Quantitative model checking on probabilistic systems using $pL\mu^{\odot}_{\oplus}$

Olav Bunte

November 13, 2017

Msc Thesis
Computer Science and Engineering

Supervised by Tim A.C. Willemse

Eindhoven University of Technology
Department of Mathematics and Computer Science

Abstract

Although most model checking logics focus on proving qualitative properties, with the logic $pL\mu^{\odot}_{\oplus}$ introduced in Mio's PhD thesis [Mio12] one can check the probability that some behaviour happens. In this work we try to bring this logic to practice. We attempt to find intuitive meaning for $pL\mu^{\odot}_{\oplus}$-formulas with the aim to create guidelines to create meaningful formulas. Also, we give an alternative representation of a $pL\mu^{\odot}_{\oplus}$ model checking problem in form of an equations system and we give an algorithm to extract the solution from this representation, based on the work of Mader [Mad97]. This algorithm will be compared to an approximation algorithm by applying both on a number of use cases.

Contents

1 Introduction

It is very difficult to get a software system completely bug free. By testing one can find and squash most bugs, but it is near impossible to cover all possible behaviour of the system. A more thorough way of making sure that some property holds for the system is model checking. In this approach, one specifies a property in terms of a logical formula and applies it to a transition system that models the behaviour of the software system. In [Koz83] Kozen introduces such a logic, the modal μ-calculus $L\mu$, which can be used to verify (boolean) properties on labeled transition systems. A lot of research has already gone in this logic (and its extensions) and it has even found its uses in institutions [HKW11] and companies [vBGH$^+$17].
However, sometimes one would like to prove more than whether some property is simply either true of false. Especially in the context of systems that also contain probabilistic behaviour, it might be desirable to compute the exact probability that some type of behaviour happens. For instance, in a system where data is sent over a lossy channel one might want to quantify the reliability by computing the probability that this data will actually arrive at the receiving end of the channel. In the PhD thesis of Mio [Mio12] the logic $pL\mu^{\odot}_{\oplus}$ is introduced, which allows for this type of model checking.

Mio's PhD thesis however is very theoretical. Therefore, in this work we will focus somewhat more on practical aspects. We will attempt to find the (intuitive) meaning for a selection of $pL\mu^{\odot}_{\oplus}$-formulas and to create some guidelines based on this for using $pL\mu^{\odot}_{\oplus}$ in practice. To solve a $pL\mu^{\odot}_{\oplus}$ model checking problem we will give an approximation algorithm. To compute the solution exactly, we will give an alternative representation in the form of an equation system which we will name $\mathrm{RES}^{\odot}_{\oplus}$ and an algorithm to extract the exact solution from this representation, both inspired by the work of Mader [Mad97]. Both algorithms will be compared to each other by applying them on a number of use cases.

The outline of the thesis is as follows. Firstly we will give some background knowledge on the relevant mathematical subjects, transition systems and quantitative verification in sections 2, 3 and 4. In section 5 we will discuss what has been done before in the field of verification on probabilistic systems, focusing on adaptations of the modal μ-calculus of Kozen that have lead to $pL\mu^{\odot}_{\oplus}$. The logic $pL\mu^{\odot}_{\oplus}$ will be introduced in section 6, where we will also investigate the semantics and meaning of various $pL\mu^{\odot}_{\oplus}$-formulas. In section 7 we will shortly give an approximation algorithm and introduce the $\mathrm{RES}^{\odot}_{\oplus}$ with an algorithm to solve it. Both algorithms will be tested on a number of use cases and we will consider changes to the algorithm for $\mathrm{RES}^{\odot}_{\oplus}$'s to reduce its running time in section 8. Lastly we conclude with section 9.
If one is only interested in the algorithmic part of this thesis, sections 5 and 6 can be skipped.

2 Mathematical Background

2.1 Probability theory

Probability theory is the theory for analyzing random events. Given a collection of distinct outcomes Ω, an event is a set of such outcomes. Let A and B be such events. Then we denote the probability that if an outcome happens, it is in A, or rather that event A happens with $P(A)$. The value of a probability is always a real number in the interval $[0, 1]$. Since events are sets, we can denote the probability that both events A and B happen with $P(A \cap B)$ and the probability that event A or event B happens with $P(A \cup B)$. The above probabilities are related to each other in the following addition rule:

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) \tag{1}$$

We denote the complement of A as $\overline{A}$ such that $P(\overline{A}) = 1 - P(A)$. Intuitively, this is the probability that event A does not happen. The event that equals the set of all possible outcomes Ω is called the universe, for which $P(\Omega) = 1$.
Given events A and B, the values of $P(A \cup B)$ and $P(A \cap B)$ depend on how much A and B overlap. This is known as dependence. We denote the probability that event A happens given that event B happens, where $P(B) > 0$, with $P(A|B)$. This probability relates to the others by the product rule:

$$P(A \cap B) = P(A|B) \cdot P(B) \tag{2}$$

Applying this equation twice, assuming $P(A) > 0$ as well, results in Bayes' theorem [Fel68]:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{3}$$

