Non-Deterministic Generalised Stochastic Petri Nets Modelling and Analysis

Rob Bamberg at the University of Twente
r.j.bamberg@student.utwente.nl

November 14, 2012

Abstract

Generalized Stochastic Petri Nets (GSPNs) [28] are a well-known modelling formalism to compute dependability and performance of distributed systems. GSPNs are translated to Continuous Time Markov Chains (CTMCs) [34] for their analysis, which means they can not support non-determinism. Conflicts and confusion are resolved by specifying priorities and weights for each transition, which is not desirable in all cases.

We define a more expressive and general Non-Deterministic GSPN (ND-GSPN). ND-GSPNs support the weights and priorities and additionally introduce the use of non-determinism. This means conflicts between transitions do not necessarily have to be resolved any more. ND-GSPNs also include reset, weighted inhibitor and probabilistic arcs. Probabilistic arcs make it is possible to define probability more local, because they make the result of a transition probabilistically distributed. We express the ND-GSPN in terms of the more general Markov Automaton (MA) [13]. An MA combines probability, exponentially distributed leave rates and non-determinism and is therefore suited to express the semantics of ND-GSPNs.

The Petri Net Markup Language (PNML) [45] is an ISO/IEC 15909 standard, which can be used to specify Petri Nets. We define how the PNML for ND-GSPNs should look like. MAs can be specified in the Markov Automaton Process Algebra (MAPA) [41], which is used as input for the tool SCOOP [40]. SCOOP automatically reduces its models and can export the state space directly to a model-checker. The IMCA [2] model-checker can be used to check an MA for time-bounded reachability, unbounded reachability, expected time and long run averages.

We implemented the translation from ND-GSPNs in PNML to MAPA in the tool GEMMA. This means we can specify an ND-GSPN in PNML and translate it to MAPA. SCOOP can then reduce this MAPA model and generate the MA, which can be checked with the IMCA model-checker.

Contents

Chapter 1

Introduction

Model-checking Advancements in technology follow each other rapidly, which means we become more dependable on technology each day. Medical systems and air planes are example of systems, that require high reliability. Systems need to be verified before we know that they are trustworthy.

To verify a system it is often tested. This involves testing if the system behaves as expected in various situations. The problem is that it is almost impossible to test the system on all situations that can be encountered. A solution for this is *model-checking*. In model-checking the system is specified as a model and on this model every possible situation can be checked. The expected behaviour of the model can be specified several kind of properties, for example safety and liveness properties. *Safety* properties can be used to check if something bad will not happen and *liveness* properties if something good will eventually happen.

Model-checking has its limitations. A know problem is *state-explosion*, where the model contains so much states it is impossible to check all of them. Another problem can be the modelling of the system, as an incorrect model creates inaccurate results. Therefore, the language in which the system is modelled needs to be as general as possible. In this thesis we create a translation which allows us to model more accurate and general with Petri Net models.

GSPNs Petri Nets (PNs) [28] are a widely used modelling technique for describing distributed systems. Petri nets consist of places, which can contain a number of tokens. Transitions can move tokens between places and arcs connect places with transitions and vice versa. Generalized Stochastic Petri Nets (GSPNs) [28] are a more general variant of PNs, which support rate transitions and immediate transitions. Rate transitions fire after an exponentially distributed delay, whereas immediate transitions can fire immediately. GSPNs are a well known formalism to compute dependability and performance of systems. Examples of this are workstation cluster [20], tank fluid system [12] and the Google file system [16] case studies.

The analysis of GSPNs was until now done by translating them to Continuous Time Markov Chains (CTMCs) [34]. A CTMC is a state-based model, where transitions occur after exponentially distributed delays. This translation creates a restriction for the GSPN as a CTMC does not support non-determinism. This means conflicts and confusion can not be resolved with non-determinism. A conflict when firing a transition disables another transition, which means we need to choose what transition will fire. Confusion means that it is not clear if a conflict situation has been resolved or not. These problems are currently solved by introducing weights and priorities for the immediate transitions. The weights are then used to determine the probability

that a transition will fire. However, since confusion is a semantical problem, it can not be resolved in all cases with weights and priorities, which means most analysis tools do not allow any confusion. What we would like is a GSPN that does not require to define weights for each transition. We want a GSPN in which we do not have to specify how we to resolve conflicts between transitions and thus use non-determinism.

Martin Neuhaußer [32] and Joost-Pieter Katoen [24] made a translation from GSPNs to Interactive Markov Chains (IMCs) [21]. An IMC is a state-based model, which combines the rate transitions of a CTMC with interactive transitions, which can fire immediately. They omitted the weights and priorities in his translation by replacing them with non-determinism. The probabilistic choices defined by the weights disappear in this translation and thus some information is lost. We do not want to lose any information in the translation, but we want to add non-determinism to the GSPN.

Joost-Pieter Katoen also suggested that the Markov Automaton would be a better solution for the analysis of GSPNs [24]. This suggesting was an important trigger for this project. We formalise and implement this suggestion in this thesis.

Markov Automata A solution is to translate the GSPN to a Markov Automaton (MA) [13], which is more general than an IMC. An MA is a combination of a Probabilistic Automaton (PA) [38] and an IMC. This means it extends an IMC with probabilistically-distributed next states for the interactive transitions. A translation from GSPNs to MAs could allow us to use the weights and priorities, but also allow non-determinism. We can thus define a more general non-deterministic GSPN.

ND-GSPNs In this thesis we define the Non-Deterministic GSPN (ND-GSPN). ND-GSPNs combine non-determinism with the priorities and weights for immediate transitions. This extends the approach of Martin Neuhaußer [32], as he omits the weights and priorities. ND-GSPNs are more general and expressive than GSPNs, as they allow, but do not require the priorities and weights for immediate transitions. This means we can model more generically and thus model a wider range of models with the ND-GSPN, than with the current GSPN. In this thesis we formally define the ND-GSPN model and express its semantics in terms of the MA model.

Translation We need an implementation of the translation from ND-GSPNs to MAs, in order to be able to perform any analysis on our ND-GSPN model. This means we need a general way of specifying ND-GSPNs and MAs and implement a translation between the two. The MA specification can then be used to generate the MA, so a model-checker can do the analysis on this MA.

The IMCA [2] model-checker is the only model-checker at this moment that can model-check MAs. IMCA supports the model-checking for time-bounded reachability, unbounded reachability, expected time and long run averages. IMCA requires the state space of an MA as input. An MA can be specified in the Markov Automata Process Algebra (MAPA) [41]. MAPA is used as input for the tool SCOOP [40], which uses reductions techniques to minimize the state space and speed-up the state space generation. SCOOP can export an MA to the IMCA input format, which means that an MA in MAPA can directly be checked with the model-checker IMCA. SCOOP also has a web-interface [3], which directly connects SCOOP with IMCA.

To make it easier to exchange Petri Net models between different tools the Petri Net Markup Language (PNML) [45] was developed. PNML is an XML-based language and is the official ISO standard ISO/IEC 15909. We use this standard to specify our ND-GSPNs. PNML does not state how to specify the rate transitions, priorities and

weights for a GSPN, which means we define how the PNML for our ND-GSPN should look like.

GEMMA To facilitate the analysis of ND-GSPNs we implemented the translation from an ND-GSPN as PNML to MAPA in the tool GEMMA. GEMMA requires an ND-GSPN in PNML as input and outputs the corresponding MAPA specification. GEMMA is directly connected with SCOOP via the web-interface, which makes it possible to immediately check an ND-GSPN model with the IMCA model-checker. We formally define the translation from ND-GSPNs to MAPA. We prove that the MA underlying the MAPA specification is the same as the original ND-GSPN expressed as an MA. The total overview of this tool-chain is shown in Figure 1.1.

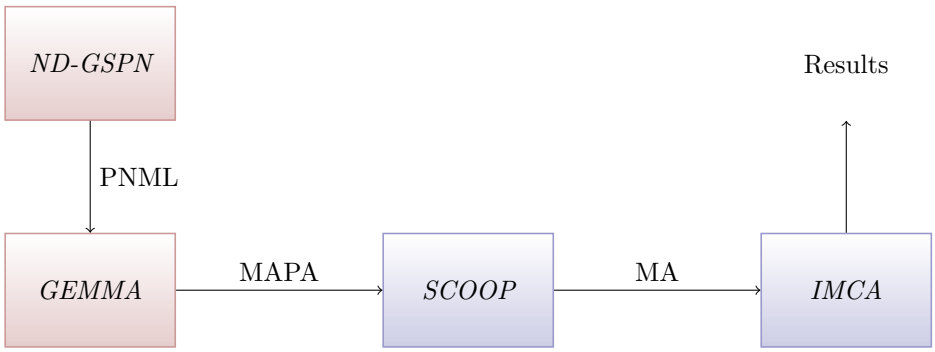


Figure 1.1: Project Results Overview. The red blocks represent our work. The blue blocks are the available work that we will use.

Extensions Translating ND-GSPNs to MAs gives the opportunity to add non-determinism to the GSPN. However, there are other features that can also be added. We add *weighted inhibitor* arcs and *reset* arcs, which both connect a transition to a place. Inhibitor arcs check if a place is empty, but weighted inhibitor arcs check if a place has less than a certain number of tokens. A reset arc empties a place.

We also add *probabilistic* arcs, which make the result of a transition probabilistically-distributed. A transition can have probabilistic arcs, which have a probability that indicates the probability that they will deliver tokens. This allows us to define probability more locally. Weights can be used to specify the probability that a transition will fire. However, this probability is determined globally by checking all weights of the enabled transitions. With the use of the probabilistic arc the result of a transition can distributed probabilistically, which means the probability is more local and independent of other transitions. The probabilistic arc does not introduce new behaviour, because with the proper use of weights and priorities it is also possible to define probability more local. However, the probabilistic arc allows us to model more intuitive and efficient.

Next is a list extensions, that are still supported by an MA. We define these extensions formally and they are used in the translation from PNML to MAPA.

1. Weighted inhibitor arcs.
2. Reset arcs.
3. Probabilistic arcs.

Case Studies Three case studies are done to verify the results of our translation. We use the tool-chain shown in Figure 1.1 to obtain the results in these case studies. The case studies are previously done using the old approach of model-checking GSPNs, where they are translated to CTMCs. This means we can compare our results, using the our new approach, with the results of the old approach. We show that our translation creates the correct MA of the GSPN models. We also show that we can model-check on ND-GSPNs, by using a policy which partly uses weights and partly uses non-determinism to resolve conflicts between immediate transitions.

The three case studies that we consider are:

- Workstation cluster [20]
- Tank fluid system [12]
- The Google file system [16]

Summary The contributions of this thesis are the formal definition of the ND-GSPN and a translation from ND-GSPNs specified in PNML to MAs specified in MAPA. The translation from PNML to MAPA is proven correct and implemented in the tool GEMMA. GEMMA makes it possible to immediately model-check on an ND-GSPN with the model-checker IMCA. The total overview of the project can be seen in Figure 1.1.

Organization of the report Part I contains the background theory. This involves the theory behind the different state-based and token-based models. This part also contains the background about the tools SCOOP [40] and IMCA [2]. Part II contains the theory, which include the definition of ND-GSPNs, the expression of ND-GSPNs as MAs and the formal translation from ND-GSPNs to MAPA. Part III contains the results of this project, which include the description of the tool GEMMA, three case studies and the final conclusions.

Part I

Background

Chapter 2

Theory on State-Based Models

In this chapter we discuss different state-based models. State-based models consist of states, which are connected by transitions. A model can transition from state to state. Later, in Chapter 3 we discus token-based models (Petri Nets [28]). Token-based models have a number of tokens distributed among places. The complete token distribution is the state of the model, where in the state-based models each state is modelled individually.

The reviewed state-based models are: Labelled Transition System (LTS), Discrete Time Markov Chain (DTMC), Probabilistic Automaton (PA), Continuous Time Markov Chain (CTMC), Interactive Markov Chain (IMC) and Markov Automaton (MA). The more general models combine the features of the less general models. How the models are related is shown in Figure 2.1, where the arrows show how the more general models are combination of other models. DTMCs introduce discrete probability distributions, LTSs introduce non-determinism and CTMCs introduce exponentially distributed delays. PAs, IMCs and MAs are combinations of these models. We review these models as they are the base for the Markov Automata, which is the model we are interested in. In order to understand the semantics of MAs we first review the less general models from which an MA is constructed.

State-based models can be put in parallel. We discuss this parallel composition in Section 2.7 on page 15. We only discuss this for the Markov Automata, as all other models are a subset of this.

Preliminaries $\mathsf{Distr}(S)$ is the set of the probability distributions over S, i.e., all functions $\mu\colon S \to [0,1]$ such that $\sum_{s \in S} \mu(s) = 1$. Thus for all $s \in S$ the probabilities sum up to one. This means the probability for $s \in S$ is defined as $\mu(s)$.

An *exponential distribution* describes the time between events in a process in which events occur continuously and independently at a constant average rate. The rate λ characterises the exponential distribution for each event. The probability that an event occurs within d time units is defined as $1 - e^{-\lambda \cdot d}$.

2.1 Labelled Transition Systems

Labelled transition systems (LTSs) are used to model and check processes within systems. An LTS is used to describe the behaviour of a process. It models the state

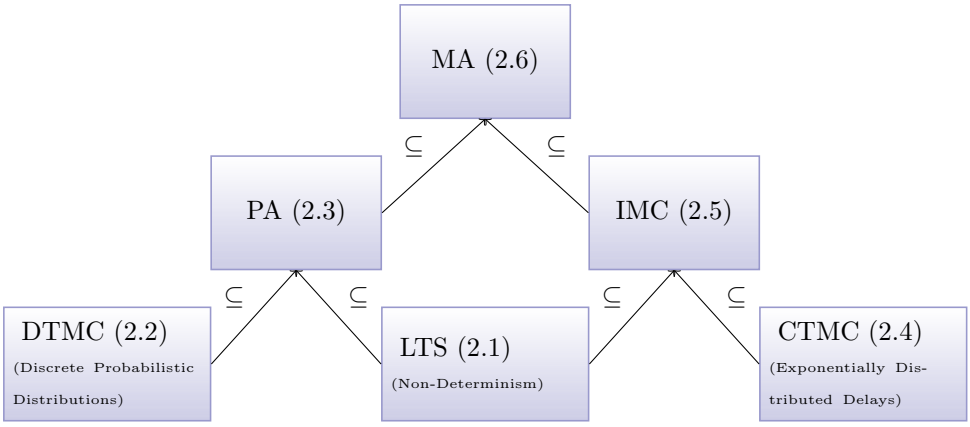


Figure 2.1: Overview of the State-Based Models. The overview also shows in which section each model is discussed.

of these processes and the transitions between those states. The behaviour is defined by the events or actions that can occur in the system, which can move the system to a different state.

The analysis of an LTS can be done by a model-checker. A model-checker can be used to check every possible situation that can occur in the model. The expected behaviour of the LTS model can be specified in properties, which can be in CTL or LTL for example [5]. The model-checker can for example check which states are reachable with which events. This means that it is possible to check all possible states of the system. Model-checking is more complete than normal testing where only a finite number of test cases are executed in the system. In model-checking all possible cases are checked, if the state space is finite. For an infinite state space it is not possible to check all states, which means the analysis will not terminate.

An LTS consists of a set of states and transitions between those states. The transitions are labelled with labels from a chosen set. The labels represent the events or actions that can occur in the system, which move the system to a different state.

Definition 1. *A labelled transition system is a 4-tuple* $\langle S, A, \hookrightarrow, s_0 \rangle$*, where:*

- S *is the countable set of states*
- A *with* $\tau \notin A$ *is the finite set of input symbols*
- $\hookrightarrow \subseteq S \times (A \cup \{\tau\}) \times S$ *is the transition relation*
- $s_0 \in S$ *is the initial state*

If $(s, a, s') \in \hookrightarrow$, we write $s \overset{a}{\hookrightarrow} s'$ and say that the action $a \in A$ can be *executed* from state s, after which we go to the next state s'. An LTS is in general *non-deterministic*, which means for an action there may be multiple next states. In special the case that for each action there is exactly one next state, we call an LTS *deterministic*.

LTSs can also have internal actions, which are defined as τ-transitions. These transitions have the τ label and do not need any input.

Example 2. Figure 2.2 shows an example of an LTS that models a coffee machine. The initial state is s_0. In this state a coin can be inserted, which can be accepted

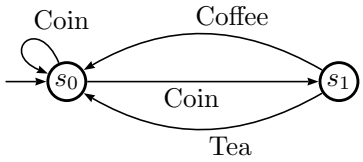


Figure 2.2: Labelled Transition System example of a simple coffee machine.

(move to s_1) or rejected (stay in s_0). In state s_1 we can get Coffee or Tea and return to state s_0.

This LTS can be formally defined by the tuple $\mathcal{A} = \langle S, A, \hookrightarrow, s_0 \rangle$, where

- $S = \{s_0, s_1\}$
- $A = \{\text{Coin, Coffee, Tea}\}$
- $\hookrightarrow = \{(s_0,\text{Coin}, s_0), (s_0,\text{Coin}, s_1), (s_1,\text{Coffee}, s_0), (s_1,\text{Tea}, s_0)\}$

This LTS is non-deterministic, because we can end up in s_0 or s_1 with the action Coin in state s_0.

2.2 Discrete Time Markov Chains

Discrete Time Markov Chains (DTMCs) [34] are used to model stochastic processes in discrete time. It models processes which evolve in time. DTMCs introduce probabilistic behaviour of the transitions. The next state is determined only by the current state and a probability.

DTMCs can for example be used to determine time-bounded reachability and long-run averages. Time-bounded reachability can be used to check the probability to reach a state in a certain amount of time. Long-run averages can be used to determine the average time in a state on the long-run.

A transition leads to a probabilistic choice over several next states. The next state of a transition is determined by the probabilistic transition function. In a DTMC a transition occurs after each time unit. The transitions in a DTMC do not have labels, but only a probability.

Definition 3. *A DTMC is a 3-tuple $\langle S, s_0, P \rangle$ where:*

- *S is the finite set of states with s_0 the initial state*
- *$P \colon S \to \mathsf{Distr}(S)$ is the probabilistic transition function*

The next state is probabilistically distributed over S. This means for state $s \in S$ the next state is distributed by $\mu \in \mathsf{Distr}(S)$, where the probability to move from s to s' is $\mu(s')$. A DTMC is a stochastic process that has the *Markov Property*. This property states that the next state only depends on the current state and not on the previous states (*memoryless*). This follows directly from the definition as the transition behaviour is defined for each state and not for each path of transitions.

Example 4. Figure 2.3 is an example of a DTMC which models the weather forecast for the next day (numbers are made up). The transitions are labelled with probabilities. After each time unit, a day in this example, one of the transitions will be taken. At the initial state (Sunny) there is a probability of 0.7 that the next day is also sunny, a probability of 0.1 that it will rain the next day and a probability of 0.2 that the next

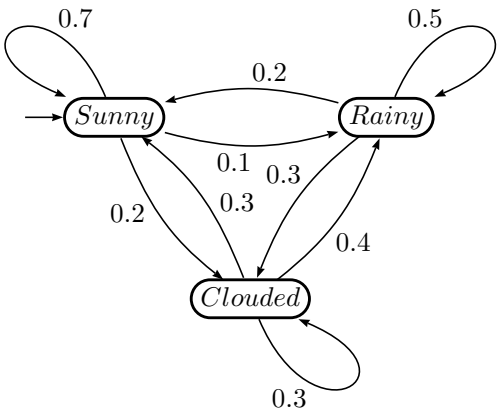


Figure 2.3: Discrete Time Markov Chain example of a weather forecast.

day will be clouded. The sum of outgoing transitions of a state always equals one. We also see that the weather of the next day only depends on the weather of the current day (*Markov Property*).

2.3 Probabilistic Automata

Probabilistic Automata (PAs) [38] are used to model and analyse asynchronous, concurrent systems with discrete probabilistic choice in a formal and precise way. Examples of PAs are randomized, distributed algorithms, probabilistic communication protocols and the randomized dining philosophers problem [27].

The PA model is a combination of the LTS and the DTMC model. A PA can have multiple transitions in each state for which the next state is distributed probabilistically. This means that each transition has its own label and is a probabilistic choice over several next states. A transition can be taken non-deterministically and the probability function determines the probability of each next state.

A PA provides more expressiveness than the LTS and the DTMC model. A PA can specify the probabilities of transitions, where an LTS can only give a non-deterministic option. A DTMC also contains probability, but it allows only one event. A PA can thus be used to specify more detailed information in the model.

Definition 5. *A PA is a 4-tuple* $\langle S, A, \hookrightarrow, s_0 \rangle$ *where:*

- S *is the finite set of states with* s_0 *the initial state*
- A *with* $\tau \notin A$ *is the finite set of actions*
- $\hookrightarrow \subseteq S \times \Sigma \cup \{\tau\} \times \mathsf{Distr}(S)$ *is the set of probabilistic transitions*

In each state the transition with input $a \in A$ is a probabilistic choice between several next states. If $(s, a, \mu) \in \hookrightarrow$, we write $s \overset{\alpha}{\hookrightarrow} \mu$, where the probability to move with input $a \in A$ from s to s' is $\mu(s')$. When $\mu(s') = 1$ the transition can be seen as non-probabilistic (*Dirac Distribution*).

A PA in which each transition can be seen as non-probabilistic is an LTS. A PA without non-determinism and only one action can be seen as a DTMC. In a PA multiple transitions with the same label can exist. When a PA only allows one transition for each label we call it an MDP [6] (Markov Decision Process). The transition function for an MDP is $\hookrightarrow : S \times \Sigma \cup \{\tau\} \to \mathsf{Distr}(S)$.

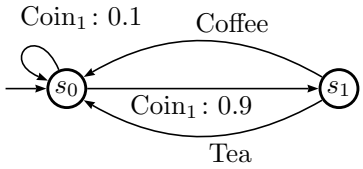


Figure 2.4: Probabilistic Automaton example of a simple coffee machine.

Example 6. Figure 2.4 is an example of a PA which models a coffee machine. There are several labelled transitions between the states which contain a probability and a label. For each label the probabilities sum up to one.

We indicate the probabilistic distribution for the labels by adding an identifier, i.e., we added the identifier 1 to the coin label (Coin$_1$). We need to add this identifier, because in the PA model it is possible that there are other distributions for the coin label. The different distribution can then be taken non-deterministically.

The difference with the LTS is that the probability of the coin being rejected can now be specified. In this case the probability that the coin is rejected is 0.1 and that it will be accepted is 0.9. The transitions Tea and Coffee do not have a next state which is probabilistically distributed and therefore they have the probability 1.

2.4 Continuous Time Markov Chains

Continuous Time Markov Chains (CTMCs) [34] are used to model stochastic processes in continuous time, which means they can be used to describe processes where the system evolves in constant time. CTMCs are for example used to model problems in queueing theory. The arrival of costumers over time is typically exponentially distributed. Systems as telephone switchboards and arrivals of customers at a service desks, are good examples of what can be modelled with CTMCs.

In CTMCs, also known as Markov Processes, the system will remain in the current state for some random amount of time before transitioning to a different state.

Definition 7. *A CTMC is a tuple $\langle S, R, s_0 \rangle$ where:*

- *S is the finite set of states, with s_0 as initial state*
- *$R\colon S \times S \to \mathbb{R}$ is the rate function*

The transitions of a CTMC are also called Markovian transitions. These transitions have a *rate* λ, which characterizes an exponentially distributed delay. If a state has multiple Markovian transitions, they will race each other (*race condition*). The transition whose delay expires first, which means has the smallest delay, will be taken. For states $s, s' \in S$, let the rate of a Markovian transition be $R(s, s')$ and $E(s)$ be the *exit rate* of the state. The exit rate is the sum of outgoing rates in a state s defined as $E(s) = \sum_{s' \in S} R(s, s')$.

Definition 8. *The probability of moving from state s to state s' within d time units in one step is:*

$$\frac{R(s, s')}{E(s)} \cdot (1 - e^{-E(s) \cdot d})$$

The rate of the transition is divided by the exit rate of the state. This is explained by the fact that if there are multiple outgoing transitions, the transition which delay

expires first will be taken. This means that the probability that a transition fires decrease, when there are multiple transitions enabled. The probability that a transition is taken is a function from the time d. A higher delay d means a higher probability that the transition fires.

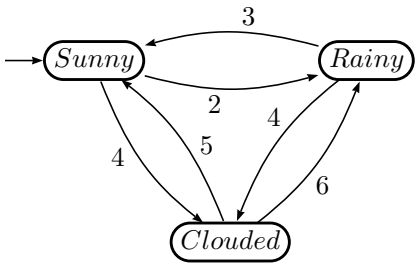


Figure 2.5: Continuous Time Markov Chain example of a weather forecast.

