{l\in Loc} R_l$:

$$\langle R\rangle^{\nearrow} = \bigcup_{l\in Loc} \langle R_l\rangle_l^{\nearrow}.$$

We define the *postcondition* $post_e[P]$ over a set of valuations $P \subseteq V$ with respect to an edge $e = (l, a, \mu, l')$ as

$$\nu \in post_e[P] \quad \Leftrightarrow \quad \exists \nu^{pre} \in V.\ \nu^{pre} \in P \wedge (\nu^{pre}, \nu) \in \mu \wedge Inv_{l'}[\nu],$$

the *region* $post_e[R_l]$ as

$$post_e[R_l] = (l', post_e[P]),$$

where $R_l = \{(l, \nu) | \nu \in P\}$, and the the set of *regions* $post[R]$ as

$$post[R] = \bigcup_{e\in Edge} post_e[R_l],$$

where $R = \bigcup_{l\in Loc} R_l$.

We use reachable region $(I \mapsto^*) \subseteq \Sigma$ to denote the set of all reachable states from states in the initial region $I \subseteq \Sigma$:

$$s \in (I \mapsto^*) \Leftrightarrow \exists s_0 \in I.\ s_0 \mapsto^* s.$$

Example 2.3.1. *Consider the hybrid automaton $\mathcal{H}_2$ in Figure 2.2. We compute the following set by using forward analysis:*

- *the set $\langle x = 0 \wedge y = 0\rangle_{l_0}^{\nearrow}$ reachable from $x = 0 \wedge y = 0$ in location l_0:*

$$\begin{aligned}\langle x = 0 \wedge y = 0\rangle_{l_0}^{\nearrow} =& \exists x^{pre} \exists y^{pre} \exists t \in \mathbb{R}^{\geq 0}. x^{pre} = 0 \wedge y^{pre} = 0 \\ & \wedge x = x^{pre} + t \wedge y = y^{pre} + 2t \wedge y \leq 10 \\ =& \exists t \in \mathbb{R}^{\geq 0}. x = t \wedge y = 2t \wedge y \leq 10 \\ =& x \geq 0 \wedge 0 \leq y \leq 10\end{aligned}$$

- *the region $\langle R_{l_0}\rangle_{l_0}^{\nearrow} = (l_0, x \geq 0 \wedge 0 \leq y \leq 10)$*

- *the set* $post_{e_2}[x \geq 0 \land 0 \leq y \leq 10]$ *reachable from* $x \geq 0 \land 0 \leq y \leq 10$ *in location* l_0 *by taking transition* e_2*:*

$$\begin{aligned} post_{e_2}[x \geq 0 \land 0 \leq y \leq 10] =& \exists x^{pre} \exists y^{pre}. x^{pre} \geq 0 \land 0 \leq y^{pre} \leq 10 \land y^{pre} \geq 5 \\ & \land x = 0 \land y = 10 \land y \geq 0 \\ =& x = 0 \land y = 10 \end{aligned}$$

- *the region* $post_{e_2}[R_{l_0}] = (l_1, x = 0 \land y = 10)$

The main idea of forward analysis is to calculate $(I \mapsto^*) \subseteq \Sigma$ of states which are reachable from the initial states I. The general forward reachability analysis algorithm shows in Algorithm 1.

Algorithm 1: General forward reachability analysis algorithm [SFÁ19]

```
  Input: Set of initial states $I$
  Output: Set of reachable states $R$
1 $R := I$
2 $R_{new} := I$
3 while $R_{new} \neq \varnothing$ do
4     $R_{new} :=$ Reach($R_{new}$) $\setminus R$
5     $R := R \bigcup R_{new}$
6 end
```

2.3.2 Backward Analysis

Informally speaking backward analysis [ACH+95] could be seen as the reverse process of forward analysis which is to compute the reachable state set $Reach^{-}(Goal)$ starting from a set of target states $Goal$, written as $Reach^{-}(Goal) = \{s \in \Sigma \mid \exists s' \in Goal.\ s \mapsto^* s'\}$. Similarly, we discuss one-step reachability under time and discrete steps separately.

We define the *backward time closure* $\langle P \rangle_l^{\swarrow}$ over a set of valuations $P \subseteq V$ at a location $l \in Loc$ as

$$\nu \in \langle P \rangle_l^{\swarrow} \quad \Leftrightarrow \quad \exists \nu^{post} \in V, t \in \mathbb{R}^{\geq 0}.\ \nu^{post} = Act_l[\nu](t) \land \nu^{post} \in P \land Inv_l[\nu],$$

the *region* $\langle R_l \rangle_l^{\swarrow}$ where $R_l = \{(l, \nu) \mid \nu \in P\}$ as

$$\langle R_l \rangle_l^{\swarrow} = (l, \langle P \rangle_l^{\swarrow}),$$

and the the set of *region* $\langle R \rangle^{\swarrow}$ as

$$\langle R \rangle^{\swarrow} = \bigcup_{l \in Loc} \langle R_l \rangle_l^{\swarrow},$$

where $R = \bigcup_{l \in Loc} R_l$.

We define the *precondition* $pre_e[P]$ over a set of valuations $P \subseteq V$ with respect to an edge $e = (l', a, \mu, l)$ as

$$\nu \in pre_e[P] \quad \Leftrightarrow \quad \exists \nu^{post} \in V.\ \nu^{post} \in P \land (\nu, \nu^{post}) \in \mu \land Inv_{l'}[\nu],$$

the *region* $pre_e[R_l]$ as

$$pre_e[R_{l'}] = (l', pre_e[P]),$$

where $R_{l'} = \{(l', \nu) | \nu \in P\}$, and the the set of *region* $pre[R]$ as

$$pre[R] = \bigcup_{e=(l',a,\mu,l) \in Edge} pre_e[R_{l'}],$$

where $R = \bigcup_{l' \in Loc} R_{l'}$.

We use initial region $(\mapsto^* R) \subseteq \Sigma$ to denote the set of all states from which states in the target region $R \subseteq \Sigma$ are reachable:

$$s \in (\mapsto^* R) \Leftrightarrow \exists s' \in R.\ s \mapsto^* s'.$$

Chapter 3

Stochastic Rectangular Automata

Rectangular automata may be non-deterministic, i.e. if more than one transitions, which have the same source location, are enabled at the same time, then system will choose one of the enabled transitions to take non-deterministically. Probabilistic rectangular automata generalize the subclass of rectangular hybrid automata that represent the non-deterministic behavior of the system by probability. Discrete-time probabilistic rectangular automata extend rectangular automata with discrete probability distributions over transitions.

3.1 Discrete-Time Probabilistic Rectangular Automata

Definition 3.1.1 (Syntax of Discrete-Time Probabilistic Rectangular Automata [Spr11]). *A discrete-time probabilistic rectangular automaton (DTPRA) is a tuple $\mathcal{H}$ is a tuple $\mathcal{H} = (Loc, Var, Lab, Edge, Act, Inv, Init, P)$ where:*

- *$(Loc, Var, Lab, Edge, Act, Inv, Init)$ is a rectangular automaton*
- *P is a transition probability function $P : Edge \to [0,1]$ such that:*

$$\forall l \in Loc. \left(\sum_{e \in Edge} P(l,e) = 1 \right)$$

Definition 3.1.2 (Semantics of Probabilistic Rectangular Automata [Spr11]). *The semantics of a probabilistic rectangular automaton $\mathcal{H} = (Loc, Var, Lab, Edge, Act, Inv, Init, P)$ also consist of jumps and flows:*

1. *Discrete step semantics*

$$\frac{\begin{array}{c} e = (l, a, pre, post, jump, l') \in Edge \quad P(l,e) > 0 \\ \nu \in pre \quad \nu' \in post \quad \forall i \notin jump. v'_i = v_i \quad \nu' \in Inv(l') \end{array}}{(l.\nu) \xrightarrow{a} (l',\nu')} Rule_{discrete}$$

except conditions of discrete step semantics of rectangular automata, transition e may be taken under the condition that probability of taking transition e at location l is greater than zero additionally.

2. *Time step semantics are same as that in rectangular automata*

Example 3.1.1. *Figure 3.1 shows a simple discrete-time probabilistic rectangular automaton $\mathcal{H}_3 = (Loc_3, Var_3, Lab_3, Edge_3, Act_3, Inv_3, Init_3)$. Compared to the rectangular automaton $\mathcal{H}_2$ in Figure 2.2, $\mathcal{H}_3$ resolve the non-determination in $\mathcal{H}_2$ by probability. If $y \geq 5$ is satisfied in location l_0, the system will reach location l_1 with the probability 0.9 and reach location l_0 with the probability 0.1 respectively.*

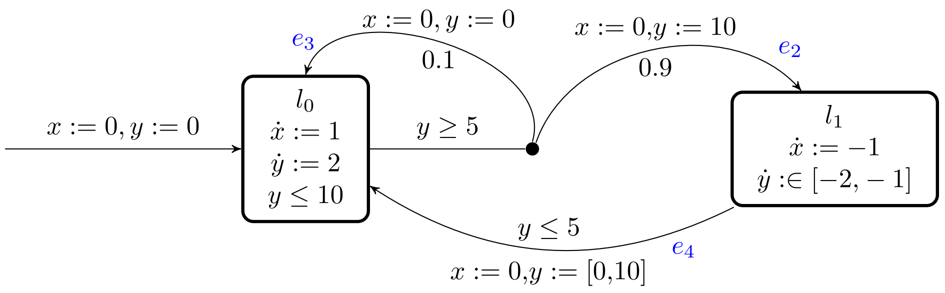


Figure 3.1: A simple discrete-time probabilistic rectangular automaton $\mathcal{H}_3$

3.2 Stochastic Rectangular Automata

Probabilistic rectangular automata using rules based on stochastic process are called stochastic rectangular automata. Intuitively the probability distribution of stochastic rectangular automata has two part, one is over delays, the other is over transitions. In the following we extend stochastic timed automata introduced in [BBB+14] to stochastic rectangular automata. We introduce probability distribution over delays firstly.