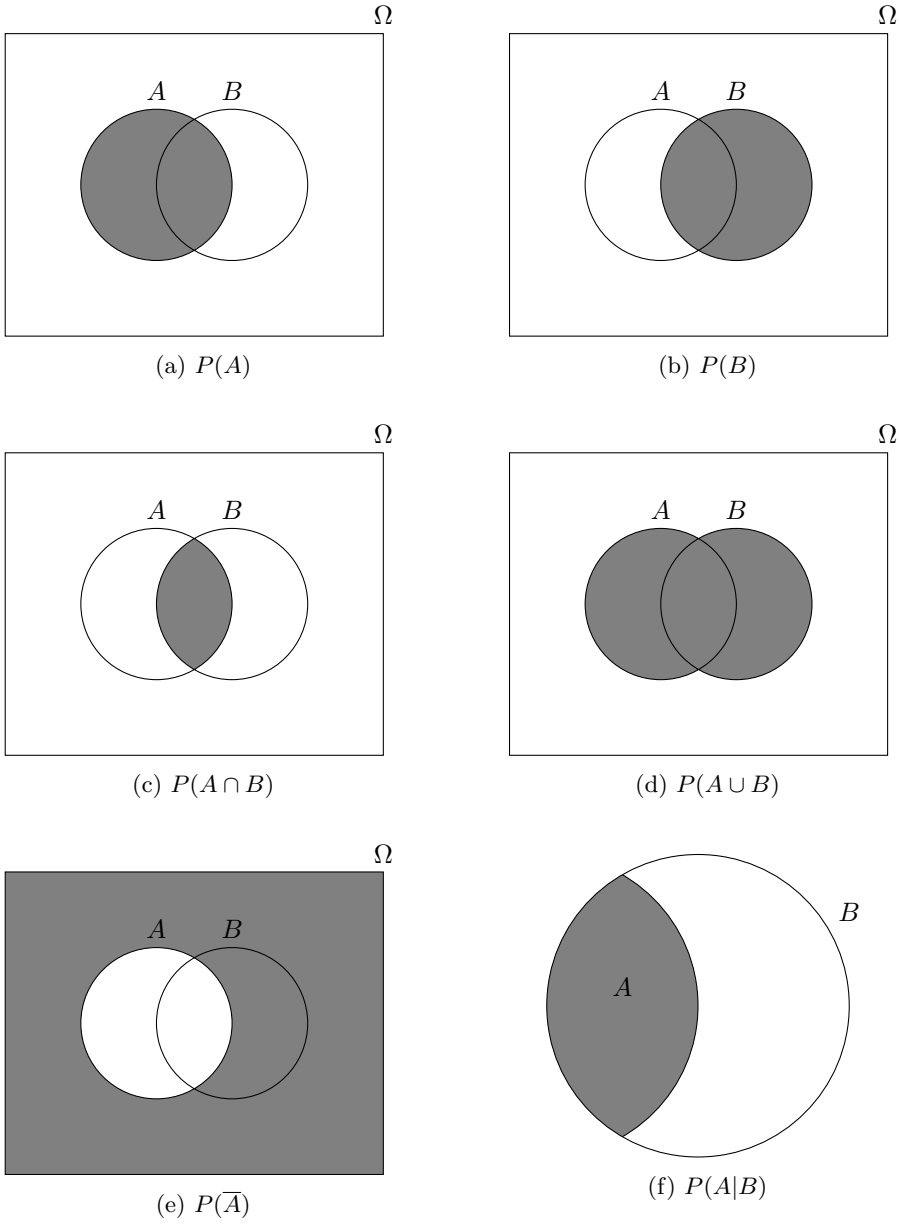
See figure 1 for some visual examples. For a practical example, see the following.

Example 2.1. Consider a six-sided die. There are a total of six outcomes, namely the numbers 1 through 6, all with the same probability of $\frac{1}{6}$. Let $A_{>1} = \{2, 3, 4, 5, 6\}$ be the event of throwing a value greater than 1 and let $B_{odd} = \{1, 3, 5\}$ be the event of throwing an odd number. Then the probability $P(A_{>1})$ of throwing a value greater than 1 equals $\frac{5}{6}$, the probability $P(A_{>1} \cap B_{odd})$ of throwing an odd value greater than 1 equals $\frac{1}{3}$, the probability $P(A_{>1} \cup B_{odd})$ to throw an odd value or a value greater than 1 equals 1 and the probability $P(B_{odd}|A_{>1})$ to throw an odd value given a value greater than 1 will be thrown equals $\frac{2}{5}$.

If $P(A)$ and $P(B)$ are known but the degree of dependence is unknown, there is a range of values possible for $P(A \cap B)$. Similarly to the work in [FBH83], we differentiate between three types of dependence.

Definition 2.1. A is *positively dependent* on B iff $P(A \cap B) > P(A) \cdot P(B)$. We will write this as $B \nearrow A$ (B supports A).

Definition 2.2. A is *independent* of B iff $P(A \cap B) = P(A) \cdot P(B)$. We will write this as $B \perp A$.