Example 9. Figure 2.5 is an example of a CTMC which models a weather forecast in days d (numbers made up). $E(Sunny) = 2 + 4 = 6$ and $R(Sunny, Clouded) = 4$. This means the probability to move from a sunny day to a clouded day in d days is $P(Sunny, Clouded) = \frac{4}{6} \cdot (1 - e^{-6 \cdot d})$. This is the probability of moving from a sunny day to a clouded day in d days. For the total probability of getting from a sunny to a clouded day within d time units we also need to consider the other paths from sunny to cloudy (for example the path: Sunny, Rainy, Clouded). This means that the probability of getting in a state after d time units is the summation of all the probabilities of all paths ending in that state.

2.5 Interactive Markov Chains

An Interactive Markov Chain [21] is a combination of an LTS and a CMTC model. IMCs are for example used as underlying semantics for dynamic fault trees [10] and architectural description languages (as ARCADE [9]).

IMCs include interactive transitions and exponential exit rates (Markovian transitions). The interactive transition are labelled (as in LTSs) and the Markovian transitions are labelled with rates (as in CTMCs).

Definition 10. *An IMC is a 5-tuple $\langle S, A, \hookrightarrow, \rightsquigarrow, s_0 \rangle$ where:*

- *S is the finite set of states with s_0 as initial state*
- *A with $\tau \notin A$ is the finite set of input symbols*
- *$\hookrightarrow \subseteq S \times \Sigma \cup \{\tau\} \times S$ is a set of interactive transitions*
- *$\rightsquigarrow \subseteq S \times \mathbb{R}_{>0} \times S$ is a set of Markovian transitions*

Intuitively, an IMC operates as follows: in every state s, an action a can take place, moving to state s', if there is a transition $(s, a, s') \in \hookrightarrow$. If there is a transition $(s, \lambda, s') \in \rightsquigarrow$, then the system can move from s to s' after an exponentially distributed delay governed by rate λ. We write $s \overset{\alpha}{\hookrightarrow} s'$ for interactive transitions and $s \overset{\lambda}{\rightsquigarrow} s'$ for Markovian transitions.

In a state with both a Markovian transition and a τ-transition, the τ-transition can be taken immediately at time zero. At time zero the probability of the Markovian

transition being taken is 0. This means that τ transitions will always take precedence over Markovian transitions (*Maximal Progress Property*).

The Markovian transitions in states with interactive transitions can be deleted from the model as they will never take place. This means that an IMC model can always be transformed to an equal IMC model in which all states are either Markovian (with Markovian transitions outgoing) or interactive (with interactive outgoing transitions). This can only be done if the model is *closed*, which means all interactive transitions can be seen as τ transitions. The analysis of IMCs is done on closed IMCs, which thus only have the label τ. Markovian transitions in interactive states are allowed in the model, because they might have influence in the parallel composition of IMCs. Parallel composition is discussed in Section 2.7.

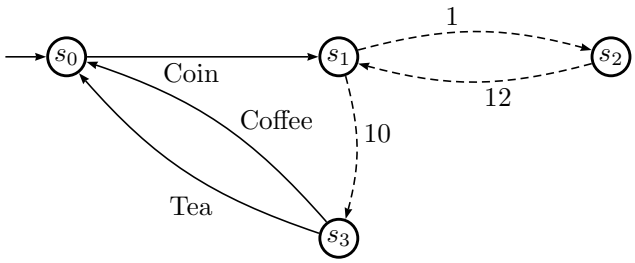


Figure 2.6: Example of an Interactive Markov Chain. Dashed arrows are Markovian transitions and solid arrows are interactive transitions.

Example 11. Figure 2.6 shows an example of an IMC, which models a coffee machine. The Markovian states are s_1 and s_2 and the interactive states are s_0 and s_3. After a coin is inserted the transitions to s_2 and s_3 race with each other, because they are both enabled in state s_1. s_2 represents the error state which will be entered when the coffee machine breaks. After a while (rate 12) the machine will be repaired. If the delay of the transition to s_3 expires first, then the coffee or tea transition can be taken.

2.6 Markov Automaton

The Markov Automaton (MA) [13] model extends the IMC model with probabilistic behaviour of interactive transitions. An MA can be seen as a combination of an IMC and a PA. The MA model is the most general model that we consider. All previous models are a subset of the Markov Automaton, which means it can be used to model a wide range of processes. MAs support non-determinism, probability and stochastic timing in the form of exponentially distributed delays.

Definition 12. *A Markov Automaton is defined as a 5-tuple* $\langle S, s_0, A, \hookrightarrow, \rightsquigarrow \rangle$ *where:*

- S *is the finite set of states with* s_0 *as initial state*
- A *is the finite set of input symbols*
- $\hookrightarrow \subseteq S \times (A \cup \{\tau\}) \times Dist(S)$ *is a set of interactive transitions*
- $\rightsquigarrow \subseteq S \times \mathbb{R}_{>0} \times S$ *is a set of Markovian transitions*

If $(s, a, \mu) \in \hookrightarrow$, we write $s \overset{\alpha}{\hookrightarrow} \mu$ and say that the action a can be *executed* from state s, after which the probability to go to $s' \in S$ is $\mu(s')$. If $(s, \lambda, s') \in \rightsquigarrow$, we write $s \overset{\lambda}{\rightsquigarrow} s'$ and say that s moves to s' with rate λ.

MA behaves much like the IMC model with the extensions that the interactive transitions are distributed probabilistically. An MA where the transitions are all Dirac distributions, thus without probabilistic choices, is an IMC. An MA with $\rightsquigarrow = \emptyset$ is a PA and an MA where $\hookrightarrow = \emptyset$ is a CTCM.

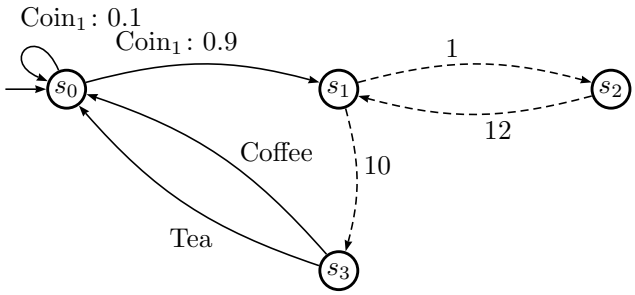


Figure 2.7: Markov Automaton

Example 13. Figure 2.7 shows an MA, which models a coffee machine. It combines Figure 2.4 and Figure 2.6. The coin is accepted with probability 0.9 and rejected with probability 0.1. The model also has the s_2 error state, which represents that the machine is broken. This model combines the distributed probability of transitions as well as the combination of interactive and Markovian transitions.

2.7 Parallel Composition

State-based models can be put in parallel. With parallel composition it is possible to create big systems which are composed from separate models. We discuss the *parallel composition* [13] for MAs, as all previous models are a subclass of MAs.

Parallel composition makes it possible to synchronize two MAs and their shared actions. Let $MA_1 = \langle S_1, A_1, \hookrightarrow_1, \rightsquigarrow_1, s_{0_1} \rangle$ and $MA_2 = \langle S_2, A_2, \hookrightarrow_2, \rightsquigarrow_2, s_{0_2} \rangle$. We define $\parallel$ as the parallel composition operator. We denote $s_1 \parallel s_2$ as the state which arises by the parallel composition of s_1 and s_2. Let the probability distribution over MA_1 be μ_1 and over MA_2 be μ_2. The parallel composition of the distribution $\mu_1 \parallel \mu_2$ is defined as $(\mu_1 \parallel \mu_2)(s_1 \parallel s_2) = \mu_1(s_1) \cdot \mu_2(s_2)$

Definition 14. *We define* $MA_1 \parallel MA_2 = \langle S, A_1 \cup A_2, \hookrightarrow, \rightsquigarrow, s_0 \rangle$*, where:*

- $S = S_1 \times S_2$
- $(s_1 \parallel s_2, \alpha, \mu_1 \parallel \mu_2) \in \hookrightarrow$ *iff either*
  - $\alpha \in A_1 \cap A_2$ *and for each* $i \in \{1, 2\}, (s_i, \alpha, \mu_i) \in \hookrightarrow_i$ *or*
  - $\alpha \notin A_1 \cap A_2$ *and* $(s_1, \alpha, \mu_1) \in \hookrightarrow_1 \wedge \mu_2 = \{s_2 \mapsto 1\}$ *or* $(s_2, \alpha, \mu_2) \in \hookrightarrow_2 \wedge \mu_1 = \{s_1 \mapsto 1\}$
- $(s_1 \parallel s_2, \lambda, s_1' \parallel s_2') \in \rightsquigarrow$ *iff either*
  - *for each* $i \in \{1, 2\}, R(s_i, s_i') > 0 \ \wedge \ s_i = s_i'$ *then* $\lambda = R(s_1, s_1') + R(s_2, s_2')$ *otherwise*
  - $\lambda = R(s_1, s_1')$ *and* $s_2' = s_2$*, or* $\lambda = R(s_2, s_2')$ *and* $s_1' = s_1$*.*
- $s_0 = s_{0_1} \parallel s_{0_2} \in S_1 \times S_2$ *is the initial state*

The definition of parallel composition is backwards compatible with all the discussed sub-models of the MA.

For the interactive transitions we see that the transitions with the same labels from the same parallel states also occur in the parallel composition. A τ transition occurs when one of the parallel states has an interactive transition, which the other does not have. If both states have a Markovian transition then the new Markovian transition is the sum of the rates. If only one has a Markovian transition then this one is used.

Chapter 3

Theory on Token-Based Models

Now we review a different kind of probabilistic models, the token-based models. The models that we consider are several variants of Petri Nets [28]. Petri Nets are a widely used modelling technique for describing distributed systems.

The models that we review are: Petri Net (PN), Stochastic Petri Net (SPN) and Generalized Stochastic Petri Net (GSPN). An overview of the models can be seen in Figure 3.1. We added the non-deterministic GSPN to illustrate our work. ND-GSPNs are described in Section 5 on page 33.

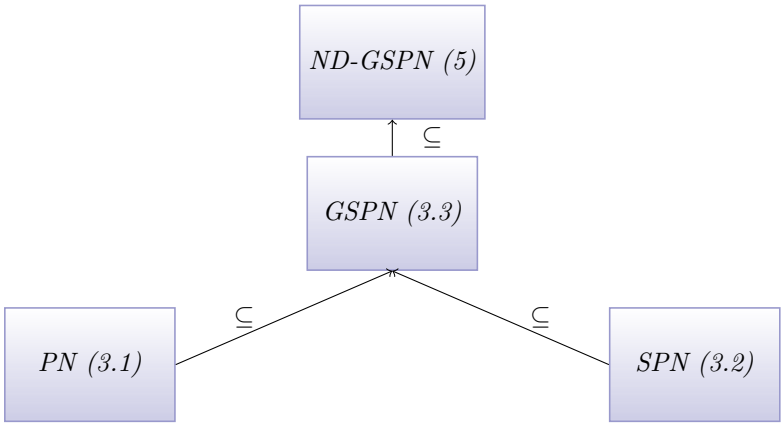


Figure 3.1: Overview of Token-Based Models. We also show the section in which the model is discussed.

History First we give a brief summary of the history of the Petri Nets, mainly derived from [28]. Petri Nets were originally introduces by Carl Adam Petri in 1966 [35]. He used them to describe concurrent systems in terms of cause and effect relations with time explicit time considerations. Several years later different people [30, 36, 37] tried to introduce temporal concepts in the PN models with different approaches, mostly based on deterministic timing, i.e, transitions take a fixed delay before they fire. Later Symons [39], Floring and Natking [14] and Molloy [31] introduced stochastic timing in

PNs, i.e., the time before a transition fires is governed by a probabilistic distribution. From this moment the PNs could be linked to the field of performance evaluation. PNs with stochastic timing are currently known as Stochastic Petri Nets (SPNs). The SPNs were extended with immediate transitions or null delays in 1984 [29]. This model is currently known as a Generalized SPNs (GSPNs). Examples of known case studies for GSPNs are the Working Station [20] and Google File System [16] case studies.

Extensions There are various extensions and variations of Petri Nets. The most used extension is the *inhibitor* arc. We include this arc in our definition of the PN, SPN and GSPN model. It is equipped with a condition that requires a place to be empty. The inhibitor arc can be extended with a weight. This would mean it requires a place to have less than a specified number of tokens. Another extension is the reset arc, which can empty a place when a transition fires.

Coloured Petri Nets (CPNs) [23] can have multiple types of tokens. This means that multiple types of tokens can be modelled in one place. CPNs are mainly used to model more compactly. For example, the Dining Philosophers problem [27] can be modelled smaller in CPNs than in normal PNs.

Another variant are the Fluid Stochastic Petri Nets (FSPNs) [42]. These are an extension of GSPNs which add fluid places and arcs. Fluid places contain a continuous fluid level. This can be used to model continuous variables as temperatures and pressure. The firing of a transition may depend on the fluid level of a place connected with a fluid arc.

Tooling There are many tools for the specification and analysis of Petri Nets (for example PIPE [8] and GreatSPN [4]). A database with the overview of these tools can be found in the Petri Nets World website [33]. This database shows which type of Petri Nets each tool supports, what their main features are and on which platform they run.

To make it easier to exchanges models between tools, the Petri Net Markup Language (PNML) [45] was developed. PNML is an ISO standard for the specification of Petri Nets. Unfortunately, not every tool uses the PNML syntax. A tool which support GSPNs and PNML is the PIPE [8] tool, meaning it is useful for us.

3.1 Petri Nets

First we consider the Petri Net (PN) [28] model. PNs are used to model systems in which activities may take place concurrently under the precedence of frequency constraints. The four philosophers problem [27] or the producer and consumer problem are good examples that can be modelled with PNs.

A Petri Net consist of a number of *places*, which can contain a number of *tokens*. *Transitions* can move these tokens between places. The movement of tokens through the places can be seen as actions in the model. *Arcs* connect the places to the transitions. The distribution of tokens over the places is called a *marking*. The marking changes after a transition. The marking can be seen as the state of the model.

Definition 15. *A Petri Net is a 7-tuple* $PN = \langle P, T, F, I, RE, MU, M_0 \rangle$*:*

- P *is the finite set of places*
- T *is the finite set of transitions*
- $F \subseteq (P \times T) \cup (T \times P)$ *is the finite set of arcs*

- $I \subseteq P \times T$ *is the finite set of inhibitor arcs*
- $RE \subseteq P \times T$ *is the finite set of reset arcs*
- $MU \colon (F \cup I) \to \mathbb{N}_{>0}$*, is a multiplicity function*
- $M_0 \colon P \to \mathbb{N}_{>0}$*, is the initial marking*

An arc connects a source to a target (place or transition). An arc has a *multiplicity* which represents how many tokens it will move. A transition may have multiple in or outgoing arcs, which can only be taken at the same time.

Transition can also have *inhibitor* and *reset* arcs. The source place connected to an inhibitor arc must have less tokens than the multiplicity of the arc or else the transition can not fire. In most Petri Net models the inhibitor arcs do not have multiplicities, but they require a place to be empty. In our definition this can be seen as a multiplicity of 1 for the inhibitor arc.

A reset arc will empty the place it is connected to. We call a transition *enabled*, when it can fire. When multiple transitions are enabled, they can fire non-deterministically in an interleaving sequence. The initial marking is the initial token distribution among the places.

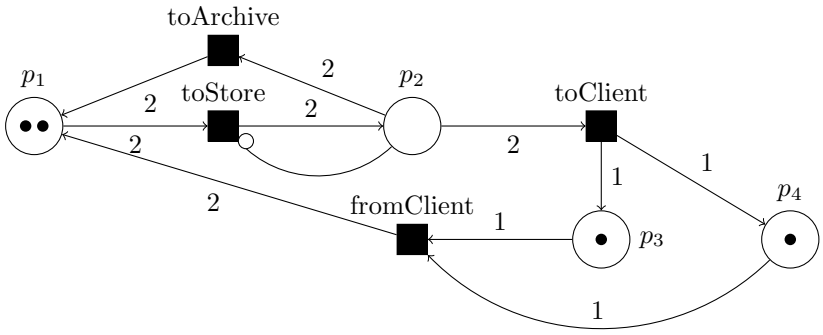


Figure 3.2: Petri Net example of a simple library.

Example 16. Figure 3.2 shows a Petri Net model of a library. The places are represented as circles with an amount of tokens. The transitions are black squares and the arcs are the arrows. The numbers on the arrows represent the multiplicity of the arc. This model shows a library where initially two books are stored in the archive (p_1) and each client has one book (p_3, p_4). Only two books at the time can be transferred to the store (place 2) and the store only has room for two books so it must be empty before it can store books (inhibitor arc, modelled with a dot at the transition side of the arc). From the store the books can be delivered to two different clients (place 3 and 4). The clients must then return a book at the same time to get the books back at the archive. The books can also be delivered back from the store to the archive.

Analysis of a Petri Net is done by creating the reachability diagram of all the markings. The reachability diagram of a Petri Net is actually an LTS, where the states are all the possible markings. The transitions in a PN and an LTS can be seen the same way. The difference in the models is the representation of the current state, i.e., in a PN the token distribution is the state and in an LTS each state is modelled individually. We only present the translation from PNs to LTSs intuitively, because the translation in Section 5.2 on page 37 covers all types of Petri nets we discuss.

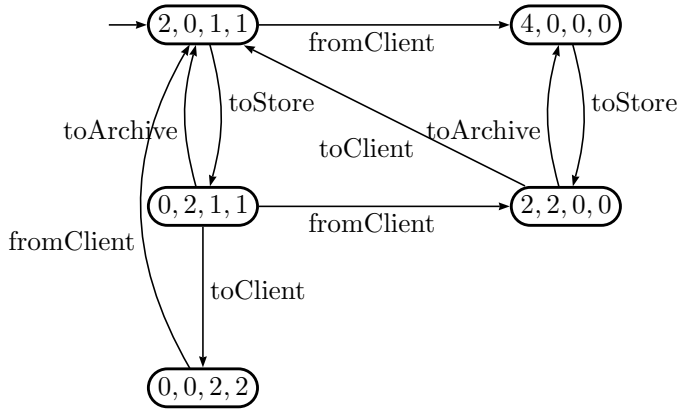


Figure 3.3: Reachability Diagram of the PN in Figure 3.2

Example 17. Figure 3.3 shows the reachability diagram of the Petri Net of Figure 3.2. The initial marking is {2,0,1,1}. From this marking it is possible to move to marking {0,2,1,1} or {4,0,0,0}.

When multiple transitions are enabled there may be a *conflict*. A conflict occurs when a transitions fires and another transition gets disabled, which means we need to chose which transition should fire. These conflicts can be resolved using different policies. For PNs conflicts are resolved by picking a transition non-deterministically, which means we do not specify how we resolve the conflict. We show later, in Section 3.3, that conflicts can not be solved with non-determinism for all types of Petri Nets.

Example 18. In Figure 3.2 the transitions toArchive and toClient are in conflict, which means that firing one of them disables the other. This is called a free-choice conflict, because both transitions are always enabled at the same time. In Figure 3.2 the conflict is solved by picking toArchive or toClient non-deterministically.

3.2 Stochastic Petri Nets

In Stochastic Petri Nets (SPNs) the transitions are timed with exponentially distributed delays. SPNs are used for performance modelling of complex stochastic systems. The use of timing in Petri Nets is crucial in the areas of hardware and computer architecture design, communication protocols and software system analysis.

Definition 19. *An SPN is a 8-tuple* $SPN = \langle P, T, F, I, RE, MU, R, M_0 \rangle$*:*

- $\langle P, T, F, I, MU, M_0 \rangle$ *is the underlying PN.*
- $R\colon T \to \mathbb{R}$ *is the rate function*

Transitions in an SPN have a rate λ, that characterises an exponentially distributed delay. An SPN uses the same exponential distribution function for its delays as the CTMC model in Section 2.4 on page 12. The transition have atomic firing and race policy. This means that the transition which expires first will be taken and tokens will leave the source place and will immediately appear at the target place.

The transitions of an SPN are the same as the transitions of a CTMC. The reachability diagram of the SPN is a CTMC. Analysis is therefore done on the underlying CTMC of the SPN.

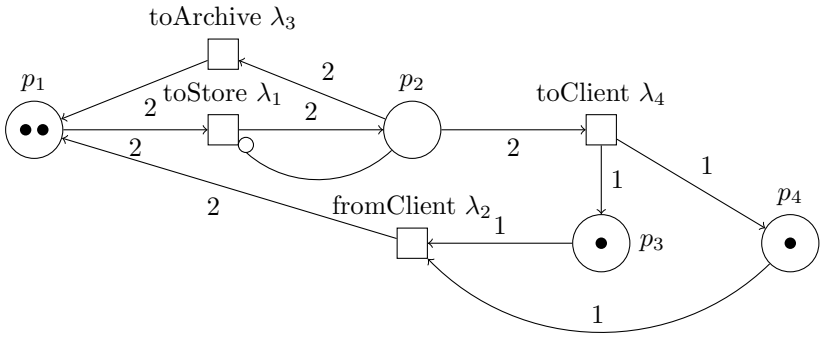


Figure 3.4: Stochastic Petri Net example of a simple library.

Example 20. Figure 3.4 shows a Stochastic Petri Net model. The transitions are now taken by exponentially distributed delays. In the store place (p_2 which is initially empty) there are two transitions which race each other (λ_2 and λ_3). The exponentially delayed transitions are modelled as white blocks.

3.3 Generalised Stochastic Petri Nets

Generalized Stochastic Petri Nets (GSPNs) [29] are the most general variant of Petri Net models that we consider. GSPNs are a well known formalism to compute dependability and performance of systems. Known examples are the Google File System [16] and the Working Station System [20].

A GSPN combines immediate transitions and rate transitions. A GSPN is thus a combination of a normal PN and an SPN.

Definition 21. *A GSPN is a 11-tuple* $GSPN = \langle P, IMT, RT, PRI, W, F, I, RE, MU, R, M_0 \rangle$ *where:*

- $\langle P, RT, F, I, MU, R, M_0 \rangle$ *is the underlying SPN*
- IMT *is the finite set of immediate transitions*
- $PRI : (IMT \cup RT) \to \mathbb{N}^0$ *defines the priorities for the transitions.*
- $W : IMT \to \mathbb{N}$ *defines the weights for transitions*

where we use $T = IMT \cup RT$ *to denote all transitions.*

Analysis of a GSPN is currently done by transforming it to a CTMC. As a CTMC does not support non-determinism, neither can the GSPN. This means conflicts between transitions need to be resolved without non-determinism. Another problem is *confusion*.

Confusion occurs when conflicts and *concurrency* are mixed. Concurrency occurs when multiple transitions are enabled which are not in conflict. Confusion happens when multiple transitions are enabled at the same time and firing one of them causes a conflict situation and firing the other one does not. This means that it is not known if a conflict has been resolved after the two transitions both have fired. This means that by definition conflict and concurrency can not be mixed in a GSPN.

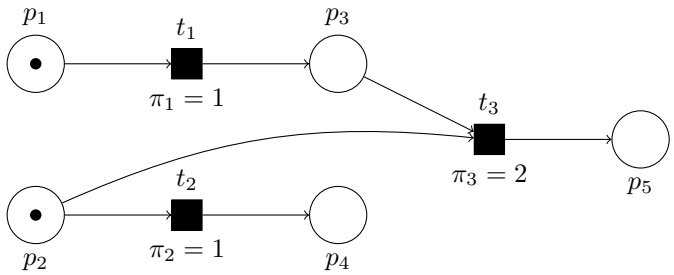


Figure 3.5: Petri Net example with confusion.

Example 22. Figure 3.5 shows an example of confusion. This example is taken from [28]. For this example we ignore the priorities of the transitions. If t_1 fires we get a conflict situation between t_2 and t_3. However, if t_2 fires before t_1 then this conflict will not happen. Thus if t_1 and t_2 both have fired it is not clear if the conflict between t_2 and t_3 has been resolved or not. This information can not be represented by the CTMC that does the analysis, which means the analysis of a model with confusion is not possible as we never know how the confusion was resolved.

The GSPN model introduces the concepts of *priority* and *weights* for transitions to manage conflicts and confusion. Each immediate transition will get a priority, which is a natural number greater than 0. Rate transitions get the priority 0. If multiple transitions are enabled, the one with the highest priority will be taken. As the rate transitions always have priority 0 the immediate transitions will always be taken over rate transition (maximal progress property).

Example 23. The confusion in Figure 3.5 is resolved by giving t_3 a higher priority than t_2 (indicated by π_2, π_3). This means the confusion will not happen because the situation of firing t_1 and then t_2 can not happen any more. An alternative to resolve the confusion is the use of weights for the transitions, causing conflicts to be resolved stochastically.

Each immediate transition will also get a weight. Depending on which transitions of the same priority are enabled, the weights are used to determine the probability of the enabled transitions. Weights are thus used to resolve conflicts between transitions of equal priority.

The weights and priorities are only introduced because the GSPN is translated to an underlying CTMC for analysis. A CTMC only has rate transitions, thus the probabilities of all immediate transitions need to be merged with the rate transitions, which can only be done if the net is not confused.

However, as confusion is a semantic phenomenon, it cannot be resolved in all cases with weights and priorities. This is the reason that most analysis tools do not allow any confusion in the subnets which only contain immediate transitions. Several conditions on net level have been proposed to avoid confusion, as for example the use of extended conflict sets [11], which help to detect transitions that are in conflict. The use of non-determinism would resolve the problem of confusion as this would make it possible to not specify a choice in a conflict situation.