Definition 3.2.1 (Probability Distribution over Delays of Rectangular Automata)**.** *Let $\mathcal{H} = (Loc, Var, Lab, Edge, Act, Inv, Init)$ with state set Σ be a non-blocking rectangular automaton. The probability distribution on state $s \in \Sigma$ over delays is a probability measure μ_s over $\mathbb{R}_{\geq 0}$ which satisfies the following requirements:*

(H.1) *$\mu_s(I(s)) = \mu_s(\mathbb{R}_{\geq 0}) = 1$, where $I(s)$ is not empty as $\mathcal{H}$ is non-blocking. The probability distribution is defined over delays in interval $I(s)$, i.e. the probability of all value out of interval $I(s)$ is zero, so we define that $\mu_s(\mathbb{R}_{\geq 0} \backslash I(s)) = 0$;*

(H.2) *Let λ be a standard Lebesgue measure on $\mathbb{R}_{\geq 0}$. We consider the following two situations:*

- *If $\lambda(I(s)) = 0$, then μ_s is equivalent to the uniform distribution over points of $I(s)$. In this case $I(s)$ is a set of single points, because a point has measure zero.*

- *Otherwise, μ_s is equivalent to λ on $I(s)$. As $I(s)$ is one-dimensional set, $\lambda(I(s))$ measures the length of interval $I(s)$.*

In both cases we say that two measures μ and μ' are equivalent, if for every measurable set I, $\mu(I) = 0 \iff \mu'(I) = 0$ holds. Intuitively, we cannot set the probability of enabling some transitions outgoing from s to zero.

Notice that the probability measure μ_s is defined on Borel $\sigma - algebra$.

Next we introduce the probability distribution over transitions.

Definition 3.2.2 (Probability Distribution over Transitions of Rectangular Automata)**.** *Let $\mathcal{H} = (Loc, Var, Lab, Edge, Act, Inv, Init)$ with state set Σ be a non-blocking rectangular automaton. Let p_s be a probability distribution of state s over transitions. By assigning a weight $w(e) > 0$ to each transition $e \in Edge$ the probability of taking e is defined as:*

$$e \text{ is enabled in } s \iff p_s(e) = \frac{w(e)}{\sum_{e' \text{ is enabled}} w(e')} > 0.$$

We extend rectangular automata to stochastic rectangular automata by adding probability distribution over delays and transitions.

Definition 3.2.3 (Stochastic Rectangular Automata)**.** *A stochastic rectangular automaton (SRA) is a tuple $\mathcal{H} = (Loc, Var, Lab, Edge, Act, Inv, Init, \mu, w, \iota_{init})$ with state set Σ where:*

- *$(Loc, Var, Lab, Edge, Act, Inv, Init)$ is a rectangular automaton,*
- *μ is a set of probability distribution over delays: $\mu = \bigcup_{s\in\Sigma} \mu_s$ where μ_s is defined as in Definition 3.2.1,*
- *w is a set of weight functions: $w = \bigcup_{e\in Edge} w(e)$, the probability $p_s(e)$ is defined as in Definition 3.2.2,*
- *ι_{init} is a initial distribution over states: $\iota_{init}(s) = 0$ for all states $s \notin Init$ and $\sum_{s\in\Sigma} \iota_{init}(s) = 1$.*

We will discuss the reachability in the rest of this thesis, if the set of target states is not reachable from initial states after infinitely many steps, i.e. there is no such finite run that ever reaches one of target states, we say that target states are not reachable. In other words once there exists a finite run that reaches at least one of target states, we say that the set of target states is reachable. So what we really interested in is the probability distribution over finite paths that may reach the set of target states.

Definition 3.2.4 (Probability Distribution over Finite Paths of SRA)**.** *Let $\mathcal{H} = (Loc, Var, Lab, Edge, Act, Inv, \mu, w, \iota_{init})$ with state set Σ be a stochastic rectangular automaton and let $\widehat{\pi}(s_0, e_1 \dots e_n)$ be a finite path of $\mathcal{H}$. The probability of $\widehat{\pi}(s_0, e_1 \dots e_n)$ starting from s_0 and taking e_1,e_2,...,e_n in sequence is defined as:*

$$\mathbb{P}_{\mathcal{H}}(\widehat{\pi}(s, e_1 e_2 \dots e_n)) = \int_{t\in I(s,e_1)} p_{s+t}(e_1) \cdot \mathbb{P}_{\mathcal{H}}(\widehat{\pi}(s', e_2 \dots e_n)) d\mu_s(t)$$

where $s+t$ represents the state that stays at the location of s and changes the valuation according to the activities of the location. We use $s \xrightarrow{t} s + t \xrightarrow{e} s'$ to denote the *transition over states which takes flow $s \xrightarrow{t} s+t$ and jump $s+t \xrightarrow{e} s'$ in sequence. Initially, we define $\mathbb{P}_{\mathcal{H}}(\widehat{\pi}(s_0)) = 1$.*

We define the probability that the system starts from s_0 and takes e_1,$e_2, \dots$,e_n in sequence as:

$$\mathbb{P}_{\mathcal{H}}((s, e_1e_2 \dots e_n)) = \iota(s) \cdot \mathbb{P}_{\mathcal{H}}(\widehat{\pi}(s, e_1e_2 \dots e_n)).$$

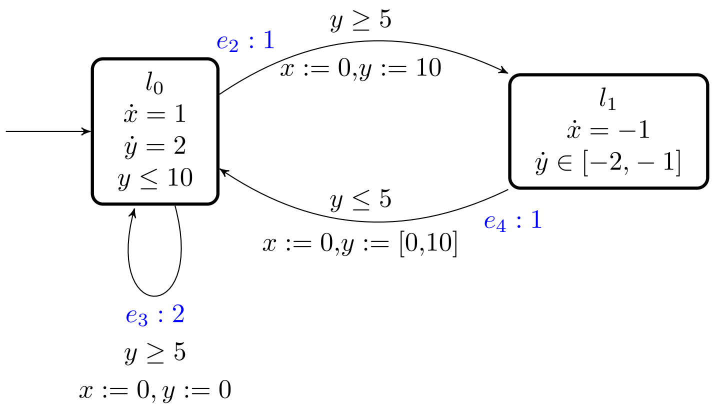


Figure 3.2: A simple stochastic rectangular automaton $\mathcal{H}_4$

Example 3.2.1. *Figure 3.2 shows a simple stochastic rectangular automaton $\mathcal{H}_4 = (Loc_4, Var_4 Lab_4, Edge_4, Act_4, Inv_4, Init_4, \mu, w, \iota_{init})$. Assume that $s_0 = (l_0$,$\{x = 0$,$y = 0\})$ is a state of $\mathcal{H}_4$, μ_{s_0} is uniform distribution over $I(s_0)$ and $\iota_{init}(s_0) = 1$. The weight of each transition is marked with numbers in blue. The probability of starting from s_0 and taking e_3,e_3 in sequence is:*

$$\begin{aligned}
\mathbb{P}_{\mathcal{H}_4}((s_0,e_3e_3)) =& \iota(s_0) \cdot \mathbb{P}_{\mathcal{H}_4}(\widehat{\pi}(s_0,e_3e_3)) \\
=& 1 \cdot \int_{t \in I(s_0,e_3)} p_{s_0+t}(e_3) \cdot \mathbb{P}_{\mathcal{H}_4}(\widehat{\pi}(s_0,e_3) d\mu_{s_0}(t) \\
=& \int_{2.5}^{5} \frac{w(e_3)}{w(e_2)+w(e_3)} \cdot \mathbb{P}_{\mathcal{H}_4}(\widehat{\pi}(s_0,e_3)) d\frac{1}{|I(s_0)|} \cdot t \\
=& \int_{2.5}^{5} \frac{2}{3} \cdot \mathbb{P}_{\mathcal{H}_4}(\widehat{\pi}(s_0,e_3)) d\frac{1}{2.5} \cdot t \\
=& \frac{2}{3} \int_{2.5}^{5} \left(\int_{u \in I(s_0,e_3)} p_{s_0+u}(e_3) \cdot \mathbb{P}_{\mathcal{H}_4}(\widehat{\pi}(s_0)) d\mu_{s_0}(u) \right) d\frac{1}{2.5} \cdot t \\
=& \frac{2}{3} \int_{2.5}^{5} \left(\frac{w(e_3)}{w(e_2)+w(e_3)} \cdot \mathbb{P}_{\mathcal{H}_4}(\widehat{\pi}(s_0)) d\frac{1}{|I(s_0)|} \cdot u \right) d\frac{1}{2.5} \cdot t \\
=& \frac{2}{3} \int_{2.5}^{5} \left(\int_{2.5}^{5} \frac{2}{3} \cdot \mathbb{P}_{\mathcal{H}_4}(\widehat{\pi}(s_0)) d\frac{1}{2.5} \cdot u \right) d\frac{1}{2.5} \cdot t \\
=& \frac{2}{3} \cdot \frac{2}{3} \int_{2.5}^{5} \left(\int_{2.5}^{5} 1 d\frac{1}{2.5} \cdot u \right) d\frac{1}{2.5} \cdot t \\
=& \frac{2}{3} \cdot \frac{2}{3} \int_{2.5}^{5} 1 d\frac{1}{2.5} \cdot t
\end{aligned}$$

$$
\begin{aligned}
&= \frac{2}{3} \cdot \frac{2}{3} \\
&= \frac{4}{9}
\end{aligned}
$$

The probability of starting from s_0 and taking transition e_3 n-times is $\mathbb{P}_{\mathcal{H}_4}((s_0,e_3^n)) = 1 \cdot \left(\frac{2}{3}\right)^n$ with $n \in \mathbb{N}$. If n is infinitely close to positive infinity, then the probability $\mathbb{P}_{\mathcal{A}}(\widehat{\pi}((s_0,e_3^n))$ is infinitely close to zero, which means that the probability of continuously taking transition e_3 infinitely many times is zero.