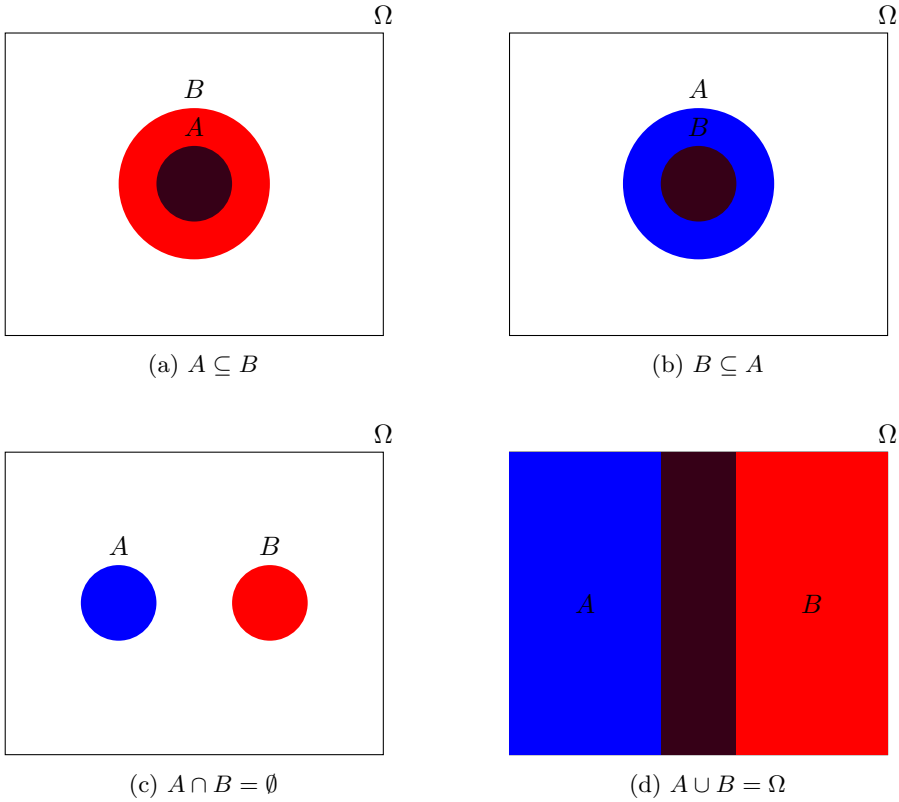


(a) $P(A)$ (b) $P(B)$

(c) $P(A \cap B)$ (d) $P(A \cup B)$

(e) $P(\overline{A})$ (f) $P(A|B)$

Figure 1: Visualization of probabilities. The value of the probability below a figure is the area coloured grey in the figure, given that the total area is equal to 1.



(a) $A \subseteq B$ (b) $B \subseteq A$

(c) $A \cap B = \emptyset$ (d) $A \cup B = \Omega$

Figure 2: Visualization of maximal dependence cases. In the top two cases A and B are maximally positively dependent and in the bottom two cases A and B are maximally negatively dependent. Event A is coloured blue, Event B is coloured red and where they overlap $(A \cap B)$ is coloured dark purple.

Definition 2.3. A is *negatively dependent* on B iff $P(A \cap B) < P(A) \cdot P(B)$. We will write this as $B \searrow A$ (B weakens A).

All three relations are symmetric and intransitive and due to their definition only one of the three relations can apply to any A and B. We will shortly pay extra attention to the two outer extremes of the range of possible values for $P(A \cap B)$. In case of the upper bound of $P(A \cap B)$ (A and B overlap as much as possible) we say that A and B are *maximally positively dependent*. In case of the lower bound of $P(A \cap B)$ (A and B overlap as little as possible) we say that A and B are *maximally negatively dependent*. Dually, the upper bound of $P(A \cup B)$ corresponds to maximal positive dependence and the lower bound of $P(A \cup B)$ corresponds to maximal negative dependence. See figure 2 for situations where these maximal dependencies occur.

More formally, we can look at events in terms of a probability space [Kol50].

Definition 2.4. A *probability space* is a tuple $\langle \Omega, \mathcal{F}, P \rangle$ where

- Ω is a set of outcomes,

- $\mathcal{F} \subseteq 2^\Omega$ is a set of events such that $\mathcal{F}$ is closed under countable union, countable intersection and complement,
- $P : \mathcal{F} \to [0,1]$ is a function assigning probabilities to events, such that $P(\Omega) = 1$ and for any set $S \subseteq \mathcal{F}$ of pairwise disjoint sets $P(\bigcup_{A \in S} A) = \sum_{A \in S} P(A)$.

If the restrictions mentioned above are met, the probability function P and the probability space are well defined. We will only consider events that can be expressed in a well-defined probability space.

A probability function $D \to [0,1]$ like P mapping elements of some countable set D to probability values is called a (discrete) probability distribution, usually denoted by δ. We call a probability distribution δ over D valid iff $\sum_{d \in D} \delta(d) = 1$. We call a probability distribution deterministic iff there is a $d \in D$ such that $\delta(d) = 1$ and $\delta(d') = 0$ for all $d' \in D$ such that $d \neq d'$.

2.2 Lattices and fixpoints

The theory that we will focus on is based on lattices [Bir40] and fixpoints [I+81]. Therefore, we will introduce these notions first.

2.2.1 Orders and lattices

Definition 2.5. A *partial order*, usually denoted with $\leq$, is a binary relation on a set Z such that $\leq$ is:

- reflexive: $z \leq z$ for all $z \in Z$,
- antisymmetric: if $y \leq z$ and $z \leq y$, then $y = z$ for all $y, z \in Z$,
- transitive: if $x \leq y$ and $y \leq z$, then $x \leq z$ for all $x, y, z \in Z$.

In case Z is a set of functions, we will use the pointwise order.

Definition 2.6. Let $Z = X \to Y$ and let $\leq$ be some order on Y. Then $\dot{\leq}$ is the *pointwise order* on Z such that $f \dot{\leq} g$ iff $f(x) \leq g(x)$ for all $x \in X$.

Definition 2.7. An *ordered set* is a set equipped with a partial order. We will denote this with $\langle Z, \leq \rangle$, where Z is a set and $\leq$ is a partial order on Z.

Lemma 2.1. Let $\langle Z, \leq \rangle$ be an ordered set and let $Z' \subseteq Z$. Then $\langle Z', \leq \rangle$ is also an ordered set.

Definition 2.8. An ordered set $\langle Z, \leq \rangle$ has a *top element*, denoted with $\top$, if for all $z \in Z$ it holds that $z \leq \top$. Dually, an ordered set $\langle Z, \leq \rangle$ has a *bottom element*, denoted with $\bot$, if for all $z \in Z$ it holds that $\bot \leq z$.