Example 24. Figure 3.6 shows a General Stochastic Petri Net model. This model combines exponentially delayed transitions and immediate transitions. The rate transitions are modelled as white blocks where the immediate transitions are black blocks.

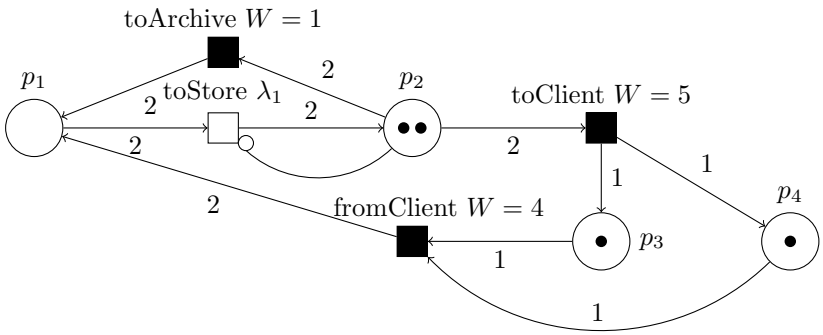


Figure 3.6: General Stochastic Petri Net example of a simple library.

To make sure that the immediate transitions always take precedence over the rate transitions they have priority 1 and the rate transitions have priority 0. The immediate transitions also get a weight. These weights are shown on the transitions (toArchive 1, fromClient 4, toClient 5).

Depending on which transitions are enabled, the probability of each transition is computed. The probability is the weight of the transition divided by the weights of all enabled transitions. In the initial state the transitions toArchive (weight 1), toClient (weight 4) and fromClient (weight 5) are enabled with probabilities $1/(1+4+5)$ for toArchive, $5/(1+4+5)$ for toClient and $4/(1+4+5)$ for fromClient. If fromClient fires the transitions toArchive and toClient are enabled with probabilities $1/(1+5)$ for toArchive and $5/(1+5)$ for toClient. The probabilities depend on which transitions are enabled and thus are not always the same for each transition. The rate transition toStore can only fire if no immediate transition can fire.

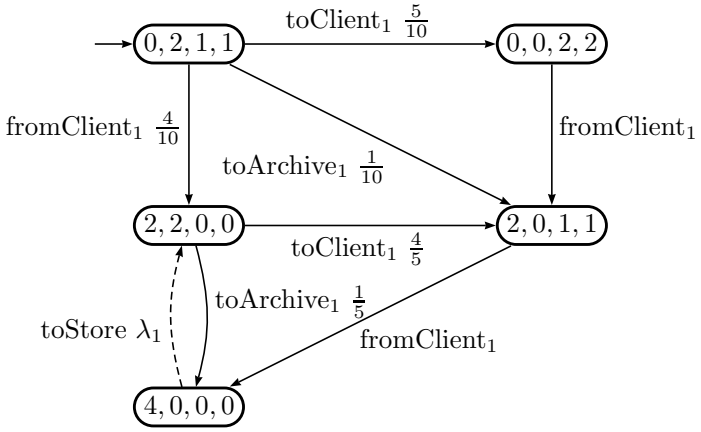


Figure 3.7: Reachability Diagram of the GSPN in Figure 3.6

Example 25. The reachability diagram of the GSPN includes the rate transitions and the probabilistic choice. The probabilities are calculated in the same way as in example 24. The reachability diagram of a GSPN is a CTMC (i.e., a purely stochastic MA). Note that the transition labels are also present in this diagram for illustrating purposes, but in practice all the labels are renamed to one τ transition, which is distributed probabilistically, for each state. This makes the reachability a purely stochastic MA.

Summary of the Models

We reviewed state-based probabilistic models up to the Markov automaton. Markov automata combine stochastic timing, probability and non-determinism, which means that they are the most complete state-based model that we considered. All other described state-based models are subsumed by the Markov Automaton.

We also considered the token-based Petri Net models. The GPSN model is the most complete token-based model that we considered. GSPNs combine stochastic timing and probability. A GSPN is translated to a CTMC for analysis. A CTMC does not support non-determinism, which means neither can the GSPN. This means all conflicts between immediate transitions need to be resolved without non-determinism. This is done with the use of weights and priorities. This is not ideal as we might not want to specify how to resolve each conflict.

Table 3.1 shows an overview of all the state-based and token-based models. It shows the most important features of all the models we reviewed. We also added the non-deterministic GSPN (ND-GSPN). An ND-GSPN is a GSPN which introduces non-determinism to resolve conflicts between immediate transitions. The definition of ND-GSPNs is discussed later in Chapter 5. We see that only the MA and the ND-GSPN model support all three features. This gives the opportunity to translate the ND-GSPN model to the MA model. In the remainder of this report we define this new ND-GSPN and the translation to the MA model. This allows us to define models as ND-GSPNs and do the analysis on the underlying MA.

	Non-Determinism	**Probability**	**Exit Rates**
State-Based Models			
LTS	X		
DTMC		X	
PA	X	X	
CTMC			X
IMC	X		X
MA	X	X	X
Token-Based Models			
PN	X		
SPN			X
GSPN		X	X
ND-GSPN	X	X	X

Table 3.1: Probabilistic Models

Figure 3.8 shows the complete overview of the analysis of Petri Nets. We see how the different Petri Nets are subsets of each other and what kind of state-based model is used for their analysis. We added the ND-GSPN to illustrate the difference of our approach. We see that the current models are all subsets of our approach. This means

that the old approach is still possible in our new approach. Our approach is thus more general.

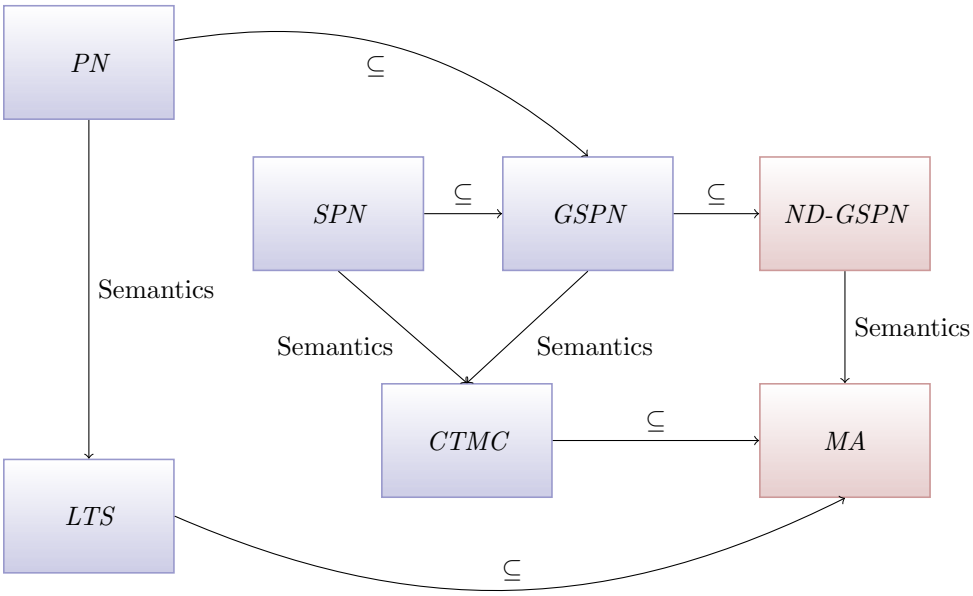


Figure 3.8: Overview Analysis of Petri Net Models. The blue blocks indicate the current models and how they are analysed. The red blocks indicate our model with our way of analysis.

Note that a GSPN supports probabilities, but the CTMC does not. This means the probabilities of the GSPN need to merge with the rate transitions to make this translation. This can only be done when the GSPN does not contain confusion, as discussed in the previous chapter.

Chapter 4

Tooling for Markov Automata

We discuss the tools which are part of the total overview in Figure 1.1. The tools we discuss are SCOOP [40] and IMCA [2]. MAPA [41] is also discussed as this is the language used as input for SCOOP.

4.1 SCOOP and MAPA

We first discuss a tool for SymboliC Optimisations of Probabilistic Processes (SCOOP) and its input language the MArkov Process Algebra (MAPA).

4.1.1 SCOOP

SCOOP is a tool which symbolically optimises process-algebraic specifications of probabilistic processes. Reductions on the state space are applied automatically by SCOOP before the complete state space is generated. The goal of the reductions is to reduce the state space of the models as much as possible. The aim of SCOOP is to model efficiently and increase the performance of the analysis of the model.

SCOOP implements the reductions of the MAPA models, but does not perform any analysis. SCOOP is able to export to different model-checkers which can perform the analysis. SCOOP is connected to CADP [15], PRISM [22], MRMC [25] and IMCA [2] (see Figure 4.1). We focus on IMCA as it is the only model-checker that can model-check Markov automata.

SCOOP can be used with the command line or via a web-interface [3]. With this web-interface it is possible to easily configure all options of SCOOP. It is also possible to export the state space of the MAPA model in different formats. The web-interface also connects SCOOP directly with the model-checker IMCA. This means that it is possible to perform the analysis on a MAPA specification directly with IMCA. The web-interface can be used via the browser, which means it can also be used on smart phones for example.

4.1.2 MAPA

Markov automata can be specified by the Markov automata Process Algebra [41] (MAPA). The MAPA language combines data, probability and exit rates to specify an MA. The probabilistic process-algebraic language MAPA is an extension to the language prCRL [26]. This language generalises μCRL [17] by adding a probabilistic

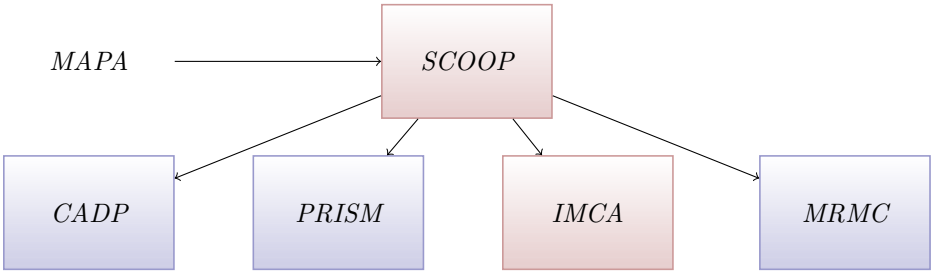


Figure 4.1: Overview of SCOOP. The red squares are the tools we are interested in. The other tools are not used in our research.

choice operator and Markovian rates. Internally, every MAPA specification is translated to an equivalent specification in a restricted form of MAPA. Such specifications are called MLPPEs (Markovian linear probabilistic process equations). Any MAPA specification can be transformed into an MLPPE on which the reductions can take place. It is also easier to generate the state space from an MLPPE.

Language

MAPA consists of process terms p, conforming to the following grammar:

$$p ::= Y(t) \mid c \Rightarrow p \mid p + p \mid \sum_{x:D} p \mid a(t) \sum_{x:D} f : p \mid (\lambda(t)) \cdot p$$

Here Y, is a process name, t a vector of expressions, c a boolean expressions, x a vector of variables ranging over countable type D, a an atomic action and f a real-valued expression yielding values in [0,1]. The last term, including the λ, is the extensions from prCRL to MAPA and represents the Markovian rates. A full system specification consists of a set of processes, an initial state and a data specification. The language also supports parallel composition, action renaming, encapsulation and hiding.

MAPA also supports the $\rightarrow$ notation, which is another notation of a probabilistic summation. The probability is giving before and the next state after the $\rightarrow$, for example $((1/6 \rightarrow X) + (5/6 \rightarrow Y))$.

Example 26. This is an example of a MAPA specification, which is taken from [40]:

$$X = \sum_{n:\{1,2,3\}} write(n) \sum_{i:1,2} \frac{i}{3} : (i = 1 \Rightarrow X + i = 2 \Rightarrow beep \cdot X)$$

This example models a system which chooses and writes a number 1,2 or 3 continuously in a non-deterministically manner. After a write it can also beep with a probability of $\frac{2}{3}$.

Example 27. This is the MAPA specification of the MA in Figure 2.7 on page 15.

$$X = \text{Coin} \cdot \sum_{i:\{1,9\}} \frac{i}{10} : (i = 1 \Rightarrow X + i = 9 \Rightarrow Z)$$
$$Z = (10) \cdot Y + (1) \cdot W$$
$$W = (12) \cdot Z$$
$$Y = \text{Coffee} \cdot X + \text{Tea} \cdot X$$
$$\text{init } X$$

4.1.3 Reduction Techniques of SCOOP

The state space explosion problem is well-known in model-checking. A small extension to a model can create an exponential rise in the number of states that need to be explored. SCOOP applies several reductions to minimize the state space of the model. A smaller state space means that less states have to be explored during the analysis of the model, which means better performance. SCOOP uses state space reduction techniques and MLPPE simplification techniques. The internal overview of how SCOOP applies these techniques is shown in Figure 4.2

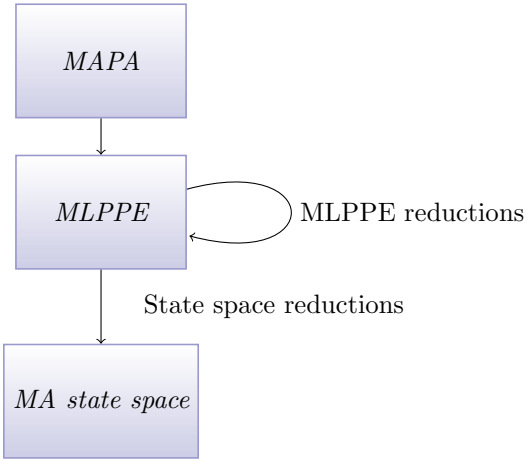


Figure 4.2: Internal overview of SCOOP.

MLPPE simplification techniques do not change the actual state space, but improve readability and speed-up the state space generation. Figure 4.3 shows the overview of the two types of reduction techniques.

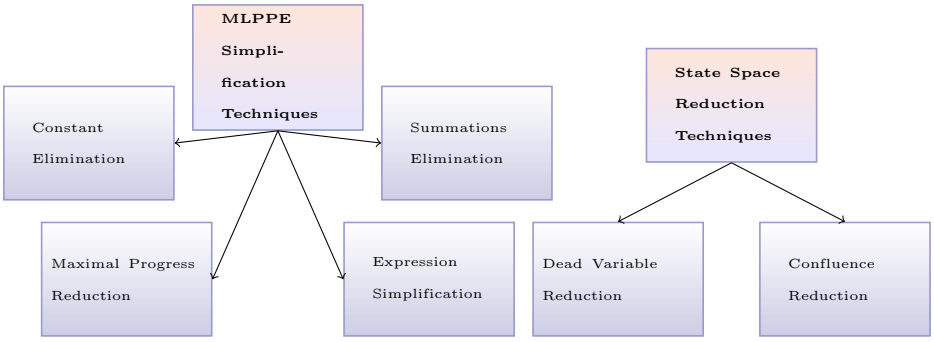


Figure 4.3: Overview SCOOP. The two types of reductions are grouped together.

MLPPE simplification techniques

SCOOP implements four MLPPE simplification techniques [40]. The maximal progress reduction only applies to MAPA specifications where the others all apply to MAPA

and prCRL. The examples are derived from [26].

- *Constant elimination* omits and replaces references to parameters which are never changed.

  Example:

  $X(id\colon Id) = print(id) \cdot X(id)$

  $init\ X(Bob)$

  becomes

  $X = print(Bob) \cdot X$

- *Summation elimination* simplifies non-deterministic choices in which only one of the possible alternatives enables real behaviour. Summation can also be deleted if it sums multiple rate transitions, in this case the rates can be added up.

  Example:

  $X = \sum_{x\colon \{1,2,3\}} x = 3 \Rightarrow print(x) \cdot X$

  $Y = \sum_{x\colon \{1,2,3\}} (3) \cdot Y$

  becomes

  $X = print(3) \cdot X$

  $Y = (9) \cdot Y$

- *Expression simplification* rewrites conditions, action parameters and next state parameters using for instance basic logical identities and the evaluation of functions.

  Example:

  $X = (5 = 2 + 3 \vee x < 5) \Rightarrow beep \cdot Y$

  becomes

  $X = beep \cdot Y$

- *Maximal progress reduction* exploits the maximal progress property by deleting rate transitions when a τ transition can also take place (only used for MAPA specifications).

  Example:

  $X = \tau \cdot X + (3) \cdot X$

  becomes

  $X = \tau \cdot X$

State space reduction techniques

SCOOP implements also two state space reduction techniques.

- *Dead variable reduction* [44] reduces state space by resetting irrelevant variables based on the control flow of an LPPE, while preserving strong bisimulation.
- *Confluence reduction* [43] which basically detects internal τ-transitions that do not influence the system's behaviour. This reduces the number of states that have to be visited and stored during state generation. This reduction is at the moment only available for PA models.

4.2 IMC Analyser (IMCA)

The IMC Analyser [2] is developed by Dennis Guck at the RWTH Aachen. IMCA is a model-checker which supports the analysis of IMCs and MAs. This makes it the only available Markov automata model-checker at the moment.

4.2.1 Input

IMCA expects a few different input files describing an IMC or an MA model. The .imc-file describes the IMC model with its Markovian transitions, interactive transitions, initial states and goal states. The .ma-file describes the MA model. A transitions in a state is modelled with a label and the possible next states, indicated by $*$. Markovian transitions have the label ! followed by the possible next states and the corresponding rates. The interactive transitions have a label and are followed by the possible next state with their probability.

IMCA only considers *closed* MAs and IMCs. An MA or IMC is closed when all its interactive transitions are considered to be τ. MAs or IMCs that are not closed are only useful for parallel composition.

IMCA can also model-check on an IPC. An IPC (Interactive Probability Chain) is like an IMC model where the rates are replaced with probabilities. We do not consider IPCs in this report, but they are close to the PA models described in Section 2.3 on page 11.

Example 28. This is the IMCA specification of the MA in Figure 2.7.

```
#INITIALS
s0
#GOALS
#TRANSITIONS
s3 Coffee
* s0 1.0
s3 Tea
* s0 1.0
s2 !
* s1 12.0
s1 !
* s3 10.0
* s2 1.0
s0 Coin
* s0 0.1
* s1 0.9
```

4.2.2 Functionality

IMCA can model-check on specified properties. These properties state a set of goal states. IMCA can compute the maximum and minimum values for:

- *Time-bounded reachability* [46]: with this you can calculate the probability to reach a set of goal states within a specified amount of time d. An error bound is used to specify the accuracy for the time-bounded reachability. This error bound is 0.000001 by default, but can be set manually. A smaller error bound increases the accuracy of the result, but will take IMCA longer.

- *Unbounded reachability*: with this you can calculate the probability to reach a set of goal states.

- *Expected time*: with which you can calculate the expected time to reach a set of goal states.
- *Long-run average*: with this you can calculate the average amount of time to stay in a set of goal states.

The non-determinism of the MA model causes the minimum and maximum results. When a model does not have non-determinism this means the maximum and minimum value are the same. IMCA also minimizes an IMC model with a bisimulation algorithm as described in [5].

4.2.3 Reachability Condition

When IMCA is used in combination with SCOOP a *reachability condition* can be added to the MAPA specification. This reachability condition specifies the condition that needs to be checked by the IMCA model-checker. This condition specifies the set of goal states we want to reach. With the web-interface of SCOOP [3] it is possible to immediately check this reachability condition with IMCA.

Summary

In this chapter we discussed the important tools for this research. These tools are also shown in the total overview in Figure 1.1. SCOOP and IMCA play an important part in the tool-chain which is used for the analysis of ND-GSPNs. An ND-GSPNs can be translated to an MA. An MA can be specified by MAPA.

MAPA allows the specification of probabilistic models which can be automatically reduced by SCOOP. SCOOP does not provide the state space analysis of its models. The main goal of SCOOP is specifying and reducing probabilistic models up to Markov automata.

IMCA supports the analysis of MAs. This means it can be used to check any MAPA model. It uses a transition file as input, which makes it easy to connect with other tools. IMCA is currently directly connected to SCOOP via a web-interface. MAPA models can immediately be checked on time-bounded reachability, unbounded reachability, expected time and long-run averages. A reachability condition can be added to the MAPA specification. This reachability condition specifies the state you want to reach and is used by IMCA for the analysis. It is possible to model-check a MAPA specification immediately with IMCA with the web-interface of SCOOP.

The tools SCOOP and IMCA can be used for the analysis of ND-GSPNs. If we want this to work we must be able to translate an ND-GSPN to MAPA. The MA in this MAPA model can then be model-checked with IMCA. This is an important part in the overview presented in Figure 1.1.

Part II

Theory

Chapter 5

Non-Deterministic Generalized Stochastic Petri Nets

In this chapter we define the Non-Deterministic Generalised Stochastic Petri Net (ND-GSPN). An ND-GSPN is a new model, which adds non-determinism to the GSPN [28] model. In an ND-GSPN it is possible use non-determinism or weights to resolve conflicts between transitions. This combination was not defined until now. After the definition we formally express the ND-GSPN as an MA. In this chapter we also present how to specify ND-GSPNs in PNML [45].

ND-GSPNs ND-GSPNs are a more general variants of Petri Nets [28]. Like other types of Petri Nets, they consist of *places* and *transitions*. The places can contain *tokens* and the transitions move tokens between places. Each state of an ND-GSPN is therefore given as a *marking*, i.e., a function that assigns a natural number to each place (the number of tokens it contains).

A marking m can evolve into another marking m' if there is a transition to make this happen. This can happen either immediately (by an *immediate transition*) or delayed (by a *rate transition*). The immediate transitions of an ND-GSPN can have a priority, as well as a weight assigned. If in a given state several immediate transitions are enabled, only the ones with the highest priority may fire. Rate transitions are defined to have a priority of 0, and immediate transitions have to be assigned a priority higher than zero, this implies that immediate transition always take precedence over rate transitions (which is often called the *maximal progress property* as described in Section 2.5).

5.1 Definition ND-GSPN

Definition 29 (ND-GSPNs)**.** *An ND-GSPN is a 11-tuple* $G = \langle P, m_0, IMT, PRI, W, RT, R, I, F, RE, MU \rangle$*, where*

- P *is a finite set of* places*;*
- $m_0 \colon P \to \mathbb{N}$ *is the* initial marking
- IMT *is a finite set of* immediate transitions*;*

- $PRI\colon IMT \to \mathbb{N}^{>0}$ *is the* priority function*;*
- $W\colon IMT \to \mathbb{N}^{>0} \cup \{\bot\}$ *is the* weight function*;*
- RT *is a finite set of* rate transitions*;*
- $R\colon RT \to \mathbb{R}_{>0}$ *is the* rate function*;*
- $I \subseteq P \times T$ *is a finite set of* inhibitor arcs*;*
- $RE \subseteq P \times T$ *is a finite set of* reset arcs*;*
- $F\colon (P \times T) \cup (T \times P)$ *is a finite set of* regular arcs*;*
- $MU\colon (F \cup I) \to \mathbb{N}^{>0}$ *is the* multiplicity function*;*

where we use $T = IMT \cup RT$ *to denote all transitions.*

The set of all markings of an ND-GSPN is defined as $\mathcal{M} = P \to \mathbb{N}$. For each immediate transition $t \in IMT$, $PRI(t)$ provides its priority and $W(t)$ its weight. We use $W(i) = \bot$ to denote that t does not have a weight assigned. For each rate transition $t \in RT$, $R(t)$ provides its Markovian rate.

To give the precise semantics of G in terms of an MA, we first introduce some additional notation to deal with arcs and explain when transitions can be fired.

Definition 30 (Notations for arcs)**.** *Let* $G = \langle P, M_0, IMT, PRI, W, RT, R, I, F, RE, MU\rangle$ *be an ND-GSPN. An arc connects places to transitions or vice versa. Given an arc* $a \in F \cup I \cup RE$*, we write* $src(a)$ *for its first element and* $target(a)$ *for its second. Given a transition* $t \in T$*, we denote by* $in(t) = \{a \in F \mid target(a) = t\}$ *the set of all its incoming regular arcs. Similarly, we define* $out(t) = \{a \in F \mid src(a) = t\}$*,* $inhib(t) = \{a \in I \mid target(t) = a\}$ *and* $reset(t) = \{a \in RE \mid target(t) = a\}$*.*

Note that for some arcs $src(a) \in P$ and $target(a) \in T$, while for others $src(a) \in T$ and $target(a) \in P$.

A transition can *fire* if all places associated with its incoming arcs have enough tokens (as indicated by the multiplicity function) and all places associated with its inhibitor arcs have less tokens than the multiplicity of the arc. However, in the presence of multiple priorities some transitions that in principle could fire are disabled nevertheless. If several transitions are enabled, only the one(s) with the highest priorities are actually enabled.