Chapter 4

Reachability Analysis

4.1 Probability over Finite Path

Given a stochastic rectangular automaton $\mathcal{H} = (Loc, Var, Lab, Edge, Act, Inv, Init, \mu, w, \iota_{init})$ with state set Σ and a finite path $\widehat{\pi}(s_0, e_1 \dots e_n)$. We calculate the probability $\mathbb{P}_{\mathcal{H}}(\widehat{\pi}(s_0, e_1 e_2 \dots e_n))$ as following:

$$\mathbb{P}_{\mathcal{H}}(\widehat{\pi}(s, e_1 e_2 \dots e_n)) = \int_{t \in I(s,e_1)} p_{s+t}(e_1) \cdot \mathbb{P}_{\mathcal{H}}(\widehat{\pi}(s', e_2 \dots e_n)) d\mu_s(t)$$

where $s \xrightarrow{t} s+t \xrightarrow{e} s'$ with $s, s+t, s' \in \Sigma$, $t \geq 0$ and $e \in Edge$, and initially we define $\mathbb{P}_{\mathcal{H}}(\widehat{\pi}(s)) = 1$ for every state $s \in \Sigma$. The most important part is the calculation of interval $I(s,e)$ that may be spent on state s before transition e taken.

4.1.1 Interval Calculation

Intuitively we could get the intersection of current valuation, activities, invariants, guards and resets, then eliminating variables except the time variable. Intersection of current valuation, activities and invariants is the set of all valuations that may be reachable in current location of current state, which is similar to forward time closure.

We define the interval $[(s,e)]$ over time with a state set $S = (l, \phi^{init})$ before taking transition $e = (l, a, (guard, reset), jump, l')$ in location l as:

$$\begin{aligned} t \in [(s,e)] \quad \Leftrightarrow \quad & \exists \nu^{pre}, \nu, \nu^{post} \in V.\ 0 \leq t \wedge \nu^{pre} \models \phi^{init} \wedge \nu = Act_l[\nu^{pre}](t) \wedge Inv_l[\nu] \\ & \wedge guard[\nu] \wedge \nu^{post} = reset[\nu] \wedge Inv_{l'}[\nu^{post}] \end{aligned}$$

Stochastic timed automata is a subclass of stochastic rectangular automata, whose variables are a finite set of clocks. Clocks are a set of variables, that change with rate one and can only be inspected and reset to zero. Thus a clock indicates the amount of time after reseting. As our point of Section 4.1 is to describe the calculation process clearly, our examples in Section 4.1 deal with stochastic timed automata that are relatively simpler than stochastic rectangular automata in terms of calculation.

Example 4.1.1. *Figure 4.1 shows a simple stochastic timed automaton $\mathcal{A}_1$ with a single clock $\{x\}$ and two locations $\{l_0, l_1\}$. The weight of each transition is marked with numbers in blue. We consider transition e_0 and e_1: by taking e_0 clock x will not* *be reseted, however by taking e_1 clock x will be reseted to zero. We compute $I(s_0,e_0)$ and $I(s_0,e_1)$ where s_0 is a state to find how reseting influences the interval calculation result:*

- *the interval $[(s_0,e_0)]$ from state $s_0 = (l_0,\{x = 0\})$ before taking e_0 which does not reset x:*

$$\begin{aligned}[(s_0,e_0)] =& \exists x^{pre}, x, x^{post}.\ 0 \le t \wedge x^{pre} = 0 \wedge x = x^{pre} + t \wedge x \le 10 \\ & \wedge 5 \le x \wedge x^{post} = x \wedge x^{post} \le 8 \\ =& \exists x, x^{post}.\ 0 \le t \wedge x = 0 + t \wedge x \le 10 \\ & \wedge 5 \le x \wedge x^{post} = x \wedge x^{post} \le 8 \\ =& \exists x^{post}.\ 0 \le t \wedge t \le 10 \wedge 5 \le t \wedge x^{post} = t \wedge x^{post} \le 8 \\ =& 0 \le t \wedge 5 \le t \wedge t \le 8 \wedge t \le 10 \\ =& 5 \le t \le 8,\end{aligned}$$

so $I(s_0,e_0) = [5,8]$;

- *the interval $[(s_0,e_1)]$ from state $s_0 = (l_0,\{x = 0\})$ before taking e_1 which resets x:*

$$\begin{aligned}[(s_0,e_0)] =& \exists x^{pre}, x, x^{post}.\ 0 \le t \wedge x^{pre} = 0 \wedge x = x^{pre} + t \wedge x \le 10 \\ & \wedge 5 \le x \wedge x^{post} = 0 \wedge x^{post} \le 8 \\ =& \exists x, x^{post}.\ 0 \le t \wedge x = 0 + t \wedge x \le 10 \\ & \wedge 5 \le x \wedge x^{post} = 0 \wedge x^{post} \le 8 \\ =& \exists x^{post}.\ 0 \le t \wedge t \le 10 \wedge 5 \le t \wedge x^{post} = 0 \wedge x^{post} \le 8 \\ =& 0 \le t \wedge 5 \le t \wedge t \le 10 \wedge 0 \le 8 \\ =& 5 \le t \le 10,\end{aligned}$$

so $I(s_0,e_1) = [5,10]$;

We find that if clock x is reseted by taking transition e_1 to reach location l_1 from location l_0, the restriction Inv_{l_1} on x does not influence the result of interval calculation; otherwise the result depends on Inv_{l_1} on x.

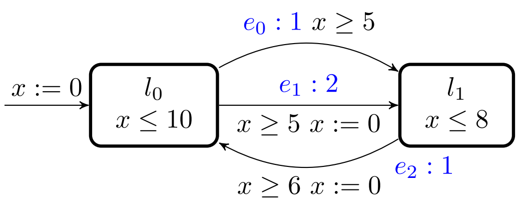


Figure 4.1: A simple stochastic timed automaton $\mathcal{A}_1$

4.1.2 Transition Probability Calculation

Once $I(s,e)$ is determined, the probability distribution over transitions can also be calculates. For each transition e' if the intersection of $I(s,e)$ and $I(s,e')$ is not empty, we say that transition e' is enabled at some time point in $I(s,e)$. We use $I^{\cap}(s,e) = \{I(s,e) \cap I(s,e') \mid \forall e' \in Edge.\ I(s,e) \cap I(s,e') \neq \emptyset\}$ to denote a set of intersections of $I(s,e)$ and $I(s,e')$ with e' enabled at some time point in $I(s,e)$. However we notice that for every possible enabled transitions e', if $I(s,e) \cap I(s,e') \neq I(s,e)$, we need to divided $I(s,e)$ into:

- $I(s,e) \cap I(s,e')$ in which e' is enabled,
- $I(s,e) \backslash (I(s,e) \cap I(s,e'))$ in which e' is not enabled.

Therefore, it is essential to construct a set of intervals for transitions, that each transition is either enabled or not enabled in each interval of the set, rather than enabled in part of the interval. We apply Algorithm 2 to separate the interval $I(s,e)$ into disjoint intervals, such that for each transition and each disjoint interval, transition is either enabled during the interval or not enabled at any time point of the interval. We use $I^{disj}(s,e) = \{I \mid I \subseteq I(s,e)\}$ with $\cup_{I \in I^{disj}} I = I(s,e)$ and $\forall I_1, I_2 \in I^{disj}.\ I_1 \cap I_2 = \emptyset$ to denote resulted disjoint intervals, $E^+(s,I) = \{e' \mid I \subseteq I(s,e')\}$ to denote the set of enabled transitions in interval I, and $I^+(s,e) = \{(I, E^+(s,I)) \mid I \in I^{disj}(s,e)\}$ to denote the pair of interval and set of enabled transition in it.