Definition 2.9. Let $\langle Z, \leq \rangle$ be an ordered set and let $Z' \subseteq Z$. Then $z \in Z$ is an *upper bound* of Z' if for all $z' \in Z'$ $z' \leq z$. Dually, $z \in Z$ is a *lower bound* of Z' if for all $z' \in Z'$ $z \leq z'$.

Definition 2.10. Let $\langle Z, \leq \rangle$ be an ordered set, let $Z' \subseteq Z$ and let $Z'_{\uparrow} \subseteq Z$ and $Z'_{\downarrow} \subseteq Z$ be the set of upper and lower bounds of Z' respectively. Then the bottom element of $Z'_{\uparrow}$, if it exists, is the *least upper bound* or *supremum* of Z', denoted by $\bigsqcup_{z' \in Z'} z'$ or $\bigsqcup Z'$. Dually, the top element of $Z'_{\downarrow}$, if it exists, is the *greatest lower bound* or *infimum* of Z', denoted by $\bigsqcap_{z' \in Z'} z'$ or $\bigsqcap Z'$.

In case Z' consists only of two elements x and y, we denote the supremum and infimum with $x \sqcup y$ and $x \sqcap y$ respectively. In case of the empty set we define $\bigsqcup \emptyset = \bot$ and $\bigsqcap \emptyset = \top$.

Definition 2.11. Let $\langle Z, \leq \rangle$ be an ordered set. Then $\langle Z, \leq \rangle$ is a *lattice* if for all $y, z \in Z$, $y \sqcup z$ and $y \sqcap z$ exist. If for all $Z' \subseteq Z$ we have that $\bigsqcup Z'$ and $\bigsqcap Z'$ exist, the lattice is *complete*.

Definition 2.12. Let $\langle Z, \leq \rangle$ be a lattice. This lattice is *distributive* if for all $x, y, z \in Z$ it holds that

$$x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$$
$$x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$$

Lemma 2.2. Let $\langle Z, \leq \rangle$, $\langle Z_1, \leq \rangle$, $\langle Z_2, \leq \rangle$, $\langle Z_1 \cup Z_2, \leq \rangle$ and $\langle Z_1 \cap Z_2, \leq \rangle$ be complete lattices where $Z_1, Z_2 \subseteq Z$. Then the following four (in)equations hold:

(a) $\bigsqcap(Z_1 \cup Z_2) = \bigsqcap Z_1 \sqcap \bigsqcap Z_2$

(b) $\bigsqcup(Z_1 \cup Z_2) = \bigsqcup Z_1 \sqcup \bigsqcup Z_2$

(c) $\bigsqcap(Z_1 \cap Z_2) \geq \bigsqcap Z_1 \sqcup \bigsqcap Z_2$

(d) $\bigsqcup(Z_1 \cap Z_2) \leq \bigsqcup Z_1 \sqcap \bigsqcup Z_2$

Proof. We will prove a and c. The others follow due to duality.

(a) Since $Z_1, Z_2 \subseteq Z_1 \cup Z_2$ and since both are complete lattices, we know that $\bigsqcap Z_1 \sqcap \bigsqcap Z_2 \in Z_1 \cup Z_2$. Since $\bigsqcap(Z_1 \cup Z_2) \leq a$ for $a \in Z_1 \cup Z_2$, we know that $\bigsqcap(Z_1 \cup Z_2) \leq \bigsqcap Z_1 \sqcap \bigsqcap Z_2$.
By definition of $\cup$, there does not exist an $a \in Z_1 \cup Z_2$ that is not in Z_1 or Z_2. Since $\bigsqcap Z_1 \leq a_1$ and $\bigsqcap Z_2 \leq a_2$ for all $a_1 \in Z_1$ and $a_2 \in Z_2$, this implies that there does not exist an $a \in Z_1 \cup Z_2$ such that $a < \bigsqcap Z_1 \sqcap \bigsqcap Z_2$, so $\bigsqcap(Z_1 \cup Z_2) \geq \bigsqcap Z_1 \sqcap \bigsqcap Z_2$

(b) Is dual to the proof of a above.

(c) By definition of $\bigsqcap$, $\bigsqcap Z_1 \leq a_1$ and $\bigsqcap Z_2 \leq a_2$ for all $a_1 \in Z_1$ and $a_2 \in Z_2$. Since $Z_1 \cap Z_2 \subseteq Z_1$ and $Z_1 \cap Z_2 \subseteq Z_2$, we also have that $\bigsqcap Z_1 \leq a$ and $\bigsqcap Z_2 \leq a$ for all $a \in Z_1 \cap Z_2$, which implies that $\bigsqcap(Z_1 \cap Z_2) \geq \bigsqcap Z_1 \sqcup \bigsqcap Z_2$.

(d) Is dual to the proof of c above.

□

The reason for the inequality in lemmas 2.2c and 2.2d is that the infimum (or supremum) of Z_1 and Z_2 may not be contained in $Z_1 \cap Z_2$, which is shown by the example below.

Example 2.2. Let $Z = [0,3]$, then $\langle Z, \leq \rangle$ is a complete lattice. Let $Z_1 = [0,2]$ and $Z_2 = [0,1] \cup \{3\}$ and thus $Z_1 \cap Z_2 = [0,1]$, then $\langle Z_1, \leq \rangle$, $\langle Z_2, \leq \rangle$ and $\langle Z_1 \cap Z_2, \leq \rangle$ are complete lattices. Then $\bigsqcup Z_1 = 2$ and $\bigsqcup Z_2 = 3$ which both are not elements of $Z_1 \cap Z_2$. This results in the inequality (as lemma 2.2d states) $\bigsqcup(Z_1 \cap Z_2) = \bigsqcup[0,1] = 1 \leq 2 = 2 \sqcap 3 = \bigsqcup Z_1 \sqcap \bigsqcup Z_2$.