Definition 31 (Enabling)**.** *Given an ND-GSPN* $G = \langle P, M_0, IMT, PRI, W, RT, R, I, F, RE, MU\rangle$ *and a marking* $m \in \mathcal{M}$*, the set of transitions that are potentially enabled from* m *is given by*

$$en_{potentially}(m) = \{t \in T \mid \forall a \in in(t)\,.\, m(src(a)) \geq MU(a) \wedge \forall a \in inhib(t)\,.\, m(src(a)) < MU(a)\}$$

The set of transitions that are actually enabled from m *is given by*

$$en(m) = \{t \in en_{potentially}(m) \mid \forall t' \in en_{potentially}(m)\,.\, PRI(t) \geq PRI(t')\}$$

If several immediate transitions with the same priority are enabled at the same time, any one of them can be chosen non-deterministically. However, if some of them have a weight assigned, those are merged into one transition that has a probabilistic outcome (based on the weights).

If a transition is enabled, it can fire. In that case, the required tokens from the places associated with its incoming arcs are removed, and the places associated with its outgoing arcs gain additional tokens (as indicated by the multiplicity function). *Reset arcs*, $a \in RE$, will empty the place they are connected with. Hence, the transition moves the GSPN from one marking to another.

Definition 32 (Firings)**.** *Given an ND-GSPN* $G = \langle P, M_0, IMT, PRI, W, RT, R, I, F, RE, MU \rangle$ *and two markings* $m, m' \in \mathcal{M}$*, a firing from* m *to* m' *by transition* $t \in T$ *is denoted by* $m \xrightarrow{t} m'$*, if* $t \in en(m)$ *and*

$$\forall p \in P \,.\, m'(p) = \text{ if } (p,t) \in RE \text{ then } 0 \text{ else } (m(p) - MU(p,t) + MU(t,p))$$

If $(p,t) \notin F$ *then* $MU(p,t) = 0$ *(same for* (t,p)*). This means the number tokens in place* p *will not be changed if* p *is not involved in the transition* t*. If there is a reset arc for the place, then in the new marking* p *will be empty.*

We define$\rightarrow^*$ *as the reflexive and transitive closure of the relation* $\rightarrow$ *where* $\rightarrow = \{(m, m') \in \mathcal{M}^2 \mid \exists t \in T \;.\; m \xrightarrow{t} m'\}$.

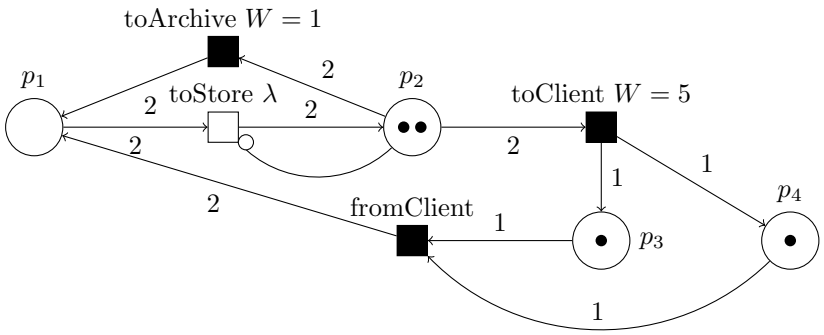


Figure 5.1: Non-Deterministic General Stochastic Petri Net. The multiplicity is showed on each arc.

Example 33. Figure 5.1 shows an ND-GSPN which models a simple library. This model has a combination of exponentially delayed transitions and immediate transitions. The immediate transitions toArchive and toClient have a weight, but fromClient has no weight. In the initial state the transition toArchive can fire with probability 1/6 and toClient with probability 5/6, where fromClient can fire non-deterministically. The Markovian transition toStore with rate λ can only fire when no immediate transitions are enabled.

In this example we do not show an example of the reset arc. The reset arc has a diamond at the side of the transition as shown in Figure 5.2. When the transition fires p_0 will become empty.

Figure 5.2: Reset arc example.

5.1.1 Combination Weights and Non-Determinism

An ND-GSPN can resolve conflicts between immediate transitions with non-determinism and with weights. In this section we show that the use of non-determinism does not affect the behaviour of the transitions that do have a weight.

We illustrate this with an example with three enabled transition where two have a weight and one can be taken non-deterministically. The ND-GSPN is shown in Figure 5.3. Transition t_1 has weight q and transition t_2 has weight r. Transition t_3 does not have a weight, but this can be interpreted that the weight can have any value x. We solve this variable weight x with non-determinism. We show that for each scenario with weight x, solving the non-determinism with probability does not affect the relation between t_1 and t_2.

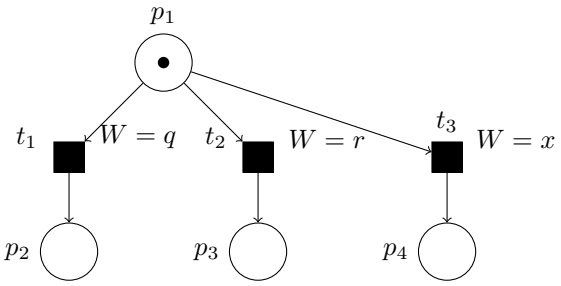


Figure 5.3: ND-GSPN with enabled weighted and unweighted transitions.

With variable weight x we can define the probabilities that we expect in Figure 5.3. These probabilities are:

$$P(p_2) = \frac{q}{q+r+x}$$
$$P(p_3) = \frac{r}{q+r+x}$$
$$P(p_4) = \frac{x}{q+r+x}$$

The non-deterministic choice between t_1 and t_2 or t_3 can be solved with probability. This can be interpreted that we have a probabilistic choice between t_1 or t_2 and t_3, where the choice between t_1 or t_2 solely depends on the weights q and r. This situation is shown as an MA in Figure 5.4, where we solve the non-determinism with the probabilities P_1 and P_2. In this MA each state corresponds with a place from Figure 5.3, where it has one token.

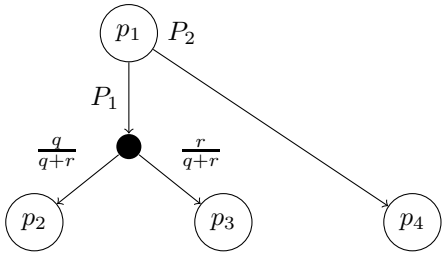


Figure 5.4: MA of Figure 5.3, where probabilities P_1 and P_2 solve the non-determinism.

We take $P_2 = \frac{x}{q+r+x}$, which means that $P(p_4) = \frac{x}{q+r+x}$, which is exactly what we want. This also means that:

$$P_1 = (1 - P_2) = 1 - \frac{x}{q+r+x} = \frac{q+r+x}{q+r+x} - \frac{x}{q+r+x} = \frac{q+r+x-x}{q+r+x} = \frac{q+r}{q+r+x}$$

When we fill in P_1 we get:

$$P(p_2) = P_1 \cdot \frac{q}{q+r} = \frac{q+r}{q+r+x} \cdot \frac{q}{q+r} = \frac{q}{q+r+x}$$

and

$$P(p_3) = P_1 \cdot \frac{r}{q+r} = \frac{q+r}{q+r+x} \cdot \frac{r}{q+r} = \frac{r}{q+r+x}$$

The results are the same as the probabilities defined earlier. This means that the non-determinism of t_3 does not affect the relation between t_1 and t_2. This means each scenario where x has a weight can be achieved by resolving the non-determinism properly.

5.2 ND-GSPN expressed as an MA

We are now ready to define the semantics of an ND-GSPN in terms of an MA. We described the MA model in Section 2.6 on page 14. As stated earlier, transitions without a weight can fire non-deterministically. The transitions with weights are merged together in a single τ transition. The next state of this transition is probabilistically distributed according to the weights of the constituent transitions.

Definition 34 (ND-GSPN semantics)**.** *Given an ND-GSPN* $G = \langle P, M_0, IMT, PRI, W, RT, R, I, F, RE, MU \rangle$*, its semantics is given by the MA* $M = \langle S, m_0, A, \hookrightarrow, \rightsquigarrow \rangle$*, where*

- $S = \{m' \in \mathcal{M} \mid m_0 \xrightarrow{t}{}^{*}\ m'\}$
- $A = \{\tau\}$
- $\hookrightarrow$ *is such that, for every* $m, m' \in S, \mu \in \mathsf{Distr}(S)$*,*

$$\begin{aligned} m \overset{\tau}{\hookrightarrow} \mathbb{1}_{m'} \quad &\iff \quad \exists t \in IMT \,.\, W(t) = \bot \wedge m \xrightarrow{t} m' \\ m \overset{\tau}{\hookrightarrow} \mu \quad &\iff \quad \exists t \in IMT \,.\, W(t) \neq \bot \wedge \\ &\qquad \forall m' \in S \,.\, \mu(m') = \frac{\sum \left\{ \left| W(t) \in \mathbb{N} \mid t \in IMT \wedge W(t) \neq \bot \wedge m \xrightarrow{t} m' \right| \right\}}{\sum \left\{ \left| W(t) \in \mathbb{N} \mid t \in IMT \wedge W(t) \neq \bot \wedge \exists m'' \in S \,.\, m \xrightarrow{t} m'' \right| \right\}} \end{aligned}$$

- $\rightsquigarrow$ *is such that, for every* $m, m' \in S$*,*

$$m \overset{\lambda}{\rightsquigarrow} m' \quad \iff \quad \lambda = \sum \left\{ \left| R(t) \in \mathbb{R} \mid t \in RT \wedge m \xrightarrow{t} m' \right| \right\} \wedge \lambda > 0$$

The sums in the above definition use the following convention: $\sum \varnothing = 0$*.*

Note that the rate from a marking m to a marking m' is determined by the sum of all enabled rate transitions from m to m'. The MA only has the label τ, which means it is a closed MA.

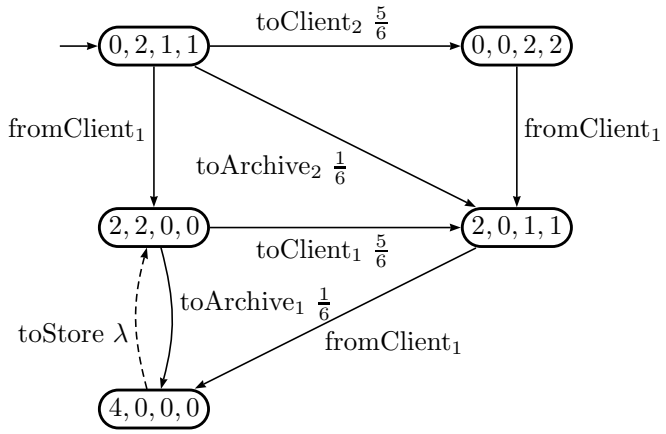


Figure 5.5: Markov Automaton of Figure 5.1

Example 35. Figure 5.5 shows the underlying Markov automaton of the ND-GSPN in Figure 5.1. Note that we added the transition labels to the diagram for illustrating purposes, but in practice these labels are all translated to τ for each distribution. This means each non-deterministic transition gets a τ label and each probabilistic choice gets a τ label. We used numbers to identify the different transition for each state. For example in the initials state there are two transition, where fromClient can be taken non-deterministically and there is a distribution between toClient and toArchive.

5.3 Specification of ND-GSPNs

In this section we define how to specify an ND-GSPN in the Petri Net Markup Language (PNML) [45].

5.3.1 PNML

There are many different tools which support the modelling of Petri Nets. To make it easy to exchange models between tools, the Petri Net Markup Language (PNML) was developed [45]. PNML is an official ISO standard ISO/IEC 15909, that can be used to specify Petri Nets in an XML-based syntax. PNML only supports the modelling of normal Petri Nets, but can easily be extended as it is XML-based.

PNML Syntax

A PNML file describes the places, transitions and arcs of the Petri Net model. For each of them you can define their name and id. Places also have an initial marking. An arc is connected to a place and a transition and has an inscription tag which is the multiplicity of the arc.

The specification of the rate, weights and priorities of the net is not defined at this moment. However, adding these properties as extra elements for the transitions is trivial. We give an example of a basic PNML file. In the next section we will formally define how the PNML of our ND-GSPN should look like.

Example 36. We show a Basic PNML example with a place, a transition and an arc. This simple Petri net is shown in Figure 5.6.

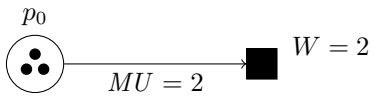


Figure 5.6: Basic Petri net with a place, transition and arc.

Next we show the PNML file of Figure 5.6. Note that the transition is immediate and has weight 2, which is represented in the PNML with the timed value false and rate value 2.

```
<pnml>
<net id="Net-One" type="P/T net">
 <place id="P0">
  <name>
   <value>P0</value>
  </name>
  <initialMarking>
   <value>3</value>
  </initialMarking>
 </place>
 <transition id="T0">
  <name>
   <value>T0</value>
  </name>
  <rate>
   <value>2.0</value>
  </rate>
  <timed>
   <value>false</value>
  </timed>
  <priority>
   <value>1</value>
  </priority>
 </transition>
 <arc id="P0toT0" source="P0" target="T0">
  <inscription>
   <value>2</value>
  </inscription>
  </arc>
</net>
</pnml>
```

Properties to Verify

PNML does not support the specification of the properties that we want to verify on the Petri Net. The properties should state what state of the Petri Net we want to reach. This means it specifies the number of tokens we want in each place.

These properties are not specified in PNML, but can be specified with SCOOP [40]. With SCOOP it is possible to define a reachability condition as described in Section 4.2.3 on page 31. This reachability condition can be used to specify a marking in the net, which will be translated to a set of goal states by SCOOP.

5.3.2 ND-GSPN as PNML

In this section we discuss how the PNML of an ND-GSPN should look like. We specify what attributes and elements for each place, transition and arc are required or optional. We provide a Document Type Definition (DTD). This DTD gives a formal definition for the specification of ND-GSPNs. We discuss the DTD for the different elements of the ND-GSPN.

Document Type Definition (DTD)

```
<!DOCTYPE pnml [
 <!ELEMENT pnml (net)>
  <!ELEMENT net (place|transition|arc)*>
   <!ATTLIST net id ID #REQUIRED >

  <!ELEMENT place (name?, initialMarking?)>
   <!ATTLIST place id ID #REQUIRED >
   <!ELEMENT initialMarking (value | text)>

  <!ELEMENT transition (name?, timed?, rate?, priority?)>
   <!ATTLIST transition id ID #REQUIRED >
   <!ELEMENT timed (value | text)>
   <!ELEMENT rate (value | text)>
   <!ELEMENT priority (value | text)>

  <!ELEMENT arc (inscription?, probability?)>
   <!ATTLIST arc id ID #REQUIRED
                 source CDATA #REQUIRED
                 target CDATA #REQUIRED
                 type (normal | inhibitor | reset) #IMPLIED>
   <!ELEMENT inscription (value | text)>
   <!ELEMENT probability (value | text)>

  <!ELEMENT name (value | text)>
   <!ELEMENT value (#PCDATA)>
   <!ELEMENT text (#PCDATA)>
]>
```

Note that this DTD is more strict than needed in practice. The DTD gives no room for other elements or attributes, but our tool accepts PNML with other tags or attributes. For example, many PNML definitions use page tags between the net and the place, transition or arc tags. In our tool these pages will be ignored. Another example are the graphic elements which specify how the net is presented in a tool. These kind of elements are also ignored by the tool.

DTD only allows us to specify ordered child elements. However, in our tool the order of the child elements is not important (for example for transition the order of the name, timed and rate elements is not important).

Net The PNML file must contain one *net* element. This element must contain an id attribute which contains the identifier of the net. The children of the net are the place, transition and arc elements.

Place A *place* must have an id attribute that contains the identifier of the place. A place may also have a *name* element which has a value or text element which contains the name of the place. The identifier will be used as name if there is no name tag available.

A place may also have an *initialMarking* element. This element specifies the number of tokens initially at the place. If this element is not available this indicates that the place is initially empty.

Transition A *transition* must have an id attribute that contains the identifier of the transition. A transition may also have a name element which has a value or text element which contains the name of the transition.

A transition may have a *timed* element. This indicates if the transition is immediate or rate. The timed element contains a value or text elements which can have the value true or false to indicate if the transition is rate transition.

The transition may also have a *rate* element. This rate element has a text or value element which contains the rate of the transition. If the transition is timed this tag contains the rate. The rate can also be interpreted as a weight if the transition is not timed.

The transition may also have a *priority* element. This element has a text or value element which contain the priority of the transition. For timed transition the priority is always interpreted as 0. For immediate transition the priority is by default 1.

Arc An *arc* must have an id attribute which contains the identifier of the transition. An arc must also have a source and a target which corresponds to the id of the source and target of the arc. An arc can have a *type* element which has a value attribute. This attribute specifies the type of the arc. By default the arc is typed normal. Other types can be inhibitor or reset.

An arc may have an *inscription* element. This element has a text or value element which contains the multiplicity of the arc. This is by default 1 for normal arcs and inhibitor arcs. The inscription for reset arcs is ignored.

Later, in Chapter 7, we describe the semantics and theory for the probabilistic arc extension for ND-GSPNs. We now only state how to model this extension, which means adding a probability to an arc. An arc can also have a *probability* element. This element has a text or value element which contains the probability that the arc will deliver its token(s). Arcs which have a probability are always probabilistic arcs.

Chapter 6

Translation ND-GSPN to MAPA

In this chapter we translate the ND-GSPN model to a MAPA specification [41]. MAPA is a process algebra language which is used to model Markov Automata. MAPA is used as input language for the tool SCOOP [40]. SCOOP automatically reduces its models and is connected to IMCA [2] which performs the analysis of the models. Thus translating the ND-GSPN to MAPA makes it possible to model check on the ND-GSPN. MAPA is described in more detail in Section 4.1.2.

6.1 ND-GSPN to MAPA

We specify an ND-GSPN in terms of MAPA [41]. The specification consists of one process which has the transitions as process terms. This process has a variable for each place. A valuation of these variables corresponds to a marking of the ND-GSPN. The initial marking is thus the initial state of the process.

Given an ND-GSPN $G = \langle P, M_0, IMT, PRI, W, RT, R, I, F, RE, MU \rangle$. Then the corresponding MAPA specification is a process $X(\overline{n})$, where $\overline{n}$ is a vector of integer variables of length $|P|$ indexed by P, with the initial state $X(M_0)$. This means n_p is the number of tokens in place p. The behaviour of X is given by a set of summands derived from the transitions of G. The behaviour of X follows the ND-GSPN behaviour described in sections 5 and 5.2. Process X is specified in the following subsections which distinguish rate transitions and unweighted and weighted immediate transitions.

Note that the statements $\bigwedge$ and $\bigvee$ are not part of the MAPA syntax, but in the actual MAPA specification we get all the terms specified by these statements.

6.1.1 Rate Transitions

For each rate transition $t \in RT$, X has a summand

$$cr_t \Rightarrow R(t) \cdot X(\overline{n}_t)$$

with the condition:

$$cr_t = \bigwedge_{a \in in(t)} src(a) \geq MU(a) \quad \wedge \quad \bigwedge_{a \in inhib(t)} src(a) < MU(a)$$

Note that this and the following conditions are completely in MAPA as syntax and not just the evaluation of the terms in the condition. The conditions we specify are an abbreviation for the actual conditions in the MAPA specification. During the state space generation the conditions are evaluated in each individual state, because they depend on the number of tokens in the places in the current state.

The next state $\overline{n}_t = (n'_{p_1}, n'_{p_2}, .., n'_{p_n})$ for all $p \in P$ is defined for each n'_p by:

$$n'_p = \text{if } (p,t) \in RE \text{ then } 0 \text{ else } (n_p - MU(p,t) + MU(t,p))$$

If $(p,t) \notin F$ then $MU(p,t) = 0$ (same for (t,p)). This means the place variable p will not be changed if it is not involved in the transition t. If there is a reset arc for the place, then in the new marking it will be empty.

6.1.2 Unweighted Immediate Transitions

For each unweighted immediate transition $t \in IMT$ with $W(t) = \bot$, X has a summand

$$ci_t \Rightarrow \tau \cdot X(\overline{n}_t)$$

with the condition:

$$ci_t = \quad cr_t \quad \wedge \bigwedge_{t' \in IMT.PRI(t') > PRI(t)} \neg(cr_{t'})$$

where cr_t and $\overline{n}_t$ are as above. This way, unweighted transitions are only enabled if none of the transitions with a higher priority are also enabled. Note that this is not needed for rate transitions, due to the maximal progress property.

6.1.3 Weighted Immediate Transitions

Let $t_p = \{t \mid t \in IMT \wedge W(t) \neq \bot \wedge PRI(t) = p\}$ be the set of weighted transitions with priority p. For each priority p such that $|t_p| > 0$, X has a summand

$$cs_p \Rightarrow \tau \cdot \sum_{t:t_p} \frac{\text{if } ci_t \text{ then } W(t) \text{ else } 0}{\text{totalWeight}_{t_p}} : X(\overline{n}_t)$$

with the condition:

$$cs_p = \quad \bigvee_{t \in t_p} ci_t$$

where $\overline{n}_t$ and ci_t are as defined above. Note that each transition only has a non-zero probability if it is enabled. Hence, the total weight is also defined conditionally, by

$$\text{totalWeight}_{t_p} = \sum_{t \in t_p} (\text{if } ci_t \text{ then } W(t) \text{ else } 0)$$

The totalWeight statement is used in MAPA as an expression, which is defined by a *count* statement. This count statement contains tuples of a boolean with an integer value. It sums the integers of the booleans that are true. This means the tuples are specified by ci_t as boolean with $W(t)$ as value. This way the totalWeight of all the enabled transitions can be calculated depending on the current state of the model. This means that in each state of the model the totalWeight is evaluated.

6.1.4 Summary

Definition 37 (ND-GSPN semantics)**.** *Given an ND-GSPN* $G = \langle P, M_0, IMT, PRI, W, RT, R, I, F, RE, MU \rangle$*, its total behaviour can be described by the MAPA specification:*

$$
\begin{aligned}
X(M_0) = \quad & \sum_{t \in RT} && cr_t \Rightarrow R(t) \cdot X(\overline{n}_t) && + \\
& \sum_{\substack{t \in IMT \\ W(t) = \bot}} && ci_t \Rightarrow \tau \cdot X(\overline{n}_t) && + \\
& \sum_{\substack{p \in \mathbb{N} \\ |t_p| > 0}} && cs_p \Rightarrow \tau \cdot \sum_{t : t_p} \frac{\text{if } ci_t \text{ then } W(t) \text{ else } 0}{totalWeight_{t_p}} : X(\overline{n}_t)
\end{aligned}
$$

Example 38. The MAPA specification of the ND-GSPN in Figure 5.1 on page 35 is $X(0,2,1,1) = p$ with $\overline{n} = \{p_1, p_2, p_3, p_4\}$ and

$$
\begin{aligned}
p = \; & \\
& (p_1 \geq 2 \wedge p_2 < 1) \Rightarrow (\lambda) \cdot X(p_1' = p_1 - 2, p_2' = p_2 + 2) \; + \\
& (p_3 \geq 1 \wedge p_4 \geq 1) \Rightarrow \tau \cdot X(p_1' = p_1 + 2, p_3' = p_3 - 1, p_4' = p_4 - 1) \; + \\
& (p_2 \geq 2) \Rightarrow \tau \cdot \; (\\
& \qquad (1/6 \rightarrow X(p_1' = p_1 + 2, p_2' = p_2 - 2)) \; + \\
& \qquad (5/6 \rightarrow X(p_2' = p_2 - 2, p_3' = p_3 + 1, p_4' = p_4 + 1)) \;)
\end{aligned}
$$

In this example the immediate transitions with a weight are always enabled at the same time. This means that the totalWeight is always the same in this case. In practice the totalWeight would be a constant defined as: $totalWeight = count(p_2 \geq 2, 1, p_2 \geq 2, 5)$, which in the case of $p_2 \geq 2$ is always evaluated to 6.

6.2 Translation Proof

In this section we prove the correctness of the translation from ND-GSPN to MAPA. We show that the MA we specified in Section 5.2 is *isomorphic* to the MA under the MAPA specification of Section 6.1. We prove this by comparing the two MAs.

Proof Structure Both MAs are constructed from the same ND-GSPN. We compare how the MAs are defined from the original ND-GSPN. To prove that the MAs are isomorphic we show that the transitions of an ND-GSPN lead to the same transitions in both MAs.

Both MAs have the reachable markings as their state space. The states can be mapped using the function $f(m) = X(m)$. This function is *bijective*, which means *injective* and *surjective*, which means that each state in each MA can be mapped to exactly one unique state in the other MA. We write m instead of $X(m)$ in this proof, as this makes the comparison between the two MAs clearer. The action labels that both MAs use are the same, as they both only use the label τ for their interactive transitions. What we need to show is that the transitions between the states are also equal. We show that the rate and immediate transitions in the ND-GSPN will result in the same Markovian and interactive transitions in both MAs.