Algorithm 2: Disjoint Intervals Calculation

```
Input: Set of intervals $I^{\cap}(s,e)$, interval $I(s,e)$
Output: Set of new intervals $I^{disj}$
1  begin
2      $I^{disj} := \emptyset$;
3      $Intervals := I^{\cap}(s,e) \cup I(s,e)$;
4      while $Intervals \neq \emptyset$ do
5          $I := Intervals.front()$;
6          $Intervals.pop()$;
7          $i := Intervals.begin()$;
8          while $i \neq Intervals.end()$ do
9              $tempI := Intervals.at(i)$;
10             $interI := I \cap tempI$;
11             if $interI \neq \emptyset$ then
12                 if $I \backslash interI \neq \emptyset$ then
13                     if $Intervals.find(I \backslash interI) = NULL$ then
14                         $Intervals.push(I \backslash interI)$;
15                 if $tempI \backslash interI \neq \emptyset$ then
16                     if $Intervals.find(tempI \backslash interI) = NULL$ then
17                         $Intervals.push(tempI \backslash interI)$;
18                 $I := interI$;
19                 $Intervals.erase(i)$;
20             $i{+}{+}$;
21         $I^{disj}.push(I)$
```

Example 4.1.2. *Consider the stochastic timed automaton $\mathcal{A}_1$ in Figure 4.1. In Example 4.1.1 we calculate $I(s_0,e_0)$ and (s_0,e_1) with $s_0 = (l_0,\{x = 0\})$, and the results are* [5,8] *and* [5,10] *respectively. We calculate the probability $p_{s_0+t}(e1)$ with $t \in I(s_0,e_1)$ by applying Algorithm 2 we have:*

- $I^{\cap}(s_0,e_1) = \{[5,10],[5,8]\}$,
- $I^{disj}(s_0,e_1) = \{[5,8],(8,10]\}$,
- $E^+(s_0,[5,8]) = \{e_0,e_1\}$ *and* $E^+(s_0,(8,10]) = \{e_1\}$,
- $I^+(s,e) = \{([5,8],\{e_0,e_1\}),((8,10],\{e_1\})\}$,
- $t \in [5,8]$*:* $p_{s_0+t}(e1) = \frac{w(e1)}{\sum_{e\in E^+(s_0,[5,8])} w(e)} = \frac{w(e_1)}{w(e_0)+w(e_1)} = \frac{2}{3}$,
- $t \in (8,10]$*:* $p_{s_0+t}(e1) = \frac{w(e1)}{\sum_{e\in E^+(s_0,(8,10])} w(e)} = \frac{w(e_1)}{w(e_1)} = 1$.

4.1.3 Path Probability Calculation

We refine the calculation of probability $\mathbb{P}_{\mathcal{H}}(\widehat{\pi}(s,e_1e_2\dots e_n))$ as following:

$$\begin{aligned}
&\mathbb{P}_{\mathcal{H}}(\widehat{\pi}(s,e_1e_2\dots e_n))\\
&= \int_{t\in I(s,e_1)} p_{s+t}(e_1)\cdot \mathbb{P}_{\mathcal{H}}(\widehat{\pi}(s',e_2\dots e_n))d\mu_s(t)\\
&= \sum_{(I',E^+(s,I'))\in I^+(s,e_1)} \left(\int_{t\in I'} p_{s+t}(e_1)\cdot \mathbb{P}_{\mathcal{H}}(\widehat{\pi}(s',e_2\dots e_n))d\mu_s(t)\right)\\
&= \sum_{(I',E^+(s,I'))\in I^+(s,e_1)} \left(\int_{t\in I'} \frac{w(e_1)}{\sum_{e_1'\in E^+(s,I')} w(e_1')}\cdot \mathbb{P}_{\mathcal{H}}(\widehat{\pi}(s',e_2\dots e_n))d\mu_s(t)\right)
\end{aligned}$$

where $s \xrightarrow{t} s+t \xrightarrow{e} s'$ with $s,s+t,s' \in \Sigma$, $t \geq 0$ and $e \in Edge$, and initially we define $\mathbb{P}_{\mathcal{H}}(\widehat{\pi}(s)) = 1$ for every state s.

Example 4.1.3. *Consider the stochastic timed automaton $\mathcal{A}_1$ in Figure 4.1. In Example 4.1.2 we have $I^+(s,e) = \{([5,8],\{e_0,e_1\}),((8,10],\{e_1\})\}$. Intuitively there are two disjoint interval sets, which are $I_0 = [5,8]$ and $I_1 = (8,10]$. Assume that $s_0 = (l_0,\{x = 0\})$ is a state of $\mathcal{A}_1$, μ_{s_0} is uniform distribution over $I(s_0)$ and $\iota_{init}(s_0) = 1$. The weight of each transition is marked with numbers in blue in Figure 4.1. The probability of starting from s_0 and taking e_1 is:*

$$\begin{aligned}
\mathbb{P}_{\mathcal{A}_1}((s_0,e_1)) =&\iota(s_0)\cdot \mathbb{P}_{\mathcal{A}_1}(\widehat{\pi}(s_0,e_1))\\
=&1\cdot \int_{t\in I(s_0,e_1)} p_{s_0+t}(e_1)\cdot \mathbb{P}_{\mathcal{A}_1}(\widehat{\pi}(s_0)d\mu_{s_0}(t)\\
=&\int_{t\in I_0} p_{s_0+t}(e_1)\cdot \mathbb{P}_{\mathcal{A}_1}(\widehat{\pi}(s_0)d\mu_{s_0}(t)\\
&+\int_{t\in I_1} p_{s_0+t}(e_1)\cdot \mathbb{P}_{\mathcal{A}_1}(\widehat{\pi}(s_0)d\mu_{s_0}(t)\\
=&\int_5^8 \frac{w(e_1)}{w(e_0)+w(e_1)}\cdot \mathbb{P}_{\mathcal{A}_1}(\widehat{\pi}(s_0))d\frac{1}{|I(s_0)|}\cdot t
\end{aligned}$$

$$
\begin{aligned}
&+\int_{8}^{10}\frac{w(e_1)}{w(e_1)}\cdot\mathbb{P}_{\mathcal{A}_1}(\widehat{\pi}(s_0))d\frac{1}{|I(s_0)|}\cdot t\\
=&\int_{5}^{8}\frac{2}{3}\cdot\mathbb{P}_{\mathcal{A}_1}(\widehat{\pi}(s_0))d\frac{1}{5}\cdot t+\int_{8}^{10}1\cdot\mathbb{P}_{\mathcal{A}_1}(\widehat{\pi}(s_0))d\frac{1}{5}\cdot t\\
=&\int_{5}^{8}\frac{2}{3}\cdot 1d\frac{1}{5}\cdot t+\int_{8}^{10}1\cdot 1d\frac{1}{5}\cdot t\\
=&\frac{2}{3}\cdot\frac{3}{5}+\frac{2}{5}\\
=&\frac{12}{15}
\end{aligned}
$$

4.2 Forward Analysis for Probabilistic Rectangular Automata

Intuitively our analysis is based on the idea of computation tree. The nodes of the tree denote every reachable state set. For each state set s, if s' is reachable from s by taking transition e, we say that s' is a child of s. A path from node s to s' of the tree denote a transition $s \xrightarrow{t,e} s'$ with state sets s,s', time $t \geq 0$, and transition e. Figure 4.2 illustrates how the computation tree constructed (bad states are marked in red). The main difficulties during our implementation is to solve the following situations in Figure 4.2 properly:

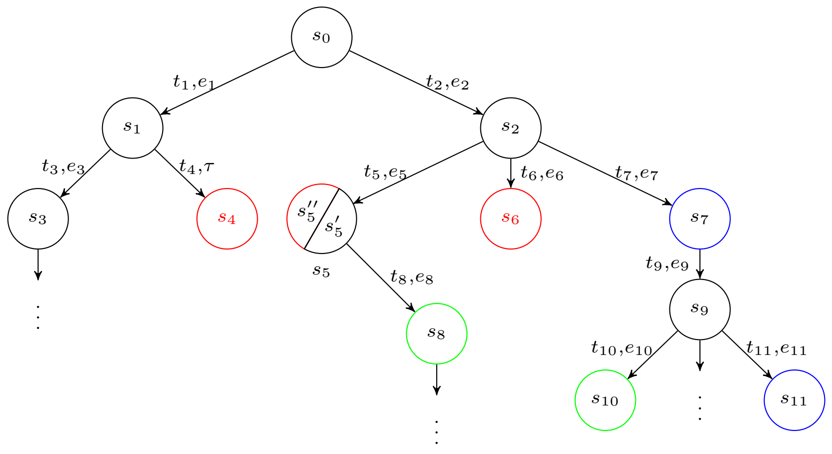


Figure 4.2: A forward analysis computation tree

1. Time-step successor contains bad states: after time t_4 and taking τ-transition from s_1 a bad state s_4 is reached; As τ-transition does not change the location, s_4 stays in the location of s_1; we need not to calculate the successors of s_4.

2. Jump-step successors contain bad states:

   - transition $s_2 \xrightarrow{t_6,e_6} s_6$ is simple to deal with. As s_6 is a bad state, we simply ignore all successors of s_6.

- for transition $s_2 \xrightarrow{t_5,e_5} s_5$ we notice that only part of s_5 is bad. Before doing further calculation we separate s_5 into s_5' and s_5'' with $s_5' \cup s_5'' = s_5$ and $s_5' \cap s_5'' = \emptyset$, where s_5'' is the intersection of s_5 and bad states. Therefore we simplify the calculation by ignoring all successors of s_5''.

3. Successors have been visited before:

   - s_{11} is one of the successors of s_9, besides s_{11} equals to s_7 and s_7 is one of the predecessors of s_9; In other words, if there exists a loop along the path, we only need to calculate every states of the loop once.
   - s_{10} is one of the successors of s_9 and s_{10} equals to s_8, however there is no such path that $s_8 \rightarrow^* s_9$; To solve this problem we construct an unbounded queue to store visited states, therefore if s_8 is visited we need not to do calculation on s_{10} which is the successor of s_9.

4.2.1 Implementation

The basic idea of step successor calculation is depicted in Algorithm 3. Time-step successor and jump-step successor are calculated in sequence. Line 4 to 9 and Line 12 to 15 describe how we deal with the situation if the time-step successor s_t contains bad states and the jump-step successor s' contains bad states respectively. We separate the successors into two parts, one called s_{bad} is the intersection of s_t and bad states. We do not need to do further calculation on s_{bad}.