2.2.2 Fixpoints

Definition 2.13. Let $f : Z \to Z$ be a function. Then $z \in Z$ is a *fixpoint* of f iff $f(z) = z$.

In 1928 Bronisław Knaster and Alfred Tarski published their well known theorem on fixpoints [KT28]:

Theorem 2.1. Let $\langle Z, \leq \rangle$ be a complete lattice and let $f : Z \to Z$ be a monotone function. Then the set of all fixpoints of f in Z is also a complete lattice.

From this it follows that f has a unique least (μ) and a unique greatest (ν) fixpoint:

$$\mu X.f(X) = \bigsqcap\{z \in Z \mid f(z) \leq z\}$$
$$\nu X.f(X) = \bigsqcup\{z \in Z \mid f(z) \geq z\}$$

where $X \in Z$ is a variable. Note that $\langle \{z \in Z \mid f(z) \leq z\}, \leq \rangle$ and $\langle \{z \in Z \mid f(z) \geq z\}, \leq \rangle$ are complete lattices as well.
The least and greatest fixpoint can also be defined recursively. Let $f_0 = \bot$, $f_{\alpha+1} = f(f_\alpha)$, $f_\lambda = \bigsqcup\{f_\alpha \mid \alpha < \lambda\}$, $f^0 = \top$, $f^{\alpha+1} = f(f^\alpha)$ and $f^\lambda = \bigsqcap\{f^\alpha \mid \alpha < \lambda\}$ for every limit ordinal λ then

$$\mu X.f(X) = \bigsqcup\{f_\alpha \mid \textit{ordinal } \alpha\}$$
$$\nu X.f(X) = \bigsqcap\{f_\alpha \mid \textit{ordinal } \alpha\}$$

This definition induces an iterative method of finding such fixpoints. When looking for $\mu X.f(X)$, starting with f_0, we can iteratively compute the next f_α. When we reach a f_β such that $f_\beta = f_{\beta-1}$, we have found the fixpoint $\mu X.f(X)$. Similarly, we can compute $\nu X.f(X)$ using f^α. However, depending on the lattice this iterative method may not terminate.
In this paper we will use the fixpoint sign σ to denote either μ or ν when in the context of fixpoints if it does not matter which of the two it is.

As will prove useful later on, the least and greatest fixpoint have properties similar to distributivity.

Lemma 2.3. Let $\langle Z, \leq \rangle$ be a complete lattice, let $f_1, f_2 : Z \to Z$ be two monotone functions and let $X \in Z$ be a variable. Then the following four (in)equalities hold:

(a) $\mu X.(f_1(X) \sqcap f_2(X)) = \mu X.f_1(X) \sqcap \mu X.f_2(X)$

(b) $\nu X.(f_1(X) \sqcup f_2(X)) = \nu X.f_1(X) \sqcup \nu X.f_2(X)$

(c) $\mu X.(f_1(X) \sqcup f_2(X)) \geq \mu X.f_1(X) \sqcup \mu X.f_2(X)$

(d) $\nu X.(f_1(X) \sqcap f_2(X)) \leq \nu X.f_1(X) \sqcap \nu X.f_2(X)$

Proof. We will prove a and c. The others follow from duality.

(a)

$$\begin{aligned}
\mu X.(f_1(X) \sqcap f_2(X)) &= \bigsqcap\{z \in Z \mid f_1(z) \sqcap f_2(z) \leq z\} \\
&= \bigsqcap\{z \in Z \mid f_1(z) \leq z \vee f_2(z) \leq z\} \\
&= \bigsqcap(\{z \in Z \mid f_1(z) \leq z\} \cup \{z \in Z \mid f_2(z) \leq z\}) \\
\{\textit{lemma 2.2a}\} &= \bigsqcap\{z \in Z \mid f_1(z) \leq z\} \sqcap \bigsqcap\{z \in Z \mid f_2(z) \leq z\} \\
&= \mu X.f_1(X) \sqcap \mu X.f_2(X)
\end{aligned}$$

(b) Is dual to the proof of a above, but using lemma 2.2b instead.

(c)

$$\begin{aligned}
\mu X.(f_1(X) \sqcup f_2(X)) &= \bigsqcap\{z \in Z \mid f_1(z) \sqcup f_2(z) \leq z\} \\
&= \bigsqcap\{z \in Z \mid f_1(z) \leq z \wedge f_2(z) \leq z\} \\
&= \bigsqcap(\{z \in Z \mid f_1(z) \leq z\} \cap \{z \in Z \mid f_2(z) \leq z\}) \\
\{\textit{lemma 2.2c}\} &\geq \bigsqcap\{z \in Z \mid f_1(z) \leq z\} \sqcup \bigsqcap\{z \in Z \mid f_2(z) \leq z\} \\
&= \mu X.f_1(X) \sqcup \mu X.f_2(X)
\end{aligned}$$

(d) Is dual to the proof of c above, but using lemma 2.2d instead.