6.2.1 Preliminaries

We define ND-GSPN $G = \langle P, m_0, IMT, PRI, W, RT, R, I, F, RE, MU \rangle$ where $T = IMT \cup RT$ denotes all transitions. MA_1 is the MA under the MAPA specification which is obtained using Definition 12 in [41]. Given a MAPA specification $X(\overline{n})$ that is a result from the translation of G by Definition 37, its semantics are given by $MA_1 = \langle S, m_0, A, \hookrightarrow_1, \rightsquigarrow_1 \rangle)$. $X(\overline{n})$ is already in the MLPPE format as defined in Definition 12 in [41], this means that we can use this Definition 12 to construct the transitions of MA_1. The semantics of G are given by $MA_2 = \langle S, m_0, A, \hookrightarrow_2, \rightsquigarrow_2 \rangle$ as in Definition 34. We use $S = \mathcal{M}$ as we cannot conclude that the reachable markings for both MAs are the same at this time. If we prove that the transitions in both MAs are the same, this means that the reachable markings in both MAs will also be the same. We use $\cong$ to denote the isomorphic relation.

For the valuation of the conditions cr, ci, cs and $totalWeight_{t_p}$ in state $m \in S$ we use $cr(m)$, $ci(m)$, $cs(m)$ and $totalWeight_{t_p}(m)$. For the next state valuation from $m \in S$ we use $\overline{n}(m) \in S$.

6.2.2 Rate Transitions

First we prove that the Markovian transitions ($\rightsquigarrow$) in the two MAs are the same. In MA_1 the Markovian transitions are derived from the rate summands. These rate summands are translated from the rate transitions in G. This means indirectly that the Markovian transitions in MA_1 are derived from the rate transitions, $t \in RT$, from G. In MA_2 the Markovian transitions are directly derived from the rate transitions of G. We show that the rate summands construct the same Markovian transitions in MA_1 as the rate transitions in MA_2.

Proof

First we specify two lemmas that are needed to prove that the Markovian transitions of the MAs are the same.

Lemma 1 (Rate Conditions)**.** *For transition $t \in RT$ and $m \in S$:*

$$t \in en_{potentially}(m) \quad \Longleftrightarrow \quad cr_t(m)$$

Proof. We compare Definition 31 ($t \in en_{potentially}(m)$) with Definition 37 ($cr_t(m)$).

From Definition 37, $cr_t(m)$:

$$\bigwedge_{a \in in(t)} src(a) \geq MU(a) \quad \wedge \quad \bigwedge_{a \in inhib(t)} src(a) \leq MU(a)$$

From Definition 31, $t \in en_{potentially}(m)$:

$$\forall a \in in(t) \,.\, m(src(a)) \geq MU(a) \;\wedge\; \forall a \in inhib(t) \,.\, m(src(a)) \leq MU(a)$$

We can immediately see that both computations are equivalent. □

Lemma 2 (Next State)**.** *For $t \in T$, $m, m' \in S$ and $m \xrightarrow{t} m'$ then*

$$m' = \overline{n}_t(m)$$

Proof. The firing of $m \xrightarrow{t} m'$ (Definition 32) is compared with $\overline{n}_t(m)$ (Definition 37).

From Definition 32, $m \xrightarrow{t} m'$:

$$\forall p \in P \,.\, m'(p) = \text{ if } (p,t) \in RE \text{ then } 0 \text{ else } (m(p) - MU(p,t) + MU(t,p))$$

From Definition 37, $\overline{n}_t(m)$:

$$\forall p \in P \,.\, n'_p = \text{if } (p,t) \in RE \text{ then } 0 \text{ else } (n_p - MU(p,t) + MU(t,p))$$

The computations in Definition 32 decreases or increases the number of tokens of the places involved in the transition. Definition 37 does the same. Both definitions do not change the number tokens when the place is not involved in the transition. The reset arcs empty a place in both definitions. This means both computations are equivalent. □

Theorem 1 (Isomorphic Rate Transitions)**.** *For MA_1 and MA_2 and states $m, m' \in S$:*

$$m \overset{\lambda}{\rightsquigarrow}_1 m' \iff m \overset{\lambda}{\rightsquigarrow}_2 m'$$

Proof. The rate transitions $t \in RT$ with $m, \overline{n}_t(m) \in S$ are translated to the summands:

$$cr_t \Rightarrow R(t) \cdot X(\overline{n}_t)$$

These summands are used to construct the Markovian transition in MA_1. These summands are directly translated from the rate transitions $t \in RT$, therefore we can sum over these rate transitions to construct the Markovian transitions of MA_1. We use Definition 12 in [41] to construct the MA from these summands (or rate transitions indirectly).

For MA_1 this means that $m \overset{\lambda}{\rightsquigarrow}_1 m'$ if and only if $\lambda > 0$ and

$$\lambda = \sum_{t \in RT \ \wedge \ cr_t(m) \ \wedge \ \overline{n}_t(m) = m'} R(t)$$

In MA_2, as defined in Definition 34, we have $m \overset{\lambda}{\rightsquigarrow}_2 m'$ only if $\lambda > 0$ and

$$\lambda = \sum_{t \in RT \ \wedge \ m \xrightarrow{t} m'} R(t)$$

This means we need to prove that for $t \in RT$:

$$m \xrightarrow{t} m' \quad \iff \quad cr_t(m) \ \wedge \ \overline{n}_t(m) = m'$$

The above follows directly from Lemmas 1 and 2. This means that λ is calculated the same way in both MAs. This means that $m \overset{\lambda}{\rightsquigarrow}_1 m'$ if and only if $m \overset{\lambda}{\rightsquigarrow}_2 m'$. □

6.2.3 Interactive Transitions

We now focus on the interactive transitions ($\hookrightarrow$) of the MAs. We show that the immediate transitions of G result in the same interactive transitions in both MAs. We do this in two steps: weighted and unweighted immediate transitions. This is done because in the expression of MA_2 and in the MAPA specification we also make a difference in weighted and unweighted immediate transitions.

Unweighted Immediate Transitions

For the unweighted transitions we do the same as for the rate transitions. We show that the transitions $t \in IMT$ with $W(t) = \bot$ lead to the same interactive transitions in both MAs.

We specify a new Lemma for this proof.

Lemma 3 (Immediate Conditions)**.** *For $t \in IMT$:*

$$t \in en(m) \quad \iff \quad ci_t(m)$$

Proof. The condition ci_t adds an extra check to manage the priorities. This extra check is also in Definition 31.

From Definition 31, $t \in en(m)$:

$$\forall t' \in en_{\text{potentially}}(m) \,.\, PRI(t) \geq PRI(t')$$

From Definition 37, $ci_t(m)$:

$$cr_t(m) \ \wedge \bigwedge_{t' \in IMT.PRI(t') > PRI(t)} \neg(cr_{t'}(m))$$

The first part of the condition is equivalent as shown in Lemma 1. The second part states in both definitions that there can not be a transition enabled with a higher priority. Hence, this part is also equivalent. □

Theorem 2 (Unweighted Transitions)**.** *For $m, m' \in S$ with $m \xrightarrow{t} m'$ for $t \in IMT$ and $W(t) = \bot$:*

$$m \overset{\tau}{\hookrightarrow}_1 \mathbb{1}_{m'} \quad \iff \quad m \overset{\tau}{\hookrightarrow}_2 \mathbb{1}_{m'}$$

Proof. The unweighted immediate transitions $t \in IMT$ with $W(t) = \bot$ and $m, \overline{n}_t(m) \in S$ are translated to the summands:

$$ci_t \Rightarrow \tau \cdot X(\overline{n}_t)$$

We use Definition 12 in [41] to construct the interactive transitions of MA_1 from the summands of the MAPA specification. The summands do not specify any probability. This means that the transitions can be taken with probability 1 (Dirac). A transition occurs in MA_1 if there is a summand in the MAPA specification. These summands are specified by the immediate transitions $t \in IMT$ with $W(t) = \bot$, therefore we sum over these transitions to derive the interactive transition of MA_1.

This means $m \overset{\tau}{\hookrightarrow}_1 \mathbb{1}_{m'}$ if and only if

$$\exists t \in IMT \,.\, W(t) = \bot \wedge ci_t(m) \ \wedge \ \overline{n}_t(m) = m'$$

In MA_2, as defined in Definition 34, we have $m \overset{\tau}{\hookrightarrow}_2 \mathbb{1}_{m'}$ if and only if

$$\exists t \in IMT \,.\, W(t) = \bot \wedge m \xrightarrow{t} m'$$

This means we need to prove that for $t \in IMT$ with $W(t) = \bot$:

$$m \xrightarrow{t} m' \quad \iff \quad ci_t(m) \ \wedge \ \overline{n}_t(m) = m'$$

The above is proven by Lemmas 2 and 3. This means that the immediate transitions $t \in IMT$ with $W(t) = \bot$ lead to the same interactive transitions in both MAs. □

Weighted Immediate Transitions

In this section we show that in both MAs the weighted immediate transitions also result in the same interactive transitions in both MAs. We show how the weighted immediate transitions in G lead to interactive transitions in MA_1 and that they are equivalent with the interactive transition in MA_2. MA_1 builds its interactive transitions from the transitions in G, where MA_2 builds them from the possible goal marking. We show that these approaches are equivalent.

Theorem 3 (Weighted Transitions)**.** *For MA_1 and MA_2 with $m \in S$ and $\mu \in \mathsf{Distr}(S)$:*

$$m \overset{\tau}{\hookrightarrow}_1 \mu \quad \iff \quad m \overset{\tau}{\hookrightarrow}_2 \mu$$

Proof. First we show how the summands in the MAPA specification lead to interactive transitions in MA_1. The weighted immediate transition $t \in IMT$ with $W(t) \neq \bot$ are grouped by their priority. For each priority $p \in \mathbb{N}^{>0}$ with $|t_p| > 0$ the transitions are translated to the summands:

$$cs_p \Rightarrow \tau \cdot \sum_{t:t_p} \frac{\text{if } ci_t \text{ then } W(t) \text{ else } 0}{totalWeight_{t_p}} : X(\overline{n}_t)$$

We again use Definition 12 in [41] to derive the transitions from these summands. The summands are defined by the transition in t_p and thus we can sum over these transitions to construct the interactive transitions in MA_1. This means $m \overset{\tau}{\hookrightarrow}_1 \mu$ if and only if

$$cs_p(m) \wedge \forall t \in t_p \,.\mu(\overline{n}_t(m)) = \sum_{t' \in t_p . \overline{n}_t(m) = \overline{n}_{t'}(m)} \frac{\text{if } ci_{t'}(m) \text{ then } W(t') \text{ else } 0}{totalWeight_{t_p}(m)}$$

Each priority gets its own summand in the MAPA specification. According to the definition of cs_p only one of them can be enabled at the time. This means that in state $m \in S$ only the enabled transitions with the highest priority end up in MA_1.

In MA_2 the interactive transitions are not grouped by priority, but the firing definition causes that only the highest enabled transitions are used in the computations of the probabilities of μ.

This means that, as defined in Definition 34, $m \overset{\tau}{\hookrightarrow}_2 \mu$ if and only if

$$\exists t \in IMT \,.\, W(t) \neq \bot \wedge$$
$$\forall m' \in S \,.\, \mu(m') = \frac{\sum \left\{ \left| W(t) \in \mathbb{N} \mid t \in IMT \wedge W(t) \neq \bot \wedge m \xrightarrow{t} m' \right| \right\}}{\sum \left\{ \left| W(t) \in \mathbb{N} \mid t \in IMT \wedge W(t) \neq \bot \wedge \exists m'' \in S \,.\, m \xrightarrow{t} m'' \right| \right\}}$$

In $\hookrightarrow_2$ the condition $\exists t \in IMT$ with $W(t) \neq \bot$ corresponds with the conditions cs_p as they are the disjunction of all $t \in IMT$ with $W(t) \neq \bot$.

This means we need to prove for $t \in IMT$ with $W(t) \neq \bot$ that $\mu(\overline{n}_t(m)) = \mu(m')$. To prove that this is equivalent we split the fractions in both definitions and compare them separately. For the first part we rewrite the sum in Definition 34.

$$\sum_{t' \in t_p . \overline{n}_t(m) = \overline{n}_{t'}(m)} (\text{if } ci_{t'}(m) \text{ then } W(t') \text{ else } 0) = \sum_{t \in IMT . W(t) \neq \bot \wedge m \xrightarrow{t} m'} W(t)$$

The left side sums the weights of all enabled transitions from m to $\overline{n}_t(m)$. As seen earlier in Lemmas 3 and 2 $m \xrightarrow{t} m'$ if and only if $ci_t(m)$ and $\overline{n}_t(m) = m'$. The right

side is taken and rewritten from Definition 34 and sums the weights of all enabled transitions from m to m'. This means both sides are equal.

$$\sum_{t \in t_p} (\text{if } ci_t(m) \text{ then } W(t) \text{ else } 0) = \sum_{t \in IMT. W(t) \neq \bot \wedge \exists m'' \in S. m \xrightarrow{t} m''} W(t)$$

The right side is taken from Definition 34 and sums the weights of all enabled transitions in m. The definition of $totalWeight_p$ is the sum of all weights where $ci_t(m)$ holds, which means all enabled transitions (as $ci_t(m)$ holds if and only if $t \in en(m)$(Lemma 3)). This means both sides are equal.

We showed for $t \in IMT$ with $W(t) \neq \bot$ that $\mu(\overline{n}_t(m)) = \mu(m')$. This proves that for $t \in IMT$ with $W(t) \neq \bot$:

$$m \xhookrightarrow{\tau}_1 \mu \quad \iff \quad m \xhookrightarrow{\tau}_2 \mu$$

□

Summary Interactive Transitions

Theorem 4 (Isomorphic Interactive Transitions)**.** *For MA_1 and MA_2 with $m \in S$ and $\mu \in \mathsf{Distr}(S)$:*

$$m \xhookrightarrow{\tau}_1 \mu \iff m \xhookrightarrow{\tau}_2 \mu$$

Proof. Based on theorem 2 and 3 we can conclude that all interactive transitions in MA_1 and MA_2 are equivalent. We showed that all immediate transitions in G lead to the same interactive transition in both MAs. □

6.2.4 Summary

Theorem 5 (Isomorphic MA_1 and MA_1)**.**

$$MA_1 \cong MA_2$$

Proof. Based on the Theorems 1 and 4, we can now conclude that both MAs are isomorphic. The transitions in $\hookrightarrow$ and $\rightsquigarrow$ are the same. This means we can now also conclude that the state spaces S of the reachable markings are isomorphic. This means the function $f(m) = X(m)$ can be used as bijective mapping of the state spaces from MA_2 to MA_1. □

Chapter 7

Probabilistic Arcs

In this chapter we present an extension to ND-GSPNs. Until now we were able to define weights for transitions to resolve the conflicts between transitions. The probabilities are calculated globally by checking the weights of all enabled transitions.

We present the *probabilistic arc* which can make the result of a transition distributed probabilistically. This means we can define probability more local. A transition can have one or more probabilistic arcs. If such a transition fires the resulting marking is a probabilistic summation over all probabilistic arcs.

An example of the use of the probabilistic arc is shown in Figure 7.1.

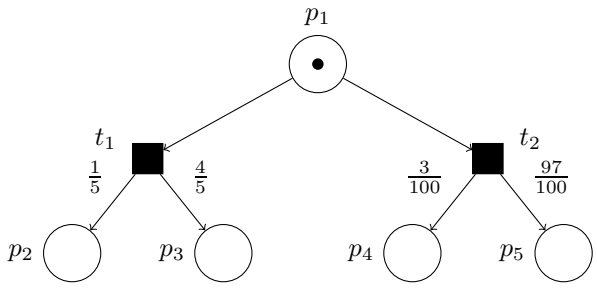


Figure 7.1: ND-GSPN with probabilistic arcs.

Example 39. In Figure 7.1 both transitions can be taken non-deterministically. Transition t_1 has two probabilistic arcs, with probabilities of $\frac{1}{5}$ and $\frac{4}{5}$. This means if t_1 fires we get a token in p_2 with probability of 0.2 or in p_3 with probability of 0.8. If t_2 fires we get a token in p_4 with probability 0.03 and in p_5 with probability 0.97.

This does not introduce new behaviour for our ND-GSPN. With the proper use of weights and priorities we can get the same result. This is shown the following example.

Example 40. In Figure 7.2 the transitions t_1 or t_2 can be taken non-deterministically. The transitions t_3, t_4, t_5 and t_6 have weights, corresponding with the probabilities in Figure 7.1, and a higher priority than t_1 and t_2. This means that it will not happen that p_2 and p_3 both have a token. All other immediate transitions in the model should also have a different priority than t_3, t_4, t_5 and t_6, as they could otherwise influence the probabilities.

If we compare Figure 7.1 with Figure 7.2 we see that Figure 7.1 is much smaller than Figure 7.2. In Figure 7.2 we need more work to manage the priorities and weights

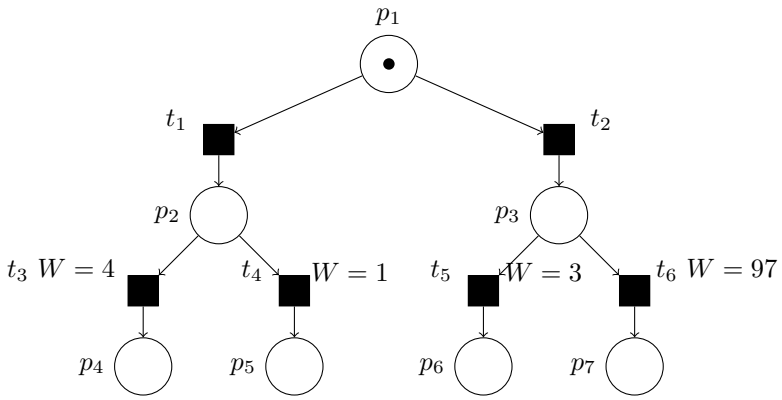


Figure 7.2: ND-GSPN of Figure 7.1 without probabilistic arcs. Transitions t_1, t_2 have priority 1 and t_3, t_4, t_5, t_6 priority 2.

to get the same results as in 7.1. Using a bigger model will make the difference between the approaches only bigger, as managing the priorities with more immediate transitions is more complex.

The probabilistic arc does not make our ND-GSPN more expressive, but does allow to model more efficient and intuitive. Probabilistic arcs are useful in situations were we want to define a local probabilistic choice, which has nothing to do with anything else in the model. For example, a check that can result in OK or NotOK, with certain probabilities, can be modelled effectively with probabilistic arcs.

7.1 ND-GSPNs with probabilistic arcs

Definition 41 (Probabilistic Arc)**.** *An ND-GSPN with probabilistic arcs is a 12-tuple* $G = \langle P, m_0, IMT, PRI, W, RT, R, I, F, RE, PA, MU \rangle$*, where*

- $P, m_0, IMT, PRI, W, RT, R, I, F, RE$ *are the same as in Definition 29;*
- $PA \subseteq T \times \langle 0, 1] \times P$ *is a finite set of probabilistic arcs;*
- $MU : (F \cup I \cup PA) \times \mathbb{N}^{>0}$ *is the multiplicity function;*

Given a transition $t \in T$ *we denote* $probarc(t) = \{a \in PA \mid source(a) = t\}$ *as the set of all probabilistic arcs, where* $source(a)$ *is the transition of the probabilistic arc* a*. For a probabilistic arc* $a \in PA$ *its probability is* $pr(a)$*. The probabilities of the probabilistic arcs always sum up to one:* $\sum_{a \in probarc(t)} pr(a) = 1$*. A transitions has either probabilistic or normal outgoing arcs, but can not have both:* $\exists a \in probarc(t) \oplus \exists a \in out(t)$*.*

Definition 41 adds the probabilistic arcs to the ND-GSPN defined in Section 5. The arc has a transition as source, a place as target and a probability. It also has a multiplicity to indicate how many tokens it will move. If there are probabilistic arcs there is one probabilistic experiment to determine which arc is taken.

Adding probabilistic arcs means the definition of firing a transition has to change. The next state is not fixed any more, but probabilistically distributed. If a transition has no probabilistic arcs, then the probability of one unique state is 1 (Dirac distribution).

Definition 42 (Probabilistic Firings)**.** *Given an ND-GSPN* $G = \langle P, M_0, IMT, PRI, W, RT, R, I, F, PA, RE, MU \rangle$*,* $\nu \in \mathsf{Distr}(S)$ *and markings* $m, m' \in \mathcal{M}$*, a firing from* m *by transition* $t \in T$ *is denoted by* $m \xrightarrow{t} \nu$*, if* $t \in en(m)$ *and*

$$(\nexists a \in probarc(t) \wedge m \xrightarrow{t} m' \wedge \nu = \mathbb{1}_{m'}) \vee$$
$$(\exists a \in probarc(t) \wedge$$
$$\forall m' \in \mathcal{M} \,.\, \nu(m') = \sum \left\{ \left| pr(a) \in \langle 0, 1] \mid a \in probarc(t) \wedge m \xrightarrow{<t,a>} m' \right. \right\})$$

Where $m \xrightarrow{<t,a>} m'$ *is defined precisely as* $m \xrightarrow{t} m'$ *in Definition 32 on, under the assumed restriction of* $out(t)$ *to the singleton set* $\{a\}$*.*

This definition guarantees that for an enabled transition there is at least one next state m' for which $\nu(m') \neq 0$.

7.2 ND-GSPNs with probabilistic arcs expressed as MA

To add this new functionally in the Definition 34 of ND-GSPN semantics as an MA we need to take the probabilistic distribution defined by the probabilistic arc into account.

Definition 43 (ND-GSPN semantics)**.** *Given an ND-GSPN* $G = \langle P, M_0, IMT, PRI, W, RT, R, I, F, PA, RE, MU \rangle$*, its semantics is given by the MA* $M = \langle S, m_0, A, \hookrightarrow, \rightsquigarrow \rangle$*, where*

- $S = \{m' \in \mathcal{M} \mid m_0 \xrightarrow{t}{}^{*} m'\}$
- $A = \{\tau\}$
- $\hookrightarrow$ *is such that, for every* $m, m' \in S$ *and* $\mu, \nu \in \mathsf{Distr}(S)$*,*

$$\begin{aligned} m \overset{\tau}{\hookrightarrow} \mu \quad \iff \quad & \exists t \in IMT \,.\, W(t) = \bot \wedge m \xrightarrow{t} \mu \\ \vee \quad & \exists t \in IMT \,.\, W(t) > 0 \ \wedge \\ & \forall m' \in \mathcal{M} \,.\, \mu(m') = \frac{\sum \left\{ \left| W(t) \cdot \nu(m') \in \mathbb{R} \mid t \in IMT \wedge W(t) > 0 \wedge m \xrightarrow{t} \nu \right. \right\}}{\sum \left\{ \left| W(t) \in \mathbb{N} \mid t \in IMT \wedge W(t) > 0 \wedge \exists m \xrightarrow{t} \nu \right. \right\}} \end{aligned}$$

- $\rightsquigarrow$ *is such that, for every* $m, m' \in S$*,*

$$m \overset{\lambda}{\rightsquigarrow} m' \quad \iff \quad \lambda = \sum \left\{ \left| R(t) \cdot \nu(m') \in \mathbb{R} \mid t \in RT \wedge m \xrightarrow{t} \nu \right. \right\} \wedge \lambda > 0$$

For non-deterministic interactive transitions we use the distribution defined by the probabilistic firing. For weighted interactive transition we multiply the probabilities defined by the probabilistic firing with the probabilities defined by the weights of the transition. For Markovian transition we multiply the rates with the probabilities defined by the probabilistic firing.

The new probabilistic distributions are merged with the transitions. This means we do not create any new states in the MA by introducing the probabilistic arc.

7.3 ND-GSPNs with probabilistic arcs translation to MAPA

In the translation from ND-GSPN to MAPA we need to adjust the statement $\overline{n}_t$. We distinguish the two cases with and without probabilistic arcs.

Definition 44 (Firing Summand)**.** *Statement $\overline{n}_t$ from Definition 37 is replaced by:*

- $\nexists a \in probarc(t)$ *then* $\overline{n}_t$
- $\exists a \in probarc(t)$ *then*

$$\tau \cdot \sum_{a \in probarc(t)} pr(a) : \overline{n}_{<t,a>}$$

Where $\overline{n}_{<t,a>}$ is defined precisely as $\overline{n}_t$ under the assumed restriction of $out(t)$ to the singleton set $\{a\}$.

This means that when there are no probabilistic arcs in the transition we use the old definition of $\overline{n}_t$ as defined in Definition 37. If there are probabilistic arcs, then the result is a summation over all probabilistic arcs.

7.4 Proof of correctness of the translation

The proof should still be valid with the addition of the probabilistic arc. We only need to show that Lemma 2 on page 45 is still correct. This Lemma is still correct if we compare Definition 42 with Definition 44. Definition 42 uses the old firing definition if there are no probabilistic arcs. If there are probabilistic arcs the result is a summation over these arcs. Each arc has a probability of $pr(a)$ to be taken as outgoing arc for transition t. Exactly the same happens in the translation to MAPA in Definition 44.

Transition Merging In the Definition 43 the probabilities defined by the probabilistic arcs are used directly to determine the probabilities of the next sates in the MA. These probabilities are multiplied with the rates for the Markovian transitions and multiplied with the probabilities defined by the weights for interactive the transitions.

In the MAPA specification we have a τ transition followed by a probabilistic summation. However, this summand can be reduced by merging the τ transition and the summation with the transition that fires. This means multiplying the rate or probability of the firing transition with the probabilities of the options in the summation. This reduction creates an equivalent MA. This means the MA under the MAPA specification is still equivalent with the MA in Definition 43.