Algorithm 3: Forward Reachability

```
   Input: State: $s$, bad states: $Bad$
   Output: Set of pairs $Results = \{(safe,(e,s'))\}$
1  begin
2      $Results := \emptyset$;
3      calculate time-step successor $s_t$ according to $s$;
4      if $s_t \cap Bad \neq \emptyset$ then
5          $s_{bad} := s_t \cap Bad$;
6          $e := \tau$;
7          $s' := s_{bad}$;
8          $Results.push((false,(e,s')))$;
9          $s_t := s_t \backslash s_{bad}$;
10     calculate jump-step successors $JumpSucc$ according to $s_t$;
11     for $(e,s') \in JumpSucc$ do
12         if $s' \cap Bad \neq \emptyset$ then
13             $s_{bad} := s' \cap Bad$;
14             $Results.push((false,(e,s_{bad})))$;
15             $s' := s' \backslash s_{bad}$;
16         $Results.push((true,(e,s')))$;
```

The significant calculation is how to get $S \backslash S_{bad}$ where $S = (l,\phi)$, $S_{bad} = (l,\phi_{bad}) \in \Sigma$ and S_{bad} is bad state set. Assume that $\phi_{bad} = a_0 \wedge a_1 \wedge \ldots \wedge a_n$ with $n \in \mathbb{N}$, in order to calculate $S \backslash S_{bad}$ we need to calculate $\phi \backslash \phi_{bad}$. The problem is that $\phi \wedge \neg \phi_{bad}$ may contain disjunctions, which we need to avoid. The idea of eliminating disjunctions is to separate $\phi \backslash \phi_{bad}$ into subsets $\phi_0 = \{\phi \wedge \neg a_0\}$, $\phi_1 = \{\phi \wedge a_0 \wedge \neg a_1\}$, $\ldots$, $\phi_n = \{\phi \wedge \ldots \wedge a_{n-1} \wedge \neg a_n\}$. Thus we get a set of states $S_0 = (l,\phi_0)$, $S_1 = (l,\phi_1)$,…,$S_n = (l,\phi_n)$. The successor calculation is then applied on S_0, …, S_n respectively.

The forward analysis is depicted in Algorithm 4. The forward analysis starts from initial states, and we use breadth-first search to implement the tree construction. For each unprocessed state we use forward reachability calculation described above to get the corresponding transitions and successors. Then we add the successors of processed state as a child of current node. If the successor is processed before we avoid to process second time which is indicated in Line 18 to 19. At the end we collect all reachable bad states.

Algorithm 4: Forward Analysis

```
   Input: Hybrid Automaton $HA$
   Output: Set of unsafe states: $UnsafeStates$, reach tree: $ReachTree$
1  begin
2      $Init$ is the set of initial states of $HA$;
3      $Bad$ is the set of bad states of $HA$;
4      $Reached = ToProcess := \emptyset$;
5      $ReachTree := \emptyset$;
6      $UnsafeStates := \emptyset$;
7      for $s_0 \in Init$ do
8          $ToProcess.push(s_0)$;
9          add $s_0$ as a root of the tree $ReachTree$;
10     while $!ToProcess.empty()$ do
11         $s := ToProcess.front()$;
12         $ToProcess.pop()$;
13         $Reached.push(s)$;
14         $Results := ForwardReachability(s,Bad)$;
15         for $(safe, (e,s')) \in Results$ do
16             add $s'$ as a child of $s$ in the tree $ReachTree$;
17             if $safe$ then
18                 if $s' \notin Reached$ then
19                     $ToProcess.push(s')$;
20             else
21                 $UnsafeStates.push(s')$ ;
```

4.3 Backward Analysis and Probability Calculation for Probabilistic Rectangular Automata

After collecting unsafe states, we could do backward analysis as well as probability calculation. Figure 4.3 illustrates how backward analysis works based on computation tree. Figure 4.3(a) depicts a computation tree constructed by forward analysis, there are two unsafe paths $\pi_1 = s_0 \xrightarrow{t_1,e_1} s_1 \xrightarrow{t_4,e_4} s_4 \xrightarrow{t_7,e_7} s_7$ and $\pi_2 = s_0 \xrightarrow{t_3,e_3} s_3 \xrightarrow{t_6,e_6} s_6$, among which π_1 is the longest. Figure 4.3(b) depicts that we get s'_4 by one step predecessor calculation on longest path π_1. If there are more than one paths which are longest, we calculate one step predecessors of the longest paths at the same level. Figure 4.3(c) depicts that both $\pi'_1 = s_0 \xrightarrow{t_1,e_1} s_1 \xrightarrow{t_4,e_4} s'_4$ and π_2 are longest paths, we process them in sequence in this step. Figure 4.3(d) depicts that s'_1 of $\pi''_1 = s_0 \xrightarrow{t_1,e_1} s'_1$ and s'_3 of $\pi'_2 = s_0 \xrightarrow{t_3,e_3} s'_3$ have the same parent, the predecessor s'_0 is dependent not only on s'_1 but also on s'_3.

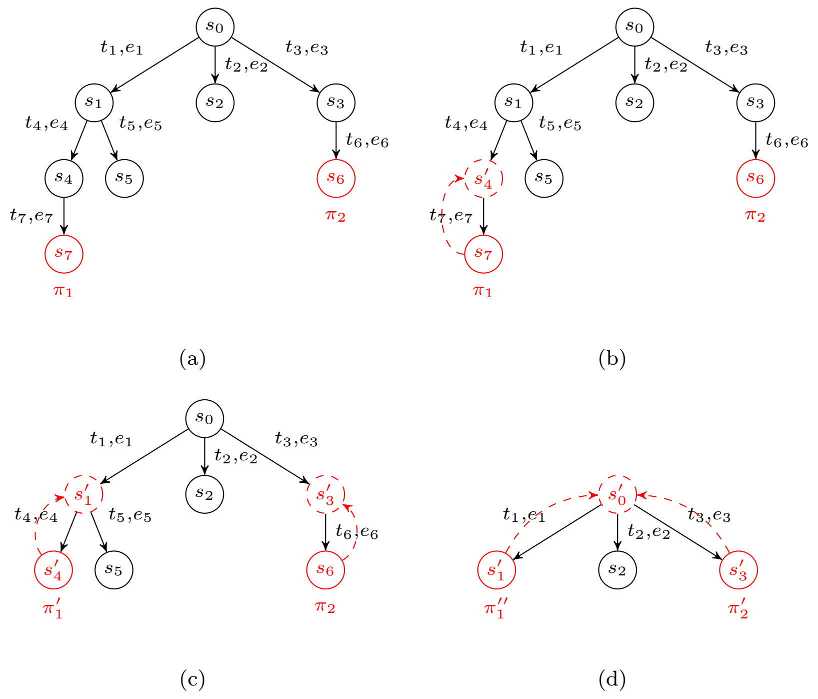


Figure 4.3: A simple backward analysis based on computation tree

4.3.1 Implementation

Algorithm 5 illustrates the backward analysis along with probability calculation. We first find a longest path and bad state s_{bad} which is indicated in Line 6. Line 17 to Line 20 depict that the probability of the state is dependent on all its children.

4.4 Direct Backward Analysis for Probabilistic Rectangular Automata

In Section 4.2 and Section 4.3 we propose a reachability analysis approach by first going through all reachable states by forward analysis and then calculating the probability from reached bad states by backward analysis. This approach gives a complete

```
Algorithm 5: Backward Analysis and Probability Calculation
  Input: Hybrid Automaton: $HA$, set of unsafe states: $UnsafeStates$, tree
         constructed after forward analysis: $ReachTree$
  Output: The probability to reach bad states: $prob_{unsafe}$
1 begin
2     $\iota_{init}$ is a initial distribution of $HA$;
3     $prob_{unsafe} := 0$;
4     $prob_s := ones()$;
5     while $!UnsafeStates.empty()$ do
6         $s_{bad}$ is the state whose path from initial state is the longest;
7         $treeNode$ is corresponding node of $s_{bad}$;
8         while $treeNode \to parent() \; ! = NULL$ do
9             $e_{bad}$ is the transition to reach $treeNode$ from the parent node
                $treeNode \to parent()$;
10            calculate jump-step predecessor $s_t$ according to $s_{bad}$ and $e_{bad}$;
11            calculate time-step predecessor $s_{bad}^{pre}$ according to $s_t$;
12            $s^{pre}$ is the state of the parent node $treeNode \to parent()$;
13            $s := s_{bad}^{pre} \cap s^{pre}$;
14            if $e_{bad} = \tau$ then
15                $prob_s[s] := prob_s[s_{bad}]$;
16            else
17                if $prob_s[s] = 1$ then
18                    $prob_s[s] := \int_{t \in I(s,e_{bad})} p_{s+t}(e_{bad}) \cdot prob_s[s_{bad}] d\mu_s(t)$;
19                else
20                    $prob_s[s] :=$
                        $prob_s[s] + \int_{t \in I(s,e_{bad})} p_{s+t}(e_{bad}) \cdot prob_s[s_{bad}] d\mu_s(t)$;
21            $s_{bad} := s$;
22            $treeNode := treeNode \to parent()$;
23        $prob_{unsafe} := prob_{unsafe} + prob_s[s] \cdot \iota_{init}(s)$
```

view of reachable states. However, sometimes we do not need to collect all informations of reachable states. Thus we consider to do backward analysis directly. The idea of backward analysis is also tree based, but bottom to top. The construction of the tree is similar to that in Section 4.2, we do not repeat it. We introduce the implementation directly.