□

In case we need the least or greatest fixpoint of multiple variables at once, we can transform it into a nested fixpoint expression using *Bekič's theorem* [BBH+84]:

Theorem 2.2. Let $\langle Z_1, \leq\rangle$ and $\langle Z_2, \leq\rangle$ be complete lattices and let $f : Z_1 \times Z_2 \to Z_1$ and $g : Z_1 \times Z_2 \to Z_2$ be monotone functions. Then

$$\sigma(X, Y).(f(X, Y), g(X, Y)) = (x, y)$$

where

$$\begin{aligned}
x &= \sigma X.f(X, \sigma Y.g(X, Y)) \\
y &= \sigma Y.g(x, Y)
\end{aligned}$$

In [AN01] this is also known as the *Gauss elimination principle*. The authors also give a more general version for an arbitrary (finite) number of variables.

Theorem 2.3. Let $n \in \mathbb{N}^+$ be some natural number. Let $\langle Z_1, \leq\rangle, \ldots \langle Z_n, \leq\rangle$ be complete lattices and let $f_i : Z_1 \times \cdots \times Z_n \to Z_i$ be monotone functions for $1 \leq i \leq n$. Then

$$\sigma(X_1, \ldots X_n).(f_1(X_1, \ldots X_n), \ldots f_n(X_1, \ldots X_n)) = (x_1, \ldots x_n)$$

where $x_1, \ldots x_n$ can be computed by first computing

$$g_1(X_2, \ldots X_n) = \sigma X_1.f_1(X_1, \ldots X_n)$$

and then recursively computing

$$\sigma(X_2, \ldots X_n).(f_2(g_1(X_2, \ldots X_n), X_2, \ldots X_n), \ldots f_n(g_1(X_2, \ldots X_n), X_2, \ldots X_n))$$

Note that the above computations will result in $x_1, \ldots x_n$ to be a tuple of huge deeply nested fixpoint formulas.

2.2.3 The lattice over [0,1]

The lattice that we will mainly focus on is the lattice $\langle [0,1], \leq \rangle$. Apart from the operators $\sqcup$ and $\sqcap$ on this lattice, we will also consider addition ($+$) and subtraction (-) on $\mathbb{R}$, product ($\cdot$) on $[0,1]$ and the coproduct ($\odot$), truncated cosum ($\ominus$), truncated sum ($\oplus$) and weighted sum ($+_{\boldsymbol{\lambda}}$) where $\lambda \in [0,1]$ on $[0,1]$ as defined below:

Definition 2.14. Let $x, y, \lambda \in [0,1]$, then the operators $\odot$, $\ominus$, $\oplus$ and $+_{\boldsymbol{\lambda}}$ are defined as follows:

$$\begin{aligned} x \odot y &= x + y - x \cdot y \\ x \ominus y &= 0 \sqcup (x + y - 1) \\ x \oplus y &= 1 \sqcap (x + y) \\ x +_{\boldsymbol{\lambda}} y &= \lambda \cdot x + (1 - \lambda) \cdot y \end{aligned}$$

Note that the operators $\sqcup$ and $\sqcap$ are equivalent to the operators max and min respectively in this lattice. Since these operators can be distributed over each other, the lattice $\langle [0,1], \leq \rangle$ is distributive. The following distributivities hold as well:

Lemma 2.4. For any $x, y, z \in [0,1]$, we have that

$$\begin{aligned} x \cdot (y \sqcap z) &= (x \cdot y) \sqcap (x \cdot z) & x \cdot (y \sqcup z) &= (x \cdot y) \sqcup (x \cdot z) \\ x \odot (y \sqcap z) &= (x \odot y) \sqcap (x \odot z) & x \odot (y \sqcup z) &= (x \odot y) \sqcup (x \odot z) \\ x \ominus (y \sqcap z) &= (x \ominus y) \sqcap (x \ominus z) & x \ominus (y \sqcup z) &= (x \ominus y) \sqcup (x \ominus z) \\ x \oplus (y \sqcap z) &= (x \oplus y) \sqcap (x \oplus z) & x \oplus (y \sqcup z) &= (x \oplus y) \sqcup (x \oplus z) \\ x +_{\boldsymbol{\lambda}} (y \sqcap z) &= (x +_{\boldsymbol{\lambda}} y) \sqcap (x +_{\boldsymbol{\lambda}} z) & x +_{\boldsymbol{\lambda}} (y \sqcup z) &= (x +_{\boldsymbol{\lambda}} y) \sqcup (x +_{\boldsymbol{\lambda}} z) \end{aligned}$$

Proof. The above equations follow simply from the facts that the lattice $[0,1]$ is distributive and that product, addition and subtraction are distributive over minimum ($\sqcap$) and maximum ($\sqcup$). □

All operators mentioned above except subtraction are monotone. The operator pairs $(\sqcup, \sqcap)$, $(\cdot, \odot)$ and $(\ominus, \oplus)$ are each other's de Morgan duals where the negation of some $x \in [0,1]$ equals $1 - x$. The operator $+_{\boldsymbol{\lambda}}$ is self dual: $1 - (x +_{\boldsymbol{\lambda}} y) = (1 - x) +_{\boldsymbol{\lambda}} (1 - y)$.

We can turn the inequalities of lemmas 2.3c and 2.3d, namely that

$$\begin{aligned} \mu X.(f_1(X) \sqcup f_2(X)) &\geq \mu X.f_1(X) \sqcup \mu X.f_2(X) \\ \nu X.(f_1(X) \sqcap f_2(X)) &\leq \nu X.f_1(X) \sqcap \nu X.f_2(X) \end{aligned}$$

for monotone functions f_1 and f_2 over a complete lattice, into equalities when in the context of the lattice $[0,1]$ if we put further restrictions on f_1 and f_2, namely that the prefixpoints (and postfixpoints) of f_1 and f_2 are intervals that overlap each other. Note that in the lemma below the assumption that we use the lattice over $[0,1]$ is not used. The lemma also holds for lattices over real intervals other than $[0,1]$, but we will only need it for this interval.