7.5 Probabilistic Arc Example

To illustrate the probabilistic arc extension we give an example. We make an addition to the library example used in Section 5. We show the ND-GSPN with probabilistic arcs and the corresponding MAPA specification.

Example 45. Figure 7.3 shows an ND-GSPN which models a simple library. We see that the results of toClient are now distributed. When toClient fires there is a probability of $\frac{1}{2}$ that the token goes to p_3 and a probability of $\frac{1}{2}$ that the tokens goes to p_4. The transitions behave the same as described in Section 5.

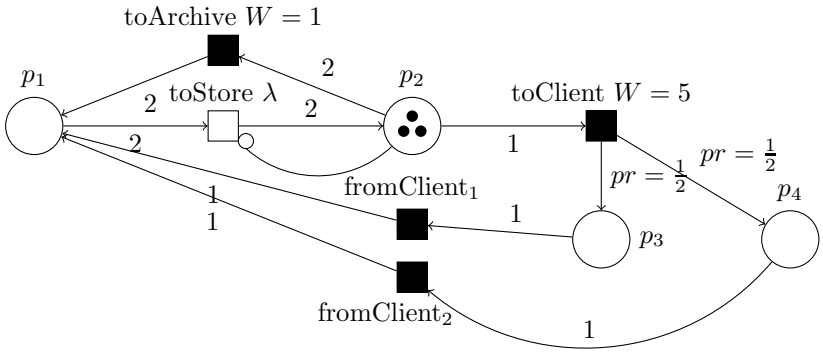


Figure 7.3: ND-GSPN with probabilistic arcs.

Example 46. The MAPA specification of the ND-GSPN in Figure 7.3 is $X(0,3,0,0) = p$ with $\overline{n} = \{p_1, p_2, p_3, p_4\}$ and

$$
\begin{aligned}
p = \\
&(p_1 \geq 2 \wedge p_2 < 1) \Rightarrow \lambda \cdot X(p_1' = p_1 - 2, p_2' = p_2 + 2) + \\
&(p_3 \geq 1) \Rightarrow \tau \cdot X(p_1' = p_1 + 1, p_3' = p_3 - 1) + \\
&(p_4 \geq 1) \Rightarrow \tau \cdot X(p_1' = p_1 + 1, p_4' = p_4 - 1) + \\
&(p_2 \geq 2) \Rightarrow \tau \cdot \ (\\
&\quad (1/6 \rightarrow X(p_1' = p_1 + 2, p_2' = p_2 - 2)) + \\
&\quad (5/6 \rightarrow \tau \cdot (1/2 \rightarrow X(p_2' = p_2 - 1, p_3' = p_3 + 1) \ + \ 1/2 \rightarrow X(p_2' = p_2 - 1, p_4' = p_4 + 1)) \ )
\end{aligned}
$$

We see that when transition toClient fires there are two possibilities with probability $\frac{1}{2}$. It doesn't matter that the firing of toClient is determined probabilistic by the weights as only the firing of a transition has changed with this extension.

Part III

Results

Chapter 8

GEMMA: a tool for translating ND-GSPNs to MAPA

We implemented a tool called GEMMA (non-deterministic GEneralised stochastic petri nets to Markov AutomatA), to automate the translation from ND-GSPN (specified as PNML) to MAPA, conform Chapter 6.

GEMMA is written in the functional program language Haskell. The functional nature of Haskell is well suited for the translation.

GEMMA consists of only 300 lines of code. GEMMA needs a PNML file as import and exports to MAPA. The source PNML file should be as described in Section 5.3.2. Any element of the net which is not conform the DTD in Section 5.3.2 is ignored by the parser and does not appear in the MAPA specification.

The HXT package [1] is used for the parsing of the PMML. HXT is an extension to Haskell which provides a framework for the parsing and creation of XML. The PNML is parsed in a few steps. In the first step the general information about the net is parsed. In this step the method *getPetriNet* calls the methods *getPlaces*, *getTransitions* and *getArcs*. Each method gets the needed information from the PNML source and saves this in an internal data structure.

8.1 Data Structure

The data in GEMMA is stored in an internal data structure. We present a class diagram of the internal data structures in Figure 8.1. This structure contains all the data of the Petri Net.

Net The net class contains an id and optional a name. The net class also contains a list of places, transitions and arcs which describe the behaviour of the net.

Place A place has an id, an optional name and a number of initial tokens. The initial tokens are used to determine the initial state of the MAPA process.

Transition A transition has an id and an optional name. A transition also has a priority. This priority is always 0 for Markovian transitions and always ≥ 1 for immediate transitions. The weight field contains the weight of the transitions. If the

transition is Markovian (priority 0), then this fields is interpreted as the rate of the transition. If the transition is immediate but doesn't have a weight then the weight field is set to -1.

Arc An arc has an id, a source, a target, a type, a multiplicity and a probability. This source and target correspond with the id of a place or a transition. The multiplicity specifies the amount of tokens the arc will move. An arc can have three types: normal, inhibitor and reset. For normal and inhibitor arcs the multiplicity is 1 by default. For reset arcs the multiplicity is not used. The probability of an arc is by default 0. When an arc has a probability this means that it will deliver its tokens giving that probability.

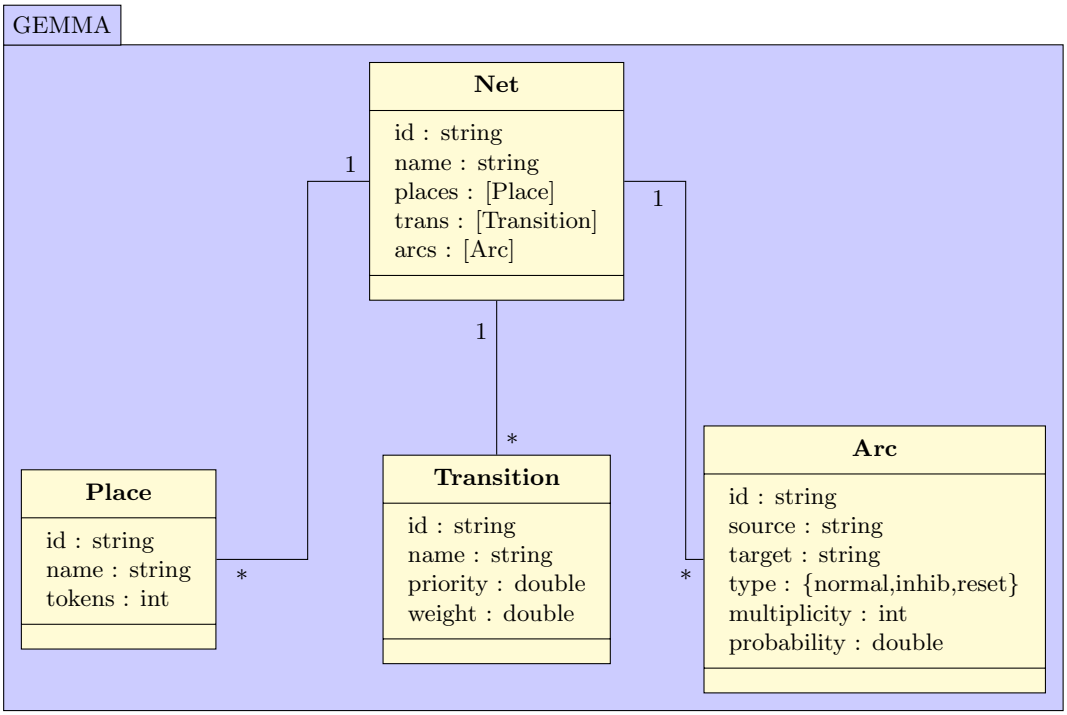


Figure 8.1: Class Diagram of the internal data structures of GEMMA.

Output

After the data is parsed it is translated to the corresponding MAPA specification. The tool will follow the ND-GSPN to MAPA translation described in Section 6.1. GEMMA will automatically name all immediate transitions τ. The MAPA specification will contain one process which is the id of the Petri Net. GEMMA will output the transitions of the process as defined in section 6.1. If the Petri net contains a probabilistic sum, then GEMMA creates a constant in the MAPA specification. This constant defines the total weight, which is used multiple times in the MAPA specification. Using this constant creates a smaller and more readable output.

8.2 Usage

GEMMA can be used via the command line or the web-interface. We discuss both ways and the configuration options GEMMA provides.

8.2.1 Command Line

GEMMA runs via the command line. The standard input (stdin) is used as input for the PNML and the MAPA specification is returned on the standard output (stdout). This means it is possible to read a file in PNML and output to a file which contains the MAPA specification. This also makes it easy to connect GEMMA directly to other tools.

8.2.2 Web-interface

GEMMA can also be used via a web-interface [3]. GEMMA is integrated in the web-interface of SCOOP. This interface combines the functionality of GEMMA, SCOOP and IMCA. An overview of this web-interface is shown in Figure 8.2. The PNML specification can be given in a text field. Another text field can be used to specify the reachability condition (denoted by Specs in Figure 8.2), this could be a desired marking in the net. It is possible to immediately use IMCA to check this reachability condition. Internally GEMMA is used to translate the PNML to MAPA, then MAPA is used as input for SCOOP. SCOOP generates the transition file of the Markov Automata and then IMCA will model-check on this transition file with the specified reachability condition. It is possible to output the intermediate steps as well.

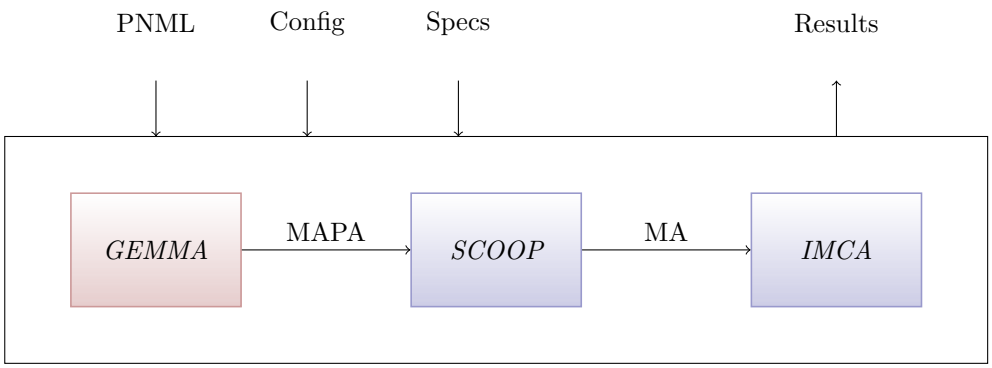


Figure 8.2: Tool Web-interface which combines the functionality of GEMMA, SCOOP and IMCA. Input is the ND-GSPN in PNML, the configurations so that the input is parsed correctly as PNML and the specification of the properties as reach condition (Specs). The output is the results of the model-checking by IMCA.

8.2.3 Options

The options can be given as flags in the command line or as check boxes in the web-interface.

GEMMA has the following options:

- -noWeights: omit all weights and replace them with non-determinism. This translates the ND-GSPN to an MA with no probability for the interactive transition, which thus is an IMC.
- -noPriority: omit all priorities and thus sets all priorities of the immediate transitions to 1.

- -keepNames: Use the names of the transitions instead of naming all immediate transitions τ.

Disabling the weights and priorities gives the translation which Martin Neuhaußer proposed in [32]. With the keepNames flag the confluence reductions made by SCOOP will not have impact on the state space as the transition now all have unique names. This only works for the immediate transitions without weights. Rate transitions do not have a label and weighted transitions are combined in a probabilistic summation with only one label.

Chapter 9

Case Studies

9.1 Case Studies Set Up

In this chapter we present several case studies to verify the results of the implementation and translation from ND-GSPN to MAPA. We show that our translation creates the correct MA from ND-GSPNs specified as PNML. We model-check on the resulting MA and compare the results with previous research. We also show the efficiency of the tool-chain GEMMA, SCOOP and IMCA by showing the computation times of each individual tool.

We consider the following case studies:

- *Workstation Cluster* [20]
- *Tank Fluid System* [12]
- *Google File System* [19]

We chose these case studies because they are all done on the old approach of model-checking for GSPNs. This means that in these case studies the GSPN is translated to a CTMC or an IMC. In our approach we translate the GSPN to an MA. We check if the results in our approach do not differ from the result of the old approach. The workstation cluster case study has also been done by translating it to an MA [18]. However, this translation was done manually and we translate the GSPN with the tool GEMMA. This means we can check if our translation creates the same MA.

We used the same set up for all three case studies. The GSPN models are created with the tool PIPE [8]. With PIPE it is possible to graphically create a GSPN and save this as PNML [45]. The PNML file of the model is then translated to a MAPA specification by the tool GEMMA. The MAPA specification is used by SCOOP, which will export the state space to the IMCA format. With IMCA we can model-check on this state space. We chose IMCA to model-check as this is currently the only model-checker for Markov automata. The set up follows the overview presented in Figure 1.1.

The properties we check on the models are specified in the MAPA specification. We add a reachability condition which defines the marking of the model which we want to reach. We present these properties in logic, before we translate them to the corresponding reachability condition.

All tools were used on a laptop running on a 32-bit Windows 7, which has 4 GB RAM and an Intel Core i5 CPU 2.54 Ghz. The computation times results are an average of three runs for computation times less than 30 seconds. If the computation

time is longer we use only one run, because a longer computation time means a smaller error margin.

Reductions in SCOOP As described earlier in Section 4.1.3 on page 28 SCOOP uses several reduction techniques to optimize the state space or speed-up the state space generation. However, these reduction are not relevant in our case. The state space reduction techniques could create a smaller state space, but the dead variable reduction [44] is not relevant and confluence reduction [43] is not yet available for MAPA. Dead variable reduction can delete variables that are never used or changed. The variables in our MAPA specifications represent places, which are always relevant in the models.

This means the reductions provided by SCOOP do not have effect in our approach. Confluence reduction could have an effect, if it becomes available for MAPA specifications. We only use SCOOP to generate the model from the MAPA specification and not for its reduction techniques.

9.2 Case Study I: Workstation Cluster

The first case study that we consider is the workstation cluster example [20]. It combines multiple workstations, which can be modelled effectively with Petri Nets. This case study has been done before ([20] [18]), so we can check if we get the same results in our new approach of model-checking. The model is used to check the performance of such a workstation system, where various components depend on each other.

The workstation cluster consists of two sides of N workstations which are connected via a backbone. Each side has a switch that connects the workstations with the backbone. The workstations can communicate with the workstations on their side. However, a workstation can only communicate to a workstation on the other side if the backbone and both switches are up and running. An overview of the system is shown in Figure 9.1, which is taken from [20].

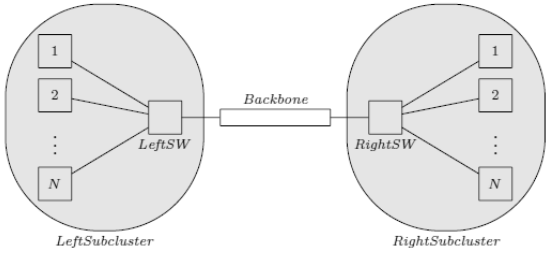


Figure 9.1: Workstation Cluster

The workstations, switches and the backbone can fail. When they fail they need to be repaired. There is a repair unit which can repair only one object at the time.

We can check the performance of this workstation cluster. Interesting are the expected time before the system is not operational any more and how long the system stays in this situation on average. An example is the situation where the backbone or a switch is down, which means the workstations on the left side can't communicate with the ones on the right side.

9.2.1 Model

The workstation cluster is modelled as ND-GSPN $\mathcal{G} = \langle P, m_0, IMT, PRI, W, RT, R, I, F, RE, MU \rangle$ in Figure 9.2. Note I and RE are not present in this model. P are the places that represent the objects in the model. The immediate transitions IMT are used to select a failed object for repair. The rate transitions RT represent the fail and repair durations. The model has a backbone which is connected to a left and a right switch. Each switch has N workstations connected to them. There is one repair unit which can repair a backbone, switch or workstation. In the initial state (m_0) the backbone, both switches and all workstations are up.

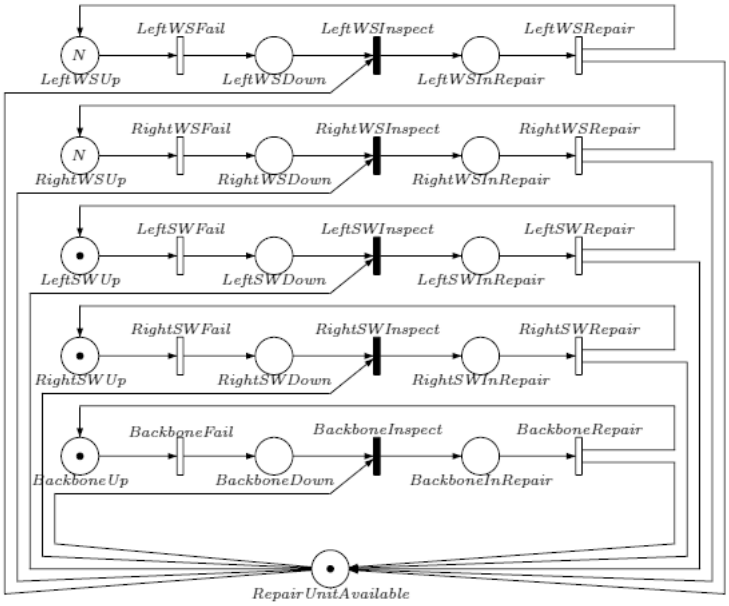


Figure 9.2: GSPN from the Workstation Cluster

The failure and repair durations are shown in Table 9.1. These are used for the rates of the rate transitions RT in the model. For example, the duration that a left or right workstation will fail is translated to a rate of $n/500$, where n is the number of operational workstations on that side. The duration that the backbone fails is translated to a rate of $1/5000$.

event	duration	event	duration
LeftWSFail	500h	LeftWSRepair	0.5h
RightWSFail	500h	RightWSRepair	0.5h
LeftSWFail	4000h	LeftSWRepair	4h
RightSWFail	4000h	RightSWRepair	4h
BackboneFail	5000h	BackboneRepair	8h

Table 9.1: Average duration of component failures and repairs for Figure 9.2. Translated to the rates of the transition: 1 / duration.

The repair unit can be claimed by the immediate transitions of each component when they have failed. It is possible that multiple components are down at the same

time, which causes a conflict between the immediate transitions. Different policies can be used to resolve these conflicts. Because there is only one repair unit, we should define a repair strategy. This can be done by defining weights or priorities for the immediate transitions. With the use of priorities we can specify that an object will always be repaired before another object. With weights we can specify the probabilities that each object is repaired. Assigning a higher weight to an object means that the probability that it well be repaired before other objects is higher. Not specifying a weight for a transition means that it can be taken non-deterministically. In practice this means we let the weight of the transition be variable and thus not specify how we resolve the conflict. We consider three policies for resolving conflicts:

(1) Non-determinism The first policy is the use of non-determinism to solve all conflicts. In this policy the reachability diagram of the ND-GSPN is an IMC. This is done in [20] and [18]. Formally this means:

$$\forall t \in IMT \ . \ W(t) = \bot$$

(2) Weights The conflicts can also be solved by assigning weights for all the immediate transitions. This means that the reachability diagram of the ND-GSPN can be transformed to a CTMC. This is the old method of analysing GSPNs, as described in [28]. We assign weight 1 to all transitions:

$$\forall t \in IMT \ . \ W(t) = 1$$

(3) Combination The last policy we consider is a combination of weights and non-determinism. This policy illustrates our work since it was not possible before. This policy allows that some transitions can have weights while other can be taken non-deterministically, making the semantics a true MA. We use the following weight assignments in this policy:

$$\begin{aligned} W(\mathit{BackboneInspect}) &= 2 \\ W(\mathit{Left/RightSWInspect}) &= 1 \\ W(\mathit{Left/RightWSInspect}) &= \bot \end{aligned}$$

9.2.2 Analysis

The GSPN described in Figure 9.2 was modelled with the tool PIPE [8]. PIPE saves the ND-GSPN as PNML [45]. Furthermore, we use the overview as in Figure 1.1 to get from the PNML specification to the results.

The failure rates of the workstations depend on the current number of workstations that are up. This is variable and can not be defined in the PNML. We edit these two failure rates manually in the MAPA specification.

We use two properties to check the performance of the system.

Property I

The first property we check is the situation in which both sides of the workstation cluster are down. This means that in both sides all workstations are down or the switch is down. We check:

1. The expected time to get in this situation.
2. How long we stay in this situation.

3. The unbounded reachability to get in this situation.

The initial state of the model is m_0. We use goal states G to specify the property:

$$G = (\mathit{LeftWorkStationUp} = 0 \vee \mathit{LeftSwitchUp} = 0) \wedge (\mathit{RightWorkStationUp} = 0 \vee \mathit{RightSwitchUp} = 0)$$

This gives us the following properties:

$$\begin{aligned} \text{Expected Time: } & eT^{min,max}(m_0, \diamond G) \\ \text{Unbounded Reachability: } & Pr^{min,max}(m_0, \diamond G) \\ \text{Long-Run Average: } & LRA^{min,max}(m_0, G) \end{aligned}$$

Analysis for these properties results in a minimum or a maximum result, which can be interpreted as the bounds of an interval. We get an interval, because we may have non-deterministic choices between transition. This means that any policy to resolve the non-determinism is within the interval, indicated by the minimum and maximum result. The properties are used as a reachability condition and added to the MAPA-specification. This reachability condition specifies the states we want to reach. Which type of model-checking we use is an option in the IMCA model-checker. This means that the three properties can be checked with the same reachability condition which corresponds to G:

$$\begin{aligned} &\mathit{reachCondition}(\mathit{LeftWorkStationUp} = 0 \mid \mathit{LeftSwitchUp} = 0) \\ &\quad \&(\mathit{RightWorkStationUp} = 0 \mid \mathit{RightSwitchUp} = 0) \end{aligned}$$

We use $|S|$ as the number of states, $|T|$ as the number of transitions and $|G|$ as the number of goal states in the model.

Computation times First we show the computation times for the tools GEMMA, SCOOP and IMCA for models with $N \in \{1, 4, 8, 16, 32, 64\}$. We check the computation times for the maximum result of the properties stated above. The computation time of GEMMA is for each of the models the same. GEMMA took an average 0.061 seconds to translate the PNML to MAPA. The results are shown in Table 9.2, where X means we did not get a result. For the computation times less than 30 seconds we took an average over three runs. The larger computation times are determined by one run, because the error margin for longer computation times is smaller.

N	$\lvert S\rvert$	$\lvert T\rvert$	$\lvert G\rvert$	SCOOP	$eT^{max}(m_0, \diamond G)$	$Pr^{max}(m_0, \diamond G)$	$LRA^{max}(m_0, G)$
1	111	320	74	0.077	0.0008	0.0012	0.0039
4	819	2996	347	0.422	0.0812	0.0510	0.1966
8	2771	10708	1019	1.376	1.0986	0.6818	2.3533
16	10131	40340	3419	5.682	18.0379	11.4612	45.9752
32	38675	156436	12443	27.145	485.5478	165.2745	X
64	151059	615956	47387	225.213	6257.081	3910.258	X

Table 9.2: Computations times of IMCA for expected time, unbounded reachability and long-run average in seconds. Also includes computation times for SCOOP for generating the model.

Analysis Results The results of the analysis of the properties are shown in Table 9.3. The result are an interval determined by the minimum and maximum results of the model-checking. We used the three different policies for resolving conflicts as

described earlier. We only consider the models with $N \in \{1, 4, 8\}$ in this table, because this is enough to make a comparison with [20] and to illustrate the results for the different policies.

	N	$eT^{[min,max]}(m_0, \diamond G)$	$LRA^{[min,max]}(m_0, G)$
Non-Det	1	[110494.13,110495.42]	[2.1147e-05,2.1273e-05]
	4	[1997317.36,1997454.42]	[2.0144e-06,2.0181e-06]
	8	[1995339.76,1995676.51]	[2.0159e-06,2.0207e-06]
Weights	1	110494.77	[2.1207e-05,2.12126e-05]
	4	1997387.86	[2.0173e-06,2.0177e-06]
	8	1995517.45	[2.0152e-06,2.01995e-06]
Combi	1	[110494.13-110495.41]	[2.1147e-05,2.1272e-05]
	4	[1997320.65-1997451.20]	[2.0155e-06,2.0180e-06]
	8	[1995346.44-1995670.20]	[2.0152e-06,2.0206e-06]

Table 9.3: Results from IMCA for expected time and long-run average.