Algorithm 6 shows the basic idea of backward reachability calculation, comparing to forward reachability calculation in Algorithm 4 we start from bad state to calculate its predecessors and once initial states are reached we do not calculate the predecessors any more.

Algorithm 7 illustrate the tree construction and probability calculation by applying backward analysis directly.

Algorithm 6: Backward Reachability

```
Input: Bad state: $s_{bad}$, initial states $Initial$
Output: Set of pairs $Results = \{(initial,(s',e))\}$
1  begin
2      $Results := \emptyset$;
3      calculate time-step predecessor $s_t$ according to $s_{bad}$;
4      if $s_t \cap Initial \neq \emptyset$ then
5          $s_{init} := s_t \cap Intial$;
6          $e := \tau$;
7          $Results.push((true,(s_{init},e))$;
8          $s_t := s_t \setminus s_{init}$;
9      calculate jump-step successors $JumpSucc$ according to $s_t$;
10     for $(s',e) \in JumpSucc$ do
11         if $s' \cap Initial \neq \emptyset$ then
12             $s_{init} := s' \cap Initial$;
13             $Results.push((true,(s_{init},e))$;
14             $s' := s' \setminus s_{init}$;
15         $Results.push((false,(s',e)))$;
```

Algorithm 7: Backward Analysis

```
Input: Hybrid Automaton: $HA$
Output: Set of unsafe states: $UnsafeStates$, reach tree: $ReachTree$
begin
    $Init$ is the set of initial states of $HA$;
    $Bad$ is the set of bad states of $HA$;
    $Reached = ToProcess := \emptyset$;
    $ReachTree := \emptyset$;
    $prob_{unsafe} := 0$;
    $prob_s := ones()$;
    for $s_{bad} \in Bad$ do
        $ToProcess.push(s_{bad})$;
        add $s_{bad}$ as node the tree $ReachTree$;
    while $!ToProcess.empty()$ do
        $s_{bad} := ToProcess.front()$;
        $ToProcess.pop()$;
        $Reached.push(s_{bad})$;
        $Results := BackReachability(s_{bad},Init)$;
        for $(initial, (s',e)) \in Results$ do
            add $s'$ as a parent of $s_{bad}$ in the tree $ReachTree$;
            if $e = \tau$ then
                $prob_s[s'] := prob_s[s_{bad}]$;
            else
                if $prob_s[s'] = 1$ then
                    $prob_s[s'] := \int_{t \in I(s',e)} p_{s+t}(e) \cdot prob_s[s_{bad}] d\mu_{s'}(t)$;
                else
                    $prob_s[s'] := prob_s[s'] + \int_{t \in I(s',e)} p_{s+t}(e) \cdot prob_s[s_{bad}] d\mu_{s'}(t)$;
            if $!initial$ then
                if $s' \notin Reached$ then
                    $ToProcess.push(s')$;
            else
                $prob_{unsafe} := prob_{unsafe} + prob_s[s'] \cdot \iota_{init}(s)$
```

Chapter 5

Results

The basic idea of reachability analysis on stochastic rectangular automata is presented in previous chapter. The library our implementation based is called HyPro [SAMK17]. HyPro provides the implementation of state set representations for hybrid automata.

5.1 Simple Test Example

In oder to give a brief idea of how the program runs, we introduce a simple test example firstly. Figure 5.1 shows a stochastic rectangular automata with two locations l_0, l_1 and one variables x. Location l_0 has two out-going transitions e_0, e_1 to reach l_1, l_0 respectively, and location l_1 has one out-going transition e_2 to reach l_0.

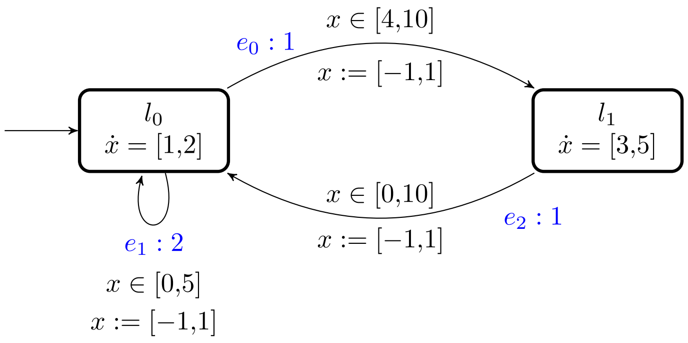
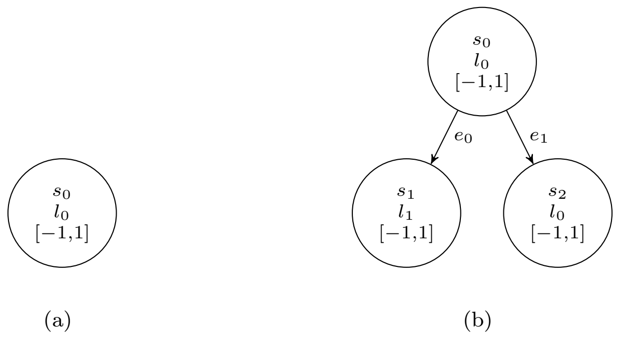


Figure 5.1: A simple stochastic rectangular automaton $\mathcal{H}_1$

5.1.1 Reachable Bad State Analysis

Assume that the initial state is $s_0 = (l_0,\{x \in [-1,1]\})$ and the bad state is $s_{bad} = (l_1,\{x \in [6,8]\})$. We apply the forward analysis introduced in Section 4.2 to construct a reach tree. Figure 5.2 shows the construction of a reach tree by applying forward analysis. To minimize the figure we eliminate the constraints which do not effect the variable value. For example, assume that the constraint set is $\{[0,1],[-1,7]\}$, then we only use {[0,1]} in the following reach tree figure. Figure 5.2(a) shows that we add the initial state s_0 as the root of the tree. In Figure 5.2(b) we add jump successors s_1, s_2 as child nodes of s_0. As $s_2 = s_0$ we mark s_2 as a processed node and do not calculate further successor of s_2 any more. Figure 5.2(c) shows that after time-successor calculation the bad state s_3 is reached. We do not need to calculate the successors of bad state s_3. As mentioned in Section 4.2.1 we separate the state set, which does not contain bad state, into substate sets to avoid disjunction. s_4, s_5 in Figure 5.2(c) are two substates after separation, and the jump-successors of s_4, s_5 are s_6, s_7. For the reason that $s_6 = s_0$ and $s_7 = s_0$ we mark s_6 and s_7 as processed nodes. Thus forward analysis is finished.



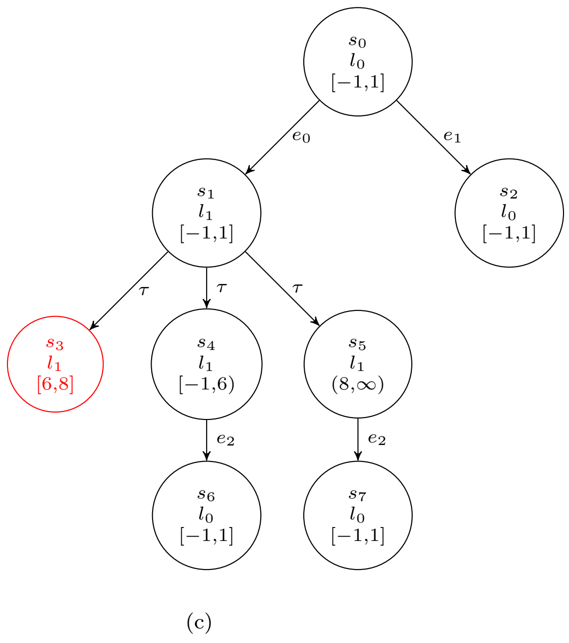


Figure 5.2: Computation tree after forward analysis

After the reach tree construction we apply backward analysis and calculate the probability at the same time. Figure 5.4 shows that we apply backward analysis starting from bad state s_3. In Figure 5.3(a) we do backward reachability analysis one step back. We get the predecessor s_1' and the probability from s_1' to reach s_3 is $\frac{2}{3}$. We do one more step backward reachability analysis and get the predecessor s_0 in Figure 5.3(b), which is the initial state. The probability of reaching s_3 from s_0 through s_1' is $\frac{13}{33}$

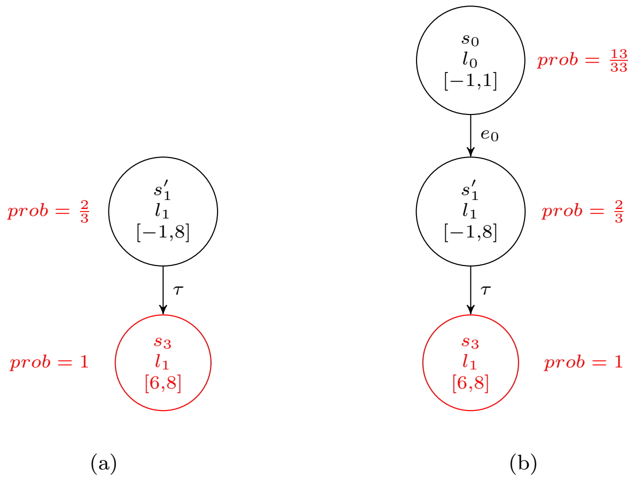


Figure 5.3: Computation tree after backward analysis

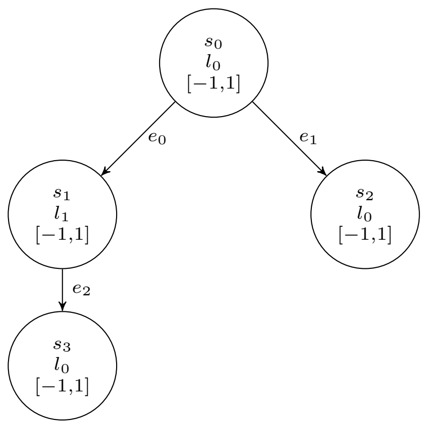
If we use direct backward analysis introduced in Section 4.4, we get a similar reach tree as shown in Figure 5.4. The only difference is that s_1' by applying direct backward analysis is (l_1,(∞,8]), because by applying direct backward analysis we do not have any restrictions of predecessors, which is given by forward analysis when we apply first forward analysis and then backward analysis.