Lemma 2.5. Let $f_1, f_2 : [0,1] \to [0,1]$ be two monotone functions such that $\{z \in [0,1] \mid f_i(z) \leq z\} = [x_i, x_i']$ and $\{z \in [0,1] \mid f_i(z) \geq z\} = [y_i, y_i']$ for some $x_i, x_i', y_i, y_i' \in [0,1]$ for $i \in 1,2$ such that $[x_1, x_1'] \cap [x_2, x_2'] \neq \emptyset$ and $[y_1, y_1'] \cap [y_2, y_2'] \neq \emptyset$. Also, let $X \in [0,1]$ be a variable. Then the following two equations hold:

(a) $\mu X.(f_1(X) \sqcup f_2(X)) = \mu X.f_1(X) \sqcup \mu X.f_2(X)$

(b) $\nu X.(f_1(X) \sqcap f_2(X)) = \nu X.f_1(X) \sqcap \nu X.f_2(X)$

Proof. We will prove a, b holds by duality.
We will first prove that $\bigsqcap([x_1, x_1'] \cap [x_2, x_2']) = \bigsqcap[x_1, x_1'] \sqcup \bigsqcap[x_2, x_2']$. Since $[x_1, x_1'] \cap [x_2, x_2'] \neq \emptyset$ we need to distinguishing four cases

- In case $x_1 \leq x_2 \leq x_1' \leq x_2'$ we can derive $\bigsqcap([x_1, x_1'] \cap [x_2, x_2']) = \bigsqcap[x_2, x_1'] = x_2 = x_1 \sqcup x_2 = \bigsqcap[x_1, x_1'] \sqcup \bigsqcap[x_2, x_2']$
- In case $x_2 \leq x_1 \leq x_2' \leq x_1'$ we can derive $\bigsqcap([x_1, x_1'] \cap [x_2, x_2']) = \bigsqcap[x_1, x_2'] = x_1 = x_1 \sqcup x_2 = \bigsqcap[x_1, x_1'] \sqcup \bigsqcap[x_2, x_2']$
- In case $x_1 \leq x_2 \leq x_2' \leq x_1'$ we can derive $\bigsqcap([x_1, x_1'] \cap [x_2, x_2']) = \bigsqcap[x_2, x_2'] = x_2 = x_1 \sqcup x_2 = \bigsqcap[x_1, x_1'] \sqcup \bigsqcap[x_2, x_2']$
- In case $x_2 \leq x_1 \leq x_1' \leq x_2'$ we can derive $\bigsqcap([x_1, x_1'] \cap [x_2, x_2']) = \bigsqcap[x_1, x_1'] = x_1 = x_1 \sqcup x_2 = \bigsqcap[x_1, x_1'] \sqcup \bigsqcap[x_2, x_2']$

With this information we can conclude that

$$\begin{aligned}
\mu X.(f_1(X) \sqcup f_2(X)) &= \bigsqcap\{z \in [0,1] \mid f_1(z) \sqcup f_2(z) \leq z\} \\
&= \bigsqcap\{z \in [0,1] \mid f_1(z) \leq z \wedge f_2(z) \leq z\} \\
&= \bigsqcap(\{z \in [0,1] \mid f_1(z) \leq z\} \cap \{z \in [0,1] \mid f_2(z) \leq z\}) \\
&= \bigsqcap\{z \in [0,1] \mid f_1(z) \leq z\} \sqcup \bigsqcap\{z \in [0,1] \mid f_2(z) \leq z\} \\
&= \mu X.f_1(X) \sqcup \mu X.f_2(X)
\end{aligned}$$

□

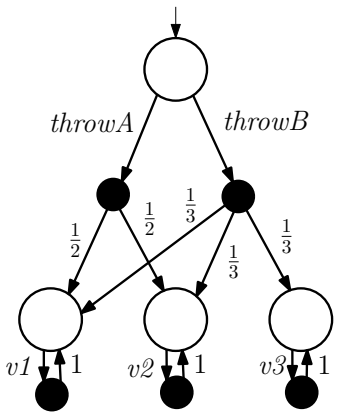
3 Transition systems

A transition system consists of states and transitions between states and it can be used to model the behaviour of a system. A state denotes a state of the system and a transition denotes a possible change of the system that will cause it to enter a (possibly) different state. It depends on the type of transition system what requirements are made on these states and transitions.

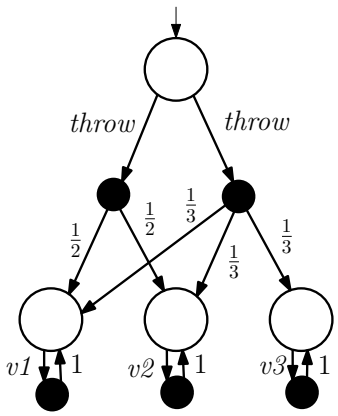
The type of transition system that we will focus on is the Probabilistic Labeled Transition System, or PLTS for short. This type of transition system has been introduced by Segala in his PhD thesis [Seg95].

Definition 3.1. A Probabilistic Labeled Transition System (PLTS) is a tuple $\langle S, s_0, A, T \rangle$ where S is a set of states, $s_0 \in S$ is the initial state, A is a set of actions and $T \subseteq S \times A \times (S \to [0,1])$ is the set of transitions, where $(S \to [0,1])$ is a valid probability distribution.