Conclusion

- The number of states, transitions and goals states in Table 9.2 are exactly the same as in [20]. In [20] the GSPN was translated to an IMC for analysis. The result should be same, as for our non-determinism policy our MA is also an IMC. Further more, the computation times do not differ much from the ones in [20].
- We see that the computation times of SCOOP depend on the number of states. The computation times increase linear with the increase in states. The computation times of IMCA increase more than linear. We see that with 10.000 states, the computation times are still within one minute. However, for 150.000 states, the computation times are over one hour. The computation time of GEMMA does not change when using a bigger model as GEMMA only translates the model syntactically. The total efficiency of the tool chain (GEMMA, SCOOP, IMCA) is thus determined by IMCA as it has, for bigger models, by far the longest computation time.
- The results in Table 9.3 are an interval specified by the minimum and maximum results of the property. For the weights policy, the results are always within the interval of the results of the non-determinism policy. This is expected, because the non-determinism policy does not specify how to resolve conflicts and this means all possible weights assignments for resolving the conflicts should be within that interval.
- We see that the expected time increases and long-run average decreases, when we have more workstations. This is expected as the probability that all workstations are down on one side decreases, when we have more workstations available.

Property II

We now specify a property to check the minimum Quality of Service (QoS). This QoS criterion requires k numbers of workstation to be operational and mutually exclusive. This can be defined by the following property:

$$QoS_k = LeftOperational_k \vee RightOperational_k \vee Operational_k$$

where

$$\begin{aligned} LeftOperational_k &= (LeftWorkstationUp \geq k \wedge LeftSwitchUp > 0) \\ RightOperational_k &= (RightWorkstationUp \geq k \wedge RightSwitch > 0) \\ Conn &= (LeftSwitchUp > 0 \wedge BackboneUp > 0 \wedge RightSwitchUp > 0) \\ Operational_k &= (LeftWorkstationUp + RightWorkstationUp \geq k) \wedge Conn \end{aligned}$$

The QoS property states that minimal k of the $2N$ workstations are required to be operational and mutually exclusive, which in this case means that the k number of workstations need to be able to communicate with each other. Note that if $k < N$ it can be sufficient that k workstation are available on the left or right side, which means communication over the backbone is not required. If this is not the case then the backbone and both switches must connect the workstations on the left and the right side to still have QoS. We define G as all the states where we have QoS.

This gives us the following properties:

$$\begin{aligned} \text{Expected Time: } & eT^{min,max}(m_0, \diamond G) \\ \text{Long-Run Average: } & LRA^{min,max}(m_0, G) \end{aligned}$$

We check the QoS properties for the following configurations of N and k:

$$\begin{aligned} N &\in \{4, 8\} \\ k &\in \{\tfrac{3}{4} \cdot N, N, \tfrac{3}{4} \cdot 2N, 2N\} \end{aligned}$$

QoS is translated to the following reachability condition:

$$\begin{aligned} reachCondition\ !((&LeftWorkStationUp \geq k\ \&\ LeftSwitchUp > 0)\ | \\ &(RightWorkStationUp \geq k \& RightSwitchUp > 0)\ | \\ &((LeftWorkStationUp + RightWorkStationUp) \geq k\ \& \\ &\ LeftSwitchUp > 0\ \&\ RightSwitchUp > 0\ \&\ BackboneUp > 0)) \end{aligned}$$

We again use $|S|$ as the number of states and $|G|$ as the number of goal states. Model-checking for the QoS property gives us again a maximum or minimum result, which we use to as bounds for an interval.

Analysis Results The computation times, for the different models and policies, of SCOOP can be found in Table 9.4. These are the times that SCOOP needs to create the state space for IMCA from the MAPA model. The computation times are an average of three runs.

We model-check the following:

1. How long it takes before the QoS is violated: expected time to violate QoS.
2. How long we stay in the states that violate the QoS: long-run average in not QoS.
3. How long the model stays in the states that do not violate the QoS: long-run average in QoS.

The results of (1) are shown in Table 9.5, the results of two are shown in Table 9.6 and the results of (3) are shown in Table 9.7. The tables show an interval, with the minimum and maximum results as boundaries. The min and max columns show the computation times of IMCA for the minimum and maximum results. The computation times are an average of three runs.

	N	k	$\|S\|$	$\|G\|$	Computation time(s)
Non-Det	4	3	819	566	0.423
	4	4	819	692	0.442
	4	6	819	807	0.451
	4	8	819	818	0.453
	8	6	2771	2009	1.525
	8	8	2771	2482	1.639
	8	12	2771	2736	1.757
	8	16	2771	2770	1.763
Weights	4	3	819	566	0.602
	4	4	819	692	0.591
	4	6	819	807	0.603
	4	8	819	818	0.610
	8	6	2771	2009	2.028
	8	8	2771	2482	2.150
	8	12	2771	2736	2.265
	8	16	2771	2770	2.272
Combi	4	3	819	566	0.468
	4	4	819	692	0.487
	4	6	819	807	0.489
	4	8	819	818	0.493
	8	6	2771	2009	1.654
	8	8	2771	2482	1.755
	8	12	2771	2736	1.895
	8	16	2771	2770	1.905

Table 9.4: Computation times (in seconds) of SCOOP for generating the state space from MAPA to IMCA format.

Conclusion

- The resulting intervals of the weights and combined policies are within the non-determinism intervals. This is expected, because the non-deterministic policy does not specify how to resolve conflicts, which means all possible other policies should be within this interval. The results of the non-determinism policy are an exact match with the results in [18]. This indicates that our translation of the ND-GSPN creates the correct corresponding MA.

- The QoS specifies how many workstations have to be operational. More workstations means that this is more difficult, thus with a bigger N the expected time to violate QoS decreases. A bigger k also decreases the expected time as this requires more workstations to be operational at the same time. The same holds for the time we stay in a state which violates the QoS.

- The computation times of SCOOP for each policy are different. We see that the use of non-determinism over weights causes less computation time. A higher k causes the computation time to increase a little, this because SCOOP has to generate more goal states when we use a higher k.

	N	k	$\|S\|$	$\|G\|$	$eT^{[min,max]}(m_0, \diamond G)$)	min	max
Non-Det	4	3	819	566	[1122465.40,1125179.46]	0.040	0.025
	4	4	819	692	[51699.58,51704.89]	0.010	0.007
	4	6	819	807	1427.22	0.001	0.001
	4	8	819	818	59.88	0.001	0.001
	8	6	2771	2009	[1719447.05,1724343.30]	0.528	0.210
	8	8	2771	2482	[25604.94,25610.45]	0.046	0.031
	8	12	2771	2736	1428.57	0.004	0.004
	8	16	2771	2770	30.58	0.002	0.003
Weights	4	3	819	566	1123836.31	0.027	0.023
	4	4	819	692	51702.22	0.007	0.006
	4	6	819	807	1427.22	0.001	0.001
	4	8	819	818	59.88	0.001	0.001
	8	6	2771	2009	1721920.44	0.268	0.245
	8	8	2771	2482	25607.67	0.032	0.026
	8	12	2771	2736	1428.57	0.003	0.004
	8	16	2771	2770	30.58	0.002	0.003
Combi	4	3	819	566	[1122468.57,1125176.95]	0.042	0.025
	4	4	819	692	[51699.61,51704.86]	0.011	0.007
	4	6	819	807	1427.22	0.001	0.001
	4	8	819	818	59.88	0.001	0.001
	8	6	2771	2009	[1719456.71,1724335.75]	0.435	0.199
	8	8	2771	2482	[25604.97,25610.43]	0.050	0.030
	8	12	2771	2736	1428.57	0.003	0.004
	8	16	2771	2770	30.58	0.002	0.003

Table 9.5: IMCA results for expected time to ¬(QoS) for workstation cluster and minimal and maximal computation time in seconds.

9.3 Case Study II: Tank Fluid System

The second case study that we consider is a tank fluid system example [12]. This system is composed of a tank containing some level of liquid (H). There are two pumps ($P1$ and $P2$), which can fill the tank. There is a valve (V) to empty the tank. There is also a controller which monitors the liquid level and is in control of the valve and the pumps. Interesting is to check the probability that a tank dry out or overflow occurs. The system scheme is shown in Figure 9.3, which is taken from [12].

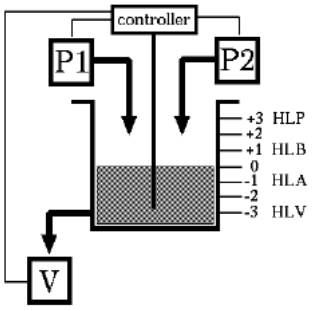


Figure 9.3: Tank Fluid System

	N	k	$\|S\|$	$\|G\|$	$LRA^{[min,max]}(m_0, G)$)	min	max
Non-Det	4	3	819	566	[3.5817e-06,3.6294e-06]	0.094	0.196
	4	4	819	692	[7.6463e-05,7.7037e-05]	0.154	0.208
	4	6	819	807	[3.5987e-03,3.5989e-03]	0.073	0.196
	4	8	819	818	1.1587e-02	0.077	0.184
	8	6	2771	2009	[2.3476e-06,2.3781e-06]	0.998	2.436
	8	8	2771	2482	[1.6212e-04,1.6352e-04]	0.528	2.463
	8	12	2771	2736	[3.6010e-03,3.6015e-03]	0.506	2.412
	8	16	2771	2770	1.9579e-02	0.529	2.439
Weights	4	3	819	566	[3.6009e-06,3.6087e-06]	0.078	0.370
	4	4	819	692	[7.6738e-05,7.6750e-05]	0.073	0.366
	4	6	819	807	[3.5987e-03,3.5988e-03]	0.068	0.389
	4	8	819	818	1.1587e-02	0.064	0.399
	8	6	2771	2009	[2.3549e-06,2.3671e-06]	0.507	14.115
	8	8	2771	2482	[1.6280e-04,1.6282e-04]	0.405	13.896
	8	12	2771	2736	3.6012e-03	0.358	12.945
	8	16	2771	2770	1.9579e-02	0.347	13.944
Combi	4	3	819	566	[3.5818e-06,3.6292e-06]	0.095	0.198
	4	4	819	692	[7.6456e-05,7.7035e-05]	0.0075	0.200
	4	6	819	807	[3.5987e-03,3.5989e-03]	0.071	0.204
	4	8	819	818	1.1587e-02	0.073	0.187
	8	6	2771	2009	[2.3461e-06,2.3779e-06]	0.998	2.436
	8	8	2771	2482	[1.6213e-04,1.6351e-04]	0.654	2.790
	8	12	2771	2736	[3.6010e-03,3.6015e-03]	0.491	2.539
	8	16	2771	2770	1.9579e-02	0.461	2.484

Table 9.6: IMCA results for long-run average in $\neg$(QoS) for workstation cluster and minimal and maximal computation time in seconds.

	N	k	$\|S\|$	$\|G\|$	$LRA^{[min,max]}(m_0, G)$)	min	max
Non-Det	4	3	819	253	0.999996	0.447	0.470
	4	4	819	127	[0.999923,0.999924]	0.448	0.280
	4	6	819	12	0.996401	0.454	0.457
	4	8	819	1	0.988413	0.465	0.487
	8	6	2771	762	0.999998	5.115	5.596
	8	8	2771	289	[0.999836,0.999838]	4.881	3.714
	8	12	2771	35	[0.996398,0.996399]	5.152	5.291
	8	16	2771	1	0.980421	5.238	5.527
Weights	4	3	819	253	0.999996	0.424	0.697
	4	4	819	127	0.999923	0.525	0.729
	4	6	819	12	0.996401	0.501	0.860
	4	8	819	1	0.988413	0.557	0.909
Combi	4	3	819	253	0.999996	0.482	0.420
	4	4	819	127	[0.999923,0.999924]	0.573	0.293
	4	6	819	12	0.996401	1.622	0.428
	4	8	819	1	0.988413	0.550	0.458

Table 9.7: IMCA results for long-run average in (QoS) for workstation cluster and minimal and maximal computation time in seconds.

Initially the liquid level is 0. The valve V and pump $P1$ are turned on and pump $P2$ is turned off. The pumps add and the valve removes liquid at the same rate, so the liquid level does not change in the initial state. When the liquid level gets outside the region between HLA (-1) and HLB (+1) there is a risk of an overflow or a dry out of the tank. The controller will then control the pumps and the valves to get the liquid level back to 0. The pumps and the valve can be ON or OFF. They can also fail, this means they can get stuck in either the ON or OFF state. The failure rates are: $P1 = 0.004566\ 1/h$, $P2 = 0.005714\ 1/h$ and $V = 0.003125\ 1/h$. The effects on the liquid level of the different configurations of the system can be seen in Table 9.8. This table shows all possible configurations of the pumps and the valve. It also shows the effect this configuration has on the liquid level H. The last column shows the variation rate for the liquid level. This is the rate at which the liquid level rises or decreases for that configuration.

State of $P1$	State of $P2$	State of V	Effect on H	Variation rate
ON	OFF	OFF	+	0.6 m/h
ON	ON	OFF	++	1.2 m/h
ON	OFF	ON	=	
ON	ON	ON	+	0.6 m/h
OFF	OFF	OFF	=	
OFF	ON	ON	+	0.6 m/h
OFF	OFF	ON	-	0.6 m/h
OFF	ON	ON	=	

Table 9.8: Variations in liquid level H for each configuration.

The two undesired situations are when the tank overflows, HLP (+3), or when the tank dries out, HLV (-3). These situations should be avoided. This can be done by turning all pumps on and the valve off for a dry out and the opposite for an overflow. Interesting properties in the system are to check the probability that the tank dries out or overflows after an amount of time. Note that when a pump or valve fails it won't be repaired. In Section 9.3.2 we consider an extension to the model where the pumps or the valve can be repaired.

9.3.1 Model

To check the performance of this system we use a GSPN model. The system is modelled as ND-GSPN $\mathcal{G} = \langle P, m_0, IMT, PRI, W, RT, R, I, F, RE, MU \rangle$ in Figure 9.4. Note that RE, PRI and W are not present in this model. We created the model with the tool PIPE [8].

The pumps and valves are shown on the left side of the level place. They can all be ON or OFF and stuck, which is modelled by three places. The four different ways of getting stuck are modelled by four rate transitions. There are four fail transitions as these components can fail in either the ON or the OFF state and when they fail they can stay in the same state or switch states. The rates of these fail transitions is half of the failure rate of the valve or pump. The valve and pumps are switched on or off depending on the liquid level. This is modelled by two immediate transition for each pump or valve. These transition will not get in conflict, as firing one of them does not influence the others. This means that the policy we use to solve the conflicts between immediate transition is irrelevant.

The liquid level is modelled by the level place, which can have 0 to 8 tokens to represent the liquid levels. The rate transitions on the right side model the variations in the liquid level H. Note that these transitions do not have a condition in this model.

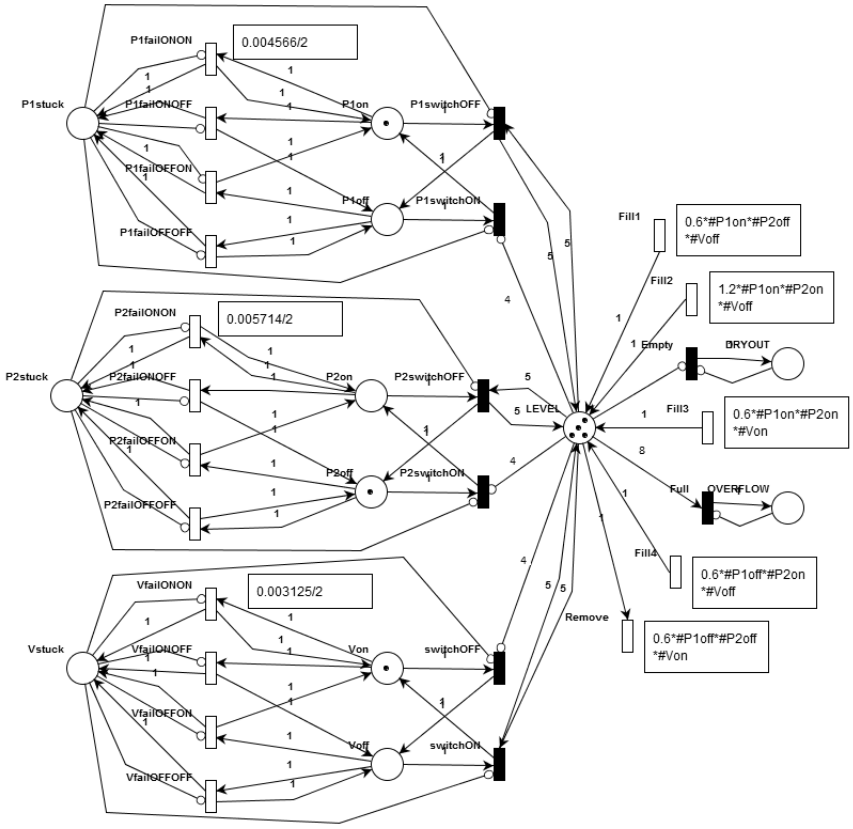


Figure 9.4: GSPN of the Tank Fluid System

However, the definition of the rate contains the condition for these transition. For example, transition *Fill1* has rate $0.6 \times \#P1on \times \#P1off \times \#Voff$. This means the transition is enabled when $P1$ is on and $P2$ and V are off.

The places DRYOUT and OVERFLOW represent the undesired situations. In the initial situation (m_0) the liquid level is 4, $P1$ and V are on and $P2$ is turned off.

Analysis

The GSPN in Figure 9.4 is unbounded as the fill transitions do not have a condition and thus can fill the level place even if there is an overflow. We gave the fill transitions the condition that the tank should not have overflown. The remove transition gets the condition that there is no dry out. This adds extra arcs, which are not shown in Figure 9.4, from the level place to the fill and remove transitions. This still creates the correct model as we are only interested in the situations of the model until an overflow or dry out occurs.

The analysis is done by checking the probability that a dry out or overflow occurs in d units of time. This time-bounded model checking is done on two properties:

$$\text{Dry out: } \mathbb{P}(\diamond^{\leq d} DRYOUT)$$
$$\text{Overflow: } \mathbb{P}(\diamond^{\leq d} OVERFLOW)$$

These properties are used as the following reachability conditions:

$$reachCondition\,DRYOUT = 1$$
$$reachCondition\,OVERFLOW = 1$$

We check these properties on time-bounded model-checking.

Analysis Results The translation from PNML to MAPA took GEMMA 0.078 seconds (average of three runs). The state space generation from MAPA to IMCA format took SCOOP 0.074 seconds (average of three runs). The model consists of 115 states and 227 transition.

The results are shown in Table 9.9. The computation times (in seconds) of IMCA are also shown in Table 9.9 and are determined by one run. We use an error bound e of 10%, as described in Section 4.2.

time d	e	$\mathbb{P}(\diamond^{\leq d} DRYOUT)$	time	e	$\mathbb{P}(\diamond^{\leq d} OVERFLOW)$	time
100	0.0001	0.0047	472.572	0.001	0.0762	47.705
200	0.001	0.0224	189.001	0.01	0.1970	19.110
300	0.001	0.0452	426.195	0.01	0.2935	43.056
400	0.001	0.0661	795.737	0.01	0.3608	76.502
500	0.001	0.0828	1192.372	0.01	0.4060	119.507
600	0.001	0.0952	1714.314	0.01	0.4361	172.287
700	0.01	0.1041	233.080	0.01	0.4562	235.576
800	0.01	0.1103	303.514	0.01	0.4698	307.617
900	0.01	0.1147	384.463	0.01	0.4790	389.689
1000	0.01	0.1177	474.693	0.01	0.4853	481.058

Table 9.9: IMCA results for time-bounded model-checking to overflow or dry out for tank fluid system. Computation times are in seconds.

Conclusion

- We compared the results in Table 9.9 with [12]. The results are very close to the ones in [12]. In [12] the GSPN was translated to a CTMC for analysis. The difference in the results is so little that it can be explained as round-off errors. The results indicate that our translation creates the correct model of the tank fluid system from the ND-GSPN.
- The computation times for dry out or overflow are almost the same. The computation times heavenly depend on the used error bound. Using an error bound 10 times smaller makes the computation time also 10 times longer. We used an error bound of 10%, but we also tried higher error bounds. When we used an error bound of 1 the results are the same, but we cannot conclude that they are correct as the error margin is too big.
- The probability of an overflow is higher than the probability of a dry out. This is expected, because according to Table 9.8, there is only one configuration in which the liquid level decreases, while there are multiple in which the liquid level increases. The probability that the model gets stuck in a configuration in which the level increases is thus higher.
- We also tried different policies for solving the conflicts of the immediate transitions. This has no effect as in this model the immediate transitions will not get

in conflict with each other. The immediate transition turn the pumps or valve on or off, but are independent of each other. Turning one pump off before the other will not have any effect on the results. We did the analysis for the policies with weights and with non-determinism and found no difference in the results.

9.3.2 Repair Extension

The GSPN in Figure 9.4 has no option to repair a pump or valve when it fails. We now add a repair policy to the model as is also done in [12]. Repairing components should have a significant effect on the probabilities of a dry out or overflow occurring. The GSPN with repair policy is shown in Figure 9.5.

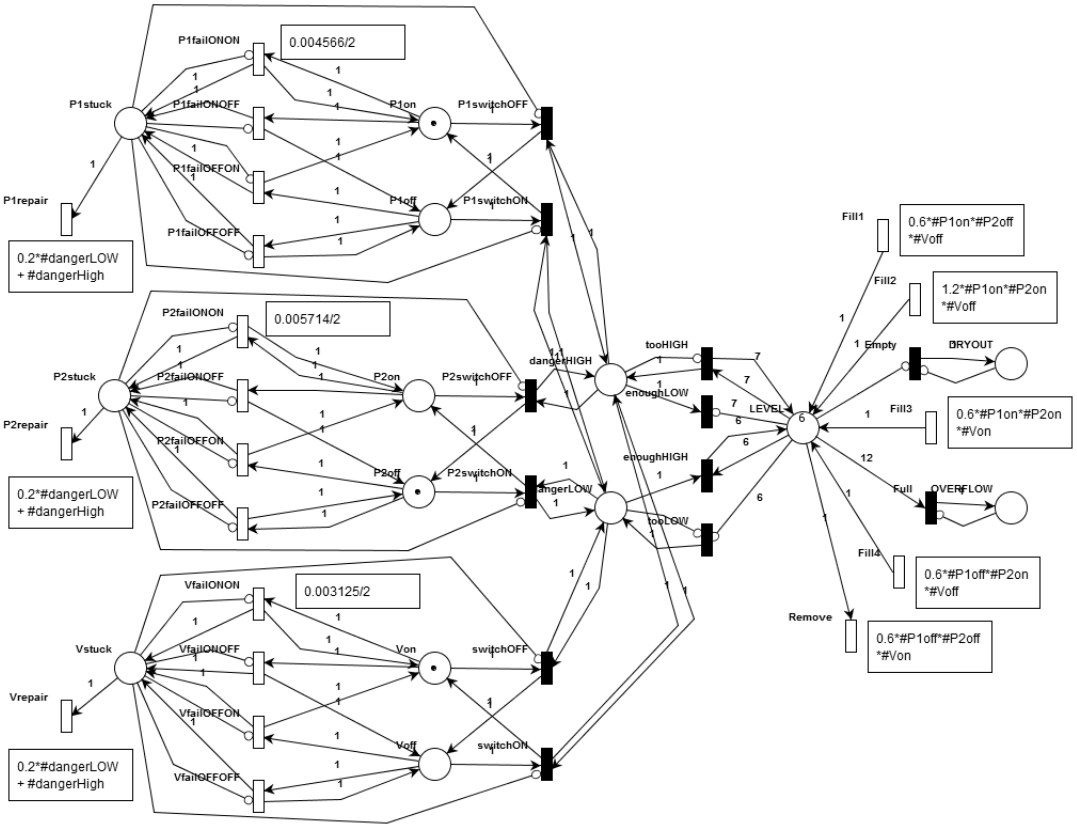


Figure 9.5: GSPN of the Tank Fluid System with repair policy

In Figure 9.5 the controller can detect when the liquid level is too high or too low. When this happens there must be a failure in one of the components and the repair transitions on the left side will get enabled. The components will get repaired at a rate of 0.2 $1/h$. The modelling of the liquid level is changed to the scale of zero to twelve. This is done to give the repair extension more time to repair components before a dry out or overflow occurs. The initial level is now 6 and overflow is level 12.

Analysis

The model is created the same way as in the previous section. The properties to check are still the same as the goal places DRYOUT and OVERFLOW are still present in

the extension of the model. We check again on the following properties:

$$\text{Dry out: } \mathbb{P}(\diamond^{\leq d} DRYOUT)$$
$$\text{Overflow: } \mathbb{P}(\diamond^{\leq d} OVERFLOW)$$

These properties are used as the following reachability condition:

$$reachCondition\, DRYOUT = 1$$
$$reachCondition\, OVERFLOW = 1$$

Analysis Results The translation from PNML to MAPA took GEMMA 0.087 seconds (average of three runs). The state space generation which was used as input for IMCA took SCOOP 1.625 seconds (average of three runs).

The results of the time-bounded model checking for different time d are shown in Table 9.10. The model consists of 3103 states and 8318 transition. We use an error bound of 0.01, as described in Section 4.2. We use an error bound of 0.01, because an error bound of 10% would take to long in this case (multiple hours for each result). This also means we omitted the computation times of IMCA, as we got the results faster with this smaller error bound.