5.1.2 Not Reachable Bad State Analysis

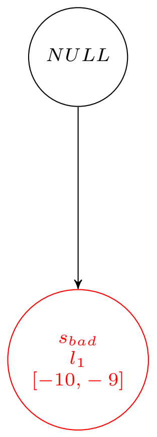
Assume that the initial state is $s_0 = (l_0,\{x \in [-1,1]\})$ and the bad state is $s_{bad} = (l_1,\{x \in [-10,-9]\})$. We first apply the forward analysis introduced in Section 4.2 to construct a reach tree. We get a reach tree shown in Figure 5.4(a). The initial state s_0 is the root of the tree, and after traveling through all reachable state we find the bad state s_{bad} is not reachable. There is no need to do backward analysis, since the bad state is not reachable. Then we apply direct backward analysis introduced in Section 4.4, the reach tree is shown in Figure 5.4(b). Instead of traveling through all reachable states, we only do one step backward reachability analysis to find that the bad state s_{bad} is not reachable.

5.1.3 Comparing the Two Approaches

Section 5.1.1 shows that the forward analysis offers more execution information of the process, which may reduce a part of the calculation when we do backward analysis. Section 5.1.2 shows that if only information we need to know is the probability to reach bad states, then apply direct backward analysis may be a better approach. Thus for different scenarios we could choose one proper approach to apply. The basic



(a)



(b)

Figure 5.4: Computation tree after backward analysis

principle is that if we only need to analysis the reachability of bad states, we apply direct backward analysis , otherwise first apply forward analysis and then backward analysis.

5.2 A Two Room Shared Heater Example

UPPAAL [upp] is a tool to model, validate and verificate the real-time systems. Some of the case studies modeled in UPPAAL are hybrid automata, which we may use as benchmarks after some adaption. A two room shared heater [DLL+15] is a Statistical Model Checking(SMC) example which can be analyzed by UPPAAL. In this example, we consider a heater which is shared by two independent rooms. At any time at most one room can be heated. The model shown in [DLL+15] is a set of parallel hybrid automata, the first step of adaption is to build the parallel composition of hybrid automata.

As committed location in UPPAAL does not effect the variable values, we build the parallel composition of hybrid automata by ignoring committed location. We get the automata shown in Figure 5.5. As our purpose is to do reachability analysis, we eliminate the locations which are not reachable from the initial location and get the automata shown in Figure 5.6. Another significant adaption is to modify the non-rectangular labeling functions in [DLL+15] to labeling functions in rectangular regions. We choose proper rectangular regions to simulate the temperature changing. The final stochastic rectangular automata is shown in Figure 5.7, and for simplification we use OFF, ON_{r_0}, ON_{r_1} to represent the location (OFF, OFF_{r_0}, OFF_{r_1}), (ON_0, ON_{r_0}, OFF_{r_1}), (ON_1, OFF_{r_0}, ON_{r_1}) in Figure 5.6 respectively. Assume that the initial state is (OFF,$\{x = 0$,$T_0 \in [20,25]$,$T_1 \in [20,25]\}$), the probability that the room r_0 is heated to $T_0 = 70$ by switch on the heater once time is 0.000030, close to zero. The constructed reach tree is shown in Figure 5.8.

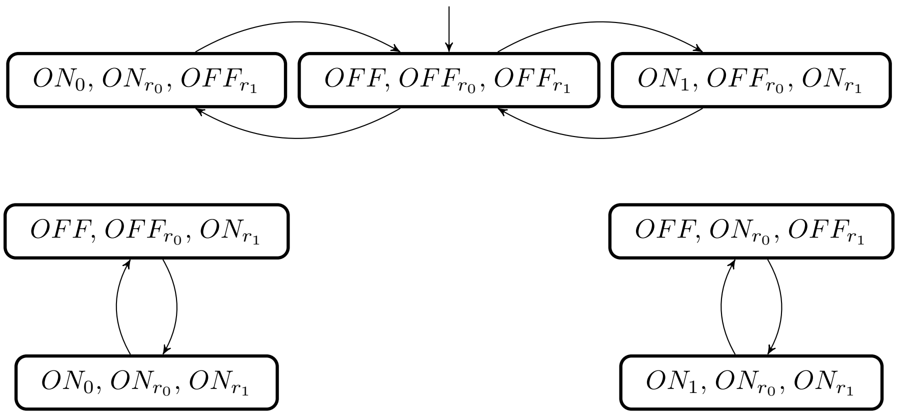

$ON_0, ON_{r_0}, OFF_{r_1}$
$OFF, OFF_{r_0}, OFF_{r_1}$
$ON_1, OFF_{r_0}, ON_{r_1}$
OFF, OFF_{r_0}, ON_{r_1}
OFF, ON_{r_0}, OFF_{r_1}
ON_0, ON_{r_0}, ON_{r_1}
ON_1, ON_{r_0}, ON_{r_1}

Figure 5.5: The parallel composition of hybrid automata of a two room shared heater example.

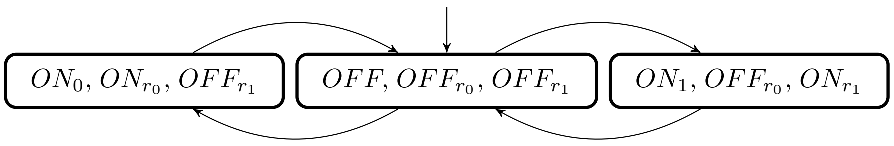

$ON_0, ON_{r_0}, OFF_{r_1}$
$OFF, OFF_{r_0}, OFF_{r_1}$
$ON_1, OFF_{r_0}, ON_{r_1}$

Figure 5.6: The parallel composition of hybrid automata of a two room shared heater example after eliminate unreachable locations.

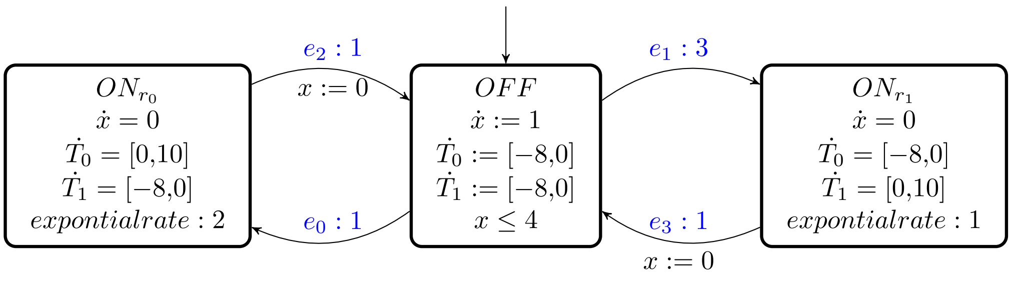


Figure 5.7: The stochastic rectangular automata of a two room shared heater example.