Conclusion

- The computation time of GEMMA to create the MAPA specification is a little longer than in the original model. The PNML is also a little bigger, than in the original model and GEMMA only translated the model syntactically. SCOOP took almost 20 times longer to generate the state space, but the state space is also about 25 times as big as in the original model.
- We again compare the results in Table 9.10 with [12]. The results are the same compared with [12]. This again indicates that the translation of our model creates the correct MA from the GSPN.
- The computation times of IMCA depend on the used error bound. If we would use an error bound of 10% getting a results would take multiple hours. We therefore used a smaller error bound. This would mean the results are less accurate, but we see that compared with [12] the results are the same.
- We also see that the probabilities of a dry out or an overflow decreased a lot compared to the results in 9.9. This is expected as the repair policy can fix the pumps or valve before an overflow or dry out occurs. This model also uses a bigger tank which also decrease the probability of a dry out or an overflow.

9.4 Case Study III: Google File System

The third case study that we consider is the Google File System [19]. This case study is an application of testing the performance of a file system.

In this system, the files are divided into *chunks* of equal size. There are several *chunk servers* which contain copies of each chunk. The *master server* knows the location of each chunk copy and can give these locations to the user. The data is then send directly from the chunk server to the user.

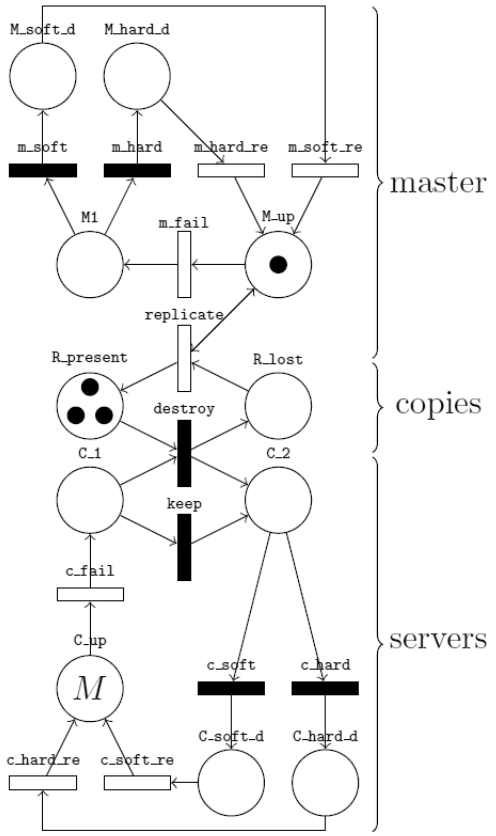
As the servers can fail and be repaired we can check the reliability of this Google file system.

time d	$\mathbb{P}(\diamond^{\leq d} DRYOUT)$	$\mathbb{P}(\diamond^{\leq d} OVERFLOW)$
50	1.0e-5	0.0010
100	6.2e-5	0.0032
150	0.0002	0.0060
200	0.0003	0.0093
250	0.0004	0.0127
300	0.0006	0.0163
350	0.0007	0.0199
400	0.0008	0.0236
450	0.0009	0.0273
500	0.0011	0.0309

Table 9.10: IMCA results for time-bounded model-checking to overflow or dry out for tank fluid system with repairs.

9.4.1 Model

The Google file system is modelled as ND-GSPN $\mathcal{G} = \langle P, m_0, IMT, PRI, W, RT, R, I, F, RE, MU \rangle$. Note PRI, RE and I are not present in this model. We created the model with the tool PIPE [8]. The model and the table with the rate/probability of the transitions is shown in Figure 9.6.



transition		rate / prob.
m_fail		0.0005
m_soft_re		#M_soft_d·12
m_hard_re		6.0
c_fail		0.05
c_soft_re		#C_soft_d·12
c_hard_re		1.0
replicate	20.0	if #R_present>0
	2.0	if #R_present=0
m_soft		0.95
m_hard		0.05
c_soft		0.95
c_hard		0.05
destroy		$\frac{\#\text{R\_present}}{\#\text{C\_up}}$
keep		$1 - \frac{\#\text{R\_present}}{\#\text{C\_up}}$

Condition for replicate: $(\#\text{C\_up} > \#\text{R\_present}) \wedge (\#\text{C\_up} \cdot S \geqslant (\#\text{R\_present}+1) \cdot N)$

Figure 9.6: Google File System

The GSPN model of this system describes the life cycle of a single chunk, which can be influenced by the load of other chunks. The model has three parts, the master, the copies and the servers. The master can be up (M_up) or down. The master can be down without knowledge of the failure ($M1$) or with knowledge of a hardware (M_hard_d) or software (M_soft_d) failure. The same goes for M number of chunk servers. If a chunk server fails its copy is either lost (destroy) or it only stores chunks we do not consider, which means no copies are lost (keep). The copies part between master and servers considers the number of copies that are available ($R\_present$) and lost (R_lost) in the system.

The model has three parameters: The number of chunk servers M, the number of chunks S a chunk server can store and the number of chunks N.

9.4.2 Analysis

Not all rates and condition could directly be specified in PNML. This means we needed to adjust the rates and condition of a few transition to get the correct MAPA model. This considers the rates of the transition which depend on a number of tokens in a place and the extra condition for replicate.

We use a small addition to the GSPN model. In our model the probabilities of C_soft and C_hard are not known, which means that these transition can be taken with non-determinism. This makes the model an ND-GSPN and this is also done in [18]. In our analysis we use $S = 5000$, $N = 100000$ and $M \in \{20, 30, 40, 50\}$. We specify three conditions for the analysis:

$$\begin{aligned} sever\_hardware\_disaster &= (M\_hard\_d = 1) \wedge (C\_hard\_d > 0.75 \cdot M) \\ sever\_software\_disaster &= (M\_soft\_d = 1) \wedge (C\_soft\_d > 0.75 \cdot M) \\ service\_level\_1 &= (M\_up = 1) \wedge (R\_present \geq 1) \end{aligned}$$

With the use of these condition we specify our goal states G and our initial state m_0. We consider two initial states. The first is where m_0 consists of all states where the condition *sever_hardware_disaster* holds and the second is where *sever_software_disaster* holds.

We define goal set G is al states where the condition *service_level_1* holds. This can be translated to the following reachability condition which is added to the MAPA specification:

$$reachCondition(M\_up = 1)\ \&\ (R\_present \geq 1)$$

Note that it is not directly possible to use multiple initial states in MAPA. We solved this by first generating the goal states for the conditions *sever_hardware_disaster* and *sever_software_disaster*. The resulting goal states where then used as initial states in the IMCA input file. This also means that we used SCOOP twice to generate the state space.

Analysis Results GEMMA took 0.062 seconds to translate the GSPN model to MAPA (average of three runs). The computation times of SCOOP for generating the model are shown in Table 9.11 (average of three runs). Note that this is the average time of one run of SCOOP, where in practice we needed to run SCOOP twice to get the complete state space as described above.

We check the properties on expected time. This means we check the expected time the system needs to recover from a software or hardware disaster. We check the

minimum and maximum results of the properties and use these as the bounds for an interval. The results with initial states from *sever_hardware_disaster* are shown in Table 9.12 and from *sever_software_disaster* are shown in Table 9.13.

M	$\|S\|$	$\|G\|$	time
20	8736	1953	4.137
30	19096	4278	9.449
40	33456	7503	19.904
50	51816	11628	35.970

Table 9.11: Computation times in seconds of SCOOP for generating the Google File System state space.

M	$\|S\|$	$\|G\|$	$eT^{[min,max]}(m_0, \diamond G)$	min	max
20	8736	1953	[0.1667-21.5236]	4.618	28.137
30	19096	4278	[0.1667-21.5149]	23.620	277.123
40	33456	7503	[0.1667-21.5149]	63.848	1393.891
50	51816	11628	[0.1667-21.5149]	168.483	4351.088

Table 9.12: IMCA results for expected time from *severe_hardware_disaster* and computation times in seconds.

M	$\|S\|$	$\|G\|$	$eT^{[min,max]}(m_0, \diamond G)$	min	max
20	8736	1953	[0.0833- 0.8435]	4.867	28.568
30	19096	4278	[0.0833-0.6698]	26.15	279.372
40	33456	7503	[0.0844-0.6145]	62.967	138.210
50	51816	11628	[0.0835-0.60156]	168.881	4369.223

Table 9.13: IMCA results for expected time from *severe_software_disaster* and computation times in seconds.

Conclusion

- We compare our results with the ones in [18], in which the model-checking is done on an MA that is created by hand and with the tool PRISM [22]. The number of states and goal states is on our case bigger, because in [18] the GSPN is translated to a CTMC, this means that immediate states will disappear from the model. However, the results are exactly the same, which indicates that our translation creates the correct ND-GSPN.
- The computation time of GEMMA is 0.062 seconds for each version of the model. The computation times of SCOOP increase with the number of states. The computation times of IMCA increase more with the increase of states. This means that the efficiency of the tool-chain is determined by IMCA as this tool takes by far the most time for its computations.
- Using a bigger M results in more chunk severs. However, the expected time to recover from severe hardware or software disaster is not influenced, when we have more chunk servers. This is explained by the goal state G, which mostly depends on the state of the master server and not on the chunk servers.

Chapter 10

Conclusion

10.1 Summary

ND-GSPN In this thesis we defined the non-deterministic GSPN (ND-GSPN) in Section 5, which adds non-determinism to the GSPN [28] model. We also expressed this ND-GSPN formally as an MA [13] in Section 5.2 and defined how we can specify ND-GSPNs in PNML [45] in Section 5.3.2.

We added non-determinism to the GSPN model and defined the more general ND-GSPN. This means we can now be more specific in how to resolve conflicts. We can now chose to not resolve a conflict and thus use non-determinism. This means we can model-check more specifically with the use of weights and priorities. It is now possible to combine the use of weights with non-determinism.

We also added extra arcs to the definition of ND-GSPNs. *Weighted inhibitor* arcs require a place to have less than a certain amount of tokens, instead of checking of a place is empty. *Reset arcs* can empty a place.

Probabilistic Arc We introduced the *probabilistic arc* in Chapter 7. This arc can make the result of a transition probabilistically distributed, which means we can define probability more local. The probabilistic arc does not introduce new behaviour, but can be used to model more intuitive and efficient. Probabilistic arcs are useful in situations were we want to define a local probabilistic choice, which has nothing to do with other elements of the model.

Translation of ND-GSPNs to MAPA To perform analysis on our ND-GSPNs, we created a translation from ND-GSPNs to MAPA [41]. An ND-GSPN specified as PNML can be translated to an MA specified as MAPA. We chose MAPA as this is the import language for the tool SCOOP [40]. SCOOP can reduce its models and is connected directly with IMCA [2], which is the only model-checker for MAs at the moment. A translation from PNML to MAPA makes the overview in Figure 1.1 complete. The translation is formally shown in Section 6.1 and proven correct in Section 6.2.

GEMMA We implemented this translation from ND-GSPNs in PNML to MAPA in the tool GEMMA, which is described in Chapter 8. GEMMA does not concern state space generation or model-checking, as these parts are the responsibility of SCOOP and IMCA as shown in Figure 1.1. With the tool-chain GEMMA, SCOOP and IMCA it is possible to model-check ND-GSPNs specified in PNML.

GEMMA is written in the functional language Haskell and consist of 300 lines of code. GEMMA translates only on syntactical level, which means it is very fast. GEMMA has the option to omit all weights or priorities and thus translate to a MAPA specification containing an IMC instead of an MA. GEMMA, SCOOP and IMCA are directly connected via a web-interface as shown in Figure 8.2. It is possible to immediately model-check an ND-GSPN with this web-interface.

Case studies In Chapter 9 we did three case studies to show our approach of model-checking and to check if our translation is correct. The case studies were also done using the old approach of model-checking GSPNs (translation to CTMC). We used the case studies to check if the results of our approach are the same as the result of the old approach. We also used the case studies to show that we can now handle more policies for resolving conflicts between transitions. We created the models in PNML and translated them with GEMMA to MAPA. We then used SCOOP to generate the MA and IMCA to perform the analysis.

We did a workstation cluster [20], a tank fluid system [12] and the Google file system [19] case studies. We model-checked on time-bounded reachability, unbounded reachability, expected time and long-run average. We got the expected results compared to the results of the old approach, which indicates that GEMMA implements the translation to MAPA correct. In the workstation cluster case we showed that we now can handle more different policies to resolve conflicts between immediate transitions. The use of non-determinism gives an interval defined by the minimal and maximal results of the checked properties. The results of any other policy is always within this interval. We are now able to define policies, which partly use weights and partly use non-determinism to resolve conflicts.

Tool-chain The efficiency of the tool chain GEMMA, SCOOP and IMCA is mostly determined by IMCA. The computation times of GEMMA are small as it only translates the model syntactically. The computation times of SCOOP depend on the number of states in the model. Especially for bigger models the computation times of IMCA can increase greatly. We showed in the workstation cluster and the Google file system case studies that IMCA had by far the longest computation times for the bigger models. This means that making SCOOP or GEMMA faster will not have a significant effect on the complete tool-chain.

Compared to the old way of model-checking on CTMCs, we see that our approach is slower. Model-checking on CTMCs is more efficient and thus faster. However, our approach allows the model-checking of ND-GSPNs and thus we can model-check more general and specific.

10.2 Future Work

Model-checking The power of the model-checkers for MAs should increase. Currently IMCA is the only model-checker for MAs. As we showed in the case studies the computation time of IMCA can increase greatly for bigger models. A more powerful model-checker for MAs would allow us to analysis bigger models.

GEMMA GEMMA could be extended to support more types of Petri nets. GEMMA could support coloured Petri nets (CPN) [23] for example. CPNs have multiple types of tokens. A place has thus multiple types of each token and a transition requires a subset of these types. CPNs were not considered in this project as in the simple case they do not introduce any new behaviour. However, there are some interesting features

in CPNs which could be included in GEMMA, for example being able to specify extra conditions for transitions. There may also be other types or variants of Petri Net, that can be added to the ND-GSPN definition. The translation to the MA model may also give more options to extend the ND-GSPN definition.

SCOOP SCOOP should focus on generating a smaller state space instead of decreasing its computation time. This can decrease the computation time of the whole tool-chain GEMMA, SCOOP and IMCA as explained in the previous section. Confluence reduction [43] for MAPA models could for example be used to reduce the state space.

SCOOP generates the state space from the MAPA specification. It uses several reduction techniques to minimize this process, as described in Section 4.1. However, the state space generation itself could also be optimized. A tool set which is specialised for this is the LTSmin [7] tool set. SCOOP could implement a module in LTSmin and let LTSmin generate the state space. LTSmin can then be linked to tools as IMCA. Implementing a module in LTSmin for SCOOP might not be trivial, as LTSmin has not been used for probabilistic models. This means some research is required to check if these models can be generated by LTSmin.

SCOOP could also use case simplification techniques. In our translation it is possible that a condition for a summation is a disjunction of multiple cases. Several of these cases could be omitted as they are subsumed by other more general cases. An example of such a simplification:

$$p_1 \geq 1 \vee p_1 \geq 2 \equiv p_1 \geq 1$$

This kind of simplifications would reduce the amount of statement checking of SCOOP.

Tooling Currently there is no tooling for ND-GSPNs. Tools as PIPE [8] can model GSPNs, but they usually do not support the use of non-determinism. This means these tool should be more liberal in conflict resolving. The tools can be easily adjusted as the restriction that a transition must have a weight can be omitted. A tool that could model ND-GSPNs can be linked to the web-interface, as described in Figure 8.2. This would make it possible to graphically create an ND-GSPN and immediately model-check it with IMCA via the web-interface. Under water the ND-GSPN would then be saved as PNML, translated by GEMMA to MAPA, SCOOP would generate the state space and IMCA would analyse.

PNML standard The current version of PNML [45] does not support GSPNs. The DTD described in Section 5.3.2 could be used as base for a new version of the PNML standard. This standard could also contain the specification of a reachability condition, which is currently specified directly in MAPA. It would be nice if the condition could be specified within the PNML of the model. The condition should then be translated with the model to the corresponding MAPA specification. GEMMA should be extended to support such a feature. The condition consist of the markings you want to reach.

Parallel Composition The translation from ND-GSPNs to MAs creates the opportunity for parallel composition for ND-GSPNs. This should be possible as parallel composition is defined for MAs as described in Section 2.7. For this it is required to specify how ND-GSPNs should synchronize. This could be done by labelling transition on which should be synchronized.

Bibliography

[1] Haskell xml toolbox. http://www.fh-wedel.de/ si/HXmlToolbox/index.html, 2012.

[2] Interactive Markov Chain Analyzer. http://www-i2.informatik.rwth-aachen.de/imca/, 2012.

[3] M. Timmer at University of Twente. Web-interface for SCOOP. http://wwwhome.cs.utwente.nl/ timmer/scoop/webbased.html, 2012.

[4] Soheib Baarir, Marco Beccuti, Davide Cerotti, Massimiliano De Pierro, Susanna Donatelli, and Giuliana Franceschinis. The greatspn tool: recent enhancements. *SIGMETRICS Perform. Eval. Rev.*, 36(4):4–9, March 2009.

[5] C. Baier and J.P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[6] R. Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.

[7] S.C.C. Blom, J.C. van de Pol, and M. Weber. Bridging the gap between enumerative and symbolic model checkers. Technical Report TR-CTIT-09-30, Centre for Telematics and Information Technology University of Twente, Enschede, June 2009.

[8] P. Bonet, C. Lladó, R. Puigjaner, and W. J. Knottenbelt. PIPE v2.5: A Petri Net Tool for Performance Modelling. In *23rd Latin American Conference on Informatics (CLEI 2007)*, October 2007.

[9] H. Boudali, P. Crouzen, B.R. Haverkort, M. Kuntz, and M.I.A. Stoelinga. Architectural dependability evaluation with arcade. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 512 –521, june 2008.

[10] M. Ĉepin and M. Borut. A dynamic fault tree. *Reliability Engineering and System Safety*, 75(1):83 – 91, 2002.

[11] G. Chiola, M.A. Marsan, G. Balbo, and G. Conte. Generalized stochastic petri nets: a definition at the net level and its implications. *Software Engineering, IEEE Transactions on*, 19(2):89 –107, feb 1993.

[12] D. Codetta-Raiteri and A. Bobbio. Stochastic petri nets supporting dynamic reliability evaluation. In *International Journal of Materials and Structural Reliability*, Vol.4, No.1, pages 65–77, 2006.

[13] C. Eisentraut, H. Hermanns, and L. Zhang. On probabilistic automata in continuous time. *Annual Symposium on Logic in Computer Science*, pages 342 – 351, 2010.

[14] G. Florin and S. Natkin. Les reseaux de petri stochastiques. In *Technique et Science Informatiques*, volume 4, February 1985.

[15] H. Garavel, R. Mateescu, F. Lang, and W. Serwe. Cadp 2006: a toolbox for the construction and analysis of distributed processes. In *Proceedings of the 19th international conference on Computer aided verification*, CAV'07, pages 158–163, Berlin, Heidelberg, 2007. Springer-Verlag.

[16] Gobioff H. Ghemawat, S. and S.T. Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

[17] J.F. Groote. The syntax and semantics of timed μcrl. Technical report, Amsterdam, The Netherlands, The Netherlands, 1997.

[18] D. Guck. Quantitative analysis of markov automata. In *Master Thesis*, pages 55–67, RWTH Aachen University, 2012.

[19] B. Haverkort, L. Cloth, H. Hermanns, J.P. Katoen, and C. Baier. Model-checking performability properties. In *Proceedings of the International Conference on Dependable Systems and Networks, DSN 2002*, pages 103–112, Los Alamitos, 2002. IEEE Computer Society Press.

[20] B. R. Haverkort, H. Hermanns, and J.P. Katoen. On the use of model checking techniques for quantitative dependability evaluation. In *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems, SRDS 2000*, pages 228–237, Los Alamitos, 2000. IEEE Computer Society Press.

[21] H. Hermanns and J.P. Katoen. The how and why of interactive markov chains. In *Proceedings of the 8th international conference on Formal methods for components and objects*, FMCO'09, pages 311–337, Berlin, Heidelberg, 2010. Springer-Verlag.

[22] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In H. Hermanns and J. Palsberg, editors, *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[23] K. Jensen. A brief introduction to coloured petri nets. In Ed Brinksma, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1217 of *Lecture Notes in Computer Science*, pages 203–208. Springer Berlin / Heidelberg, 1997. 10.1007/BFb0035389.

[24] J.-P. Katoen. Gspns revisited: Simple semantics and new analysis algorithms. In *Application of Concurrency to System Design (ACSD), 2012 12th International Conference on*, pages 6 –11, june 2012.

[25] J.P. Katoen, M. Khattri, and I.S. Zapreev. A Markov reward model checker. In *Quantitative Evaluation of Systems (QEST)*, pages 243–244, Los Alamos, CA, USA, 2005. IEEE Computer Society.

[26] J.P. Katoen, J.C. van der Pol, M.I.A. Stoelinga, and M. Timmer. A linear process-algebraic format with data for probabilistic automata. *Theoretical Computer Science*, 413(1):36–57, January 2012.

[27] D. Lehmann and M.O. Rabin. On the advantages of free choice: a symmetric and fully distributed solution to the dining philosophers problem. In *Proceedings of the 8th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '81, pages 133–138, New York, NY, USA, 1981. ACM.

[28] M.A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling With Generalised Stochastic Petri Nets*. John Wiley and Sons, 2004.

[29] M.A. Marsan, G. Conte, and G. Balbo. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Trans. Comput. Syst.*, 2(2):93–122, May 1984.

[30] P. Merlin and D. Farber. Recoverability of communication protocols–implications of a theoretical study. *Communications, IEEE Transactions on*, 24(9):1036 – 1043, sep 1976.

[31] M.K. Molloy. Performance analysis using stochastic petri nets. *Computers, IEEE Transactions on*, C-31(9):913 –917, sept. 1982.

[32] M.R. Neuhäusser. *Model checking nondeterministic and randomly timed systems*. PhD thesis, Enschede, January 2010. CTIT Ph.D.-Thesis Series No. 09-165, ISSN 1381-3617.

[33] University of Hamburg. Petri nets world. http://www.informatik.uni-hamburg.de/TGI/PetriNets/, 2012.

[34] E. Parzen. *Stochastic processes*. Holden-Day, San Francisco, 1965.

[35] C.A. Petri. *Communication with automata*. PhD thesis, Universitt Hamburg, 1966.

[36] C. Ramchandani. Analysis of asynchronous concurrent systems by timed petri nets. Technical report, Cambridge, MA, USA, 1974.

[37] J. Sifakis. Performance evaluation of systems using nets. In Wilfried Brauer, editor, *Net Theory and Applications*, volume 84 of *Lecture Notes in Computer Science*, pages 307–319. Springer Berlin / Heidelberg, 1980.

[38] M.I.A. Stoelinga. An introduction to probabilistic automata. *Bulletin of the European Association for Theoretical Computer Science*, 78:176–198, 2002.

[39] F. J. W. Symons. *Modeling and Analysis of Communication Protocols Using Numerical Petri Nets*. PhD thesis, University of Essex, 1978.

[40] M. Timmer. SCOOP: A tool for symbolic optimisations of probabilistic processes. In C. Palamidessi and A. Riska, editors, *8th International Conference on Quantitative Evaluation of SysTems, QEST 2011*, Los Alamitos, USA, 2011. IEEE Computer Society.

[41] M. Timmer, J. P. Katoen, J. C. van de Pol, and M. I. A. Stoelinga. Efficient modelling and generation of markov automata. In M. Koutny and I. Ulidowski, editors, *Proceedings of the 23rd International Conference on Concurrency Theory, CONCUR 2012, Newcastle upon Tyne, United Kingdom*, volume 7454 of *Advanced Research in Computing and Software Science, Lecture Notes in Computer Science (ARCoSS/LNCS)*, pages 364–379, Berlin, September 2012. Springer Verlag.

[42] K. Trivedi and V. Kulkarni. Fspns: Fluid stochastic petri nets. In M.A. Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 24–31. Springer Berlin / Heidelberg, 1993.

[43] J.C. van der Pol, M.I.A. Stoelinga, and M. Timmer. Confluence reduction for probabilistic systems. In P. A. Abdulla and K. R. M. Leino, editors, *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2011, Saarbrücken, Germany*, volume 6605 of *Lecture Notes in Computer Science*, pages 311–325, Berlin, March 2011. Springer Verlag.

[44] J.C. van der Pol and M. Timmer. State space reduction of linear processes using control flow reconstruction. In Z. Liu and A. P. Ravn, editors, *ATVA 2009 - Automated Technology for Verification and Analysis. 7th International Symposium, Macao SAR, China*, volume 5799 of *Lecture Notes in Computer Science*, pages 54–68, Berlin, October 2009. Springer Verlag.

[45] M. Weber, E. Kindler, Technische Universitt München, and Fakultät Für Inforatik. The Petri Net Markup Language. In *Petri Net Newsletter*, pages 124–144. Springer, 2000.

[46] L. Zhang and M. Neuhäuß er. Model checking interactive markov chains. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science*, pages 53–68. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-12002-2_5.