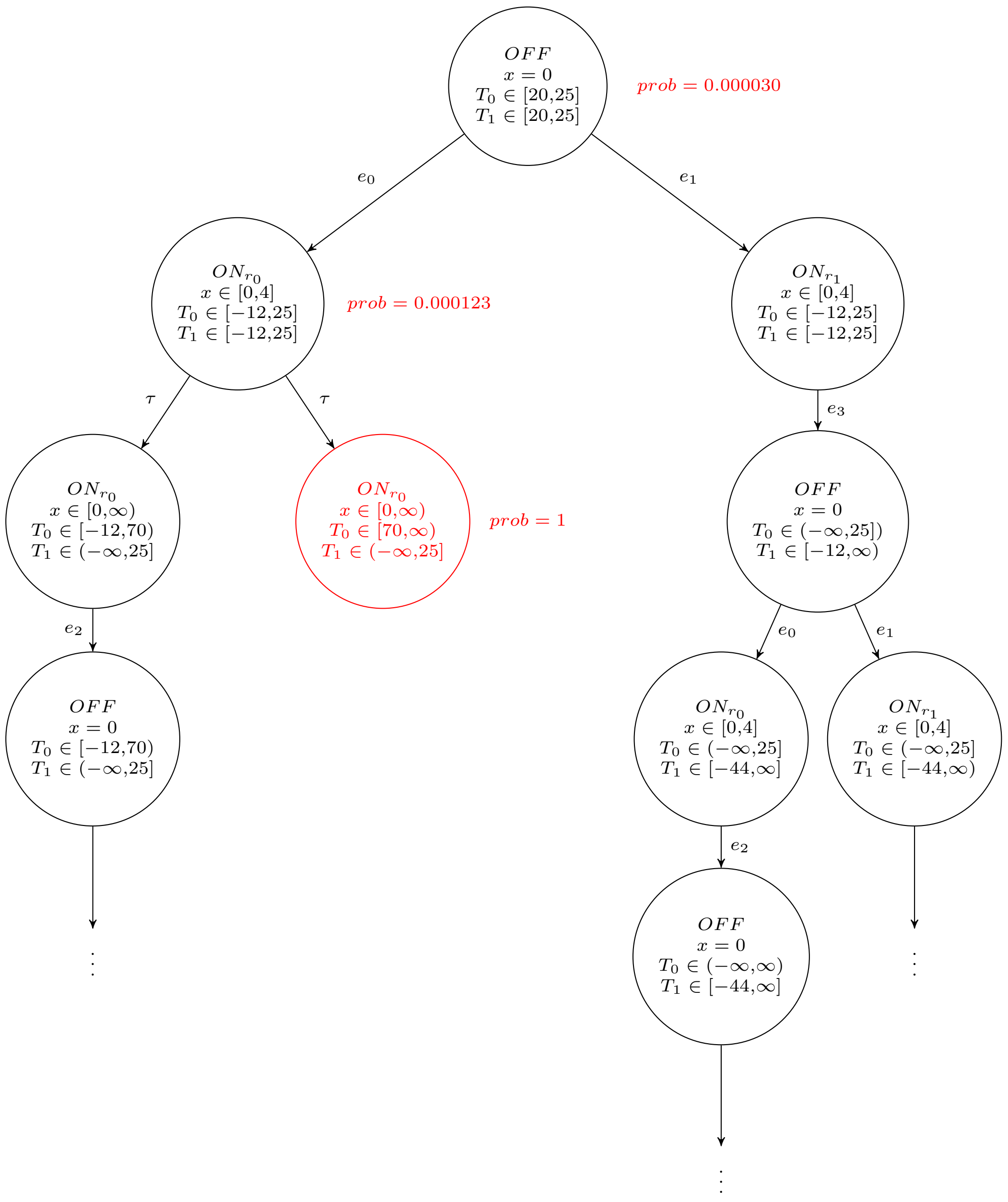

OFF, $x = 0$, $T_0 \in [20,25]$, $T_1 \in [20,25]$
$prob = 0.000030$
e_0
e_1
ON_{r_0}, $x \in [0,4]$, $T_0 \in [-12,25]$, $T_1 \in [-12,25]$
$prob = 0.000123$
ON_{r_1}, $x \in [0,4]$, $T_0 \in [-12,25]$, $T_1 \in [-12,25]$
τ
τ
e_3
ON_{r_0}, $x \in [0,\infty)$, $T_0 \in [-12,70)$, $T_1 \in (-\infty,25]$
ON_{r_0}, $x \in [0,\infty)$, $T_0 \in [70,\infty)$, $T_1 \in (-\infty,25]$
$prob = 1$
OFF, $x = 0$, $T_0 \in (-\infty,25])$, $T_1 \in [-12,\infty)$
e_2
e_0
e_1
OFF, $x = 0$, $T_0 \in [-12,70)$, $T_1 \in (-\infty,25]$
ON_{r_0}, $x \in [0,4]$, $T_0 \in (-\infty,25]$, $T_1 \in [-44,\infty]$
ON_{r_1}, $x \in [0,4]$, $T_0 \in (-\infty,25]$, $T_1 \in [-44,\infty)$
e_2
OFF, $x = 0$, $T_0 \in (-\infty,\infty)$, $T_1 \in [-44,\infty]$

Figure 5.8: The tree constructed after forward analysis for a two room shared heater example.

Chapter 6

Conclusion

6.1 Summary

In this thesis we introduce the reachability analysis for stochastic rectangular automata. Stochastic rectangular automata, which is an extension of stochastic timed automata, is a kind of probabilistic rectangular automata with continuous-time probability distribution. The probability distribution of stochastic rectangular automata has two parts, one is over delays and the other is over transitions. The probability distribution over delays is continuous. The probability distribution over transitions is dependent on enabled transition in a given time interval.

The most important part of reachability analysis implementation is time interval calculation. For a state s, the time interval before taking transition e to s' is determined by eliminating the variables expect time in intersection of the states valuation, activities, invariants, guards and resets. Then the probability calculation is based on the time intervals.

One approach to implement reachability analysis is to do forward and backward analysis in sequence. Forward analysis is to travel through all reachable states by breadth-first search, after that it builds a computation tree for processed states. According to the reach tree constructed by forward analysis we do backward analysis from reachable bad states. For each step we calculate the predecessors of the current state and the probability of moving from each predecessor to current state respectively. Once the root of the reach tree is arrived, the backward analysis is completed and we get the probability of starting from initial state to reach at least one bad state.

Another approach to implement reachability analysis is to do backward analysis directly. Starting from one of the bad states, we do backward reachability calculation step by step, in the meanwhile calculate the probability of reaching a bad state from currently processing state. Once the predecessor is marked as processed states, we repeat backward reachability calculation from one of unprocessed bad states. Until all bad states are marked as a processed states, we sum up the probability of moving from the initial states to bad states.

The two approaches are applied in different scenarios. In Chapter 5 we compare the two approaches by applying them for the same example. None of them is absolutely efficient than the other. The basic principle is that if we need the information of all reachable states, then the first approach is a proper way, otherwise we choose the second approach.

6.2 Future work

Probabilistic Rectangular Automata with pure continuous-time probability distribution. Stochastic rectangular automata introduced in this paper has two parts of probability distribution. The probability distribution over transitions is dependent on the weights of enabled transitions in a given time interval, which is still discrete-time probability distribution. By adapting the probability distribution over transitions to a continuous-time probability distribution, we would introduce a probabilistic Rectangular Automata with pure continuous-time probability distribution, which is more efficient to model the systems in our daily life.

Probabilistic Hybrid Automata with continuous-time probability distribution. Rectangular automata is a subclass of hybrid automata. The labeling function of rectangular automata is restricted to rectangular regions. However, in oder to model a more complicated systems we need a more expressible model. In the future, we could extend the probabilistic rectangular automata first to probabilistic linear hybrid automata and then to probabilistic general hybrid automata. In which case, the valuation of variables is not only dependent on time but also on the value of variables.

Bibliography

[ACH$^+$95] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical computer science*, 138(1):3–34, 1995.

[ACHH92] Rajeev Alur, Costas Courcoubetis, Thomas A Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid systems*, pages 209–229. Springer, 1992.

[AD94] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[BBB$^+$14] Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, Quentin Menet, Christel Baier, Marcus Größer, and Marcin Jurdzinski. Stochastic timed automata. *arXiv preprint arXiv:1410.2128*, 2014.

[BDG$^+$11] Thomas Brihaye, Laurent Doyen, Gilles Geeraerts, Joël Ouaknine, Jean-François Raskin, and James Worrell. On reachability for hybrid automata over bounded time. In *International Colloquium on Automata, Languages, and Programming*, pages 416–427. Springer, 2011.

[BK02] Falko Bause and Pieter S Kritzinger. *Stochastic petri nets*, volume 1. Citeseer, 2002.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[Bra05] Michael S. Branicky. *Introduction to Hybrid Systems*, pages 91–116. Birkhäuser Boston, Boston, MA, 2005.

[DLL$^+$15] Alexandre David, Kim G Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. Uppaal smc tutorial. *International Journal on Software Tools for Technology Transfer*, 17(4):397–415, 2015.

[HKPV98] Thomas A Henzinger, Peter W Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *Journal of computer and system sciences*, 57(1):94–124, 1998.

[HM00] Thomas A Henzinger and Rupak Majumdar. Symbolic model checking for rectangular hybrid systems. In *International Conference on Tools and*

Algorithms for the Construction and Analysis of Systems, pages 142–156. Springer, 2000.

[HMU01] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1):60–65, 2001.

[JELS99] Karl Henrik Johansson, Magnus Egerstedt, John Lygeros, and Shankar Sastry. On the regularization of zeno hybrid automata. *Systems & control letters*, 38(3):141–150, 1999.

[KNS02] Marta Kwiatkowska, Gethin Norman, and Jeremy Sproston. Probabilistic model checking of the ieee 802.11 wireless local area network protocol. In *Joint International Workshop von Process Algebra and Probabilistic Methods, Performance Modeling and Verification*, pages 169–187. Springer, 2002.

[Kop96] Peter W Kopke. The theory of rectangular hybrid automata. Technical report, Cornell University, 1996.

[PDMV15] Pavithra Prabhakar, Parasara Sridhar Duggirala, Sayan Mitra, and Mahesh Viswanathan. Hybrid automata-based cegar for rectangular hybrid systems. *Formal Methods in System Design*, 46(2):105–134, 2015.

[SAMK17] Stefan Schupp, Erika Abraham, Ibtissem Ben Makhlouf, and Stefan Kowalewski. H y p ro: A c++ library of state set representations for hybrid systems reachability analysis. In *NASA Formal Methods Symposium*, pages 288–294. Springer, 2017.

[SFÁ19] Stefan Schupp, Goran Frehse, and Erika Ábrahám. State set representations and their usage in the reachability analysis of hybrid systems. Technical report, Fachgruppe Informatik, 2019.

[Spr11] Jeremy Sproston. Discrete-time verification and control for probabilistic rectangular hybrid automata. In *2011 Eighth International Conference on Quantitative Evaluation of SysTems*, pages 79–88. IEEE, 2011.

[upp] Uppaal. `https://uppaal.org`.