A PLTS has two types of choice: non-deterministic choice and probabilistic choice. A non-deterministic choice is made in a state and a probabilistic choice is made by the distribution in a transition. Note that two transitions in a non-deterministic choice can have two different actions or the same action. See figures 3a and 3b for an example PLTSs. The states in S are drawn as black circles and the transitions are drawn using arrows and black dots for the probability distribution. The example PLTS models either throwing a 2-sided die A or a 3-sided die B.
We assume that all probabilistic choices made according to the distributions in the model are solely dependent on the state the model is in. This implies that all such choices are independent. We will only consider PLTSs that have at most countably many infinite states and transitions.



(a) A PLTS where the choice between die A or B is deterministic.



(b) A PLTS where the choice between die A or B is non-deterministic.

Figure 3: Two example PLTSs that model throwing either 2-sided die A or 3-sided die B. The actions *vi* denote the resulting value i. The initial state is marked with a small incoming arrow.

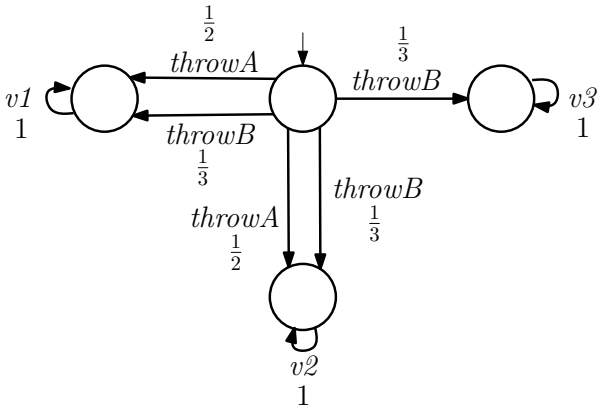


Figure 4: A PTS modeling the same behaviour as the PLTS shown in figure 3a.

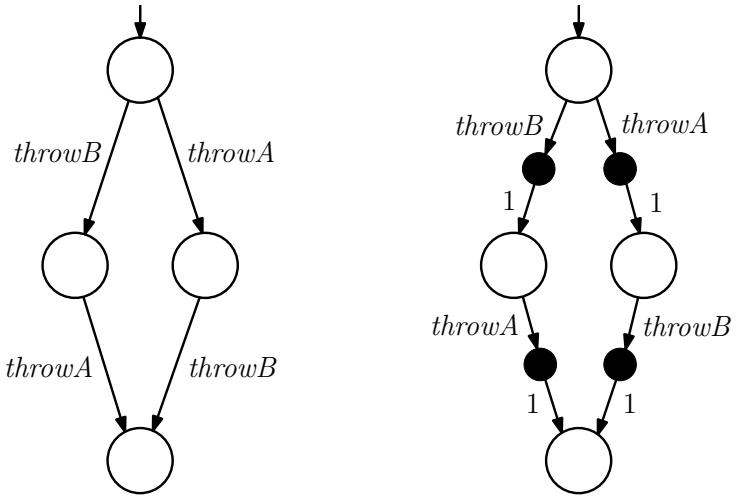


Figure 5: An LTS (left) and a PLTS (right) that model the same behaviour.

The PLTS is a generalization of other well-known transition systems used to model system behaviour. When we restrict a PLTS to not have any same-action non-determinism, the model is also known as a Probabilistic Transition System (PTS) (but in literature it is sometimes named as the PLTS) [NCI99]. When we restrict a PLTS such that all probability distributions are deterministic, effectively removing all probabilistic behaviour, we have created a Labeled Transition System (LTS) which is often used for qualitative model checking.

Definition 3.2. A Labeled Transition System (LTS) is a tuple $\langle S, s_0, A, T \rangle$ where S is a set of states, $s_0 \in S$ is the initial state, A is a set of actions and $T \subseteq S \times A \times S$ is the set of transitions.

On the other hand, if we would restrict a PLTS to only have one outgoing transition per state, effectively removing all non-deterministic behaviour, we have created a (Labeled) Markov Chain or Markov Process [ASB95]. See figures 4 and 5 for some examples.

4 Non-probabilistic model checking

Before we dive into model checking on probabilistic systems, we will first show some methods often used in qualitative model checking on non-probabilistic models.

4.1 Modal μ-calculus ($L\mu$)

The logic that is most often used in this setting is the modal μ-calculus by Kozen [Koz83], which we will refer to as $L\mu$. With this logic one can verify qualitative properties on LTSs. The reason that this logic is used that often is because it subsumes many other logics like CTL [CE81] that can be used in similar settings.

Definition 4.1. Let Var be a set of boolean variables. Then the syntax of an $L\mu$-formula ϕ is:

$$\phi ::= \mathit{false} \mid \mathit{true} \mid X \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \langle a \rangle \phi \mid [a]\phi \mid \mu X.\phi \mid \nu X.\phi$$

where $X \in Var$ is a variable and a is an action.

To preserve semantic monotonicity, which is required for the fixpoint operators, each variable may only be preceded by an even number of negations.

Definition 4.2. Let $\langle S, s_0, A, T \rangle$ be an LTS and let $[\![\phi]\!]_e \in 2^S$ be the denotational semantics of the μ-calculus, where $e : Var \to 2^S$ assigns values to variables. Then $[\![\phi]\!]_e$ is defined as follows:

$$\begin{aligned}
[\![\mathit{false}]\!]_e &= \emptyset \\
[\![\mathit{true}]\!]_e &= S \\
[\![X]\!]_e &= e(X) \\
[\![\neg\phi]\!]_e &= S \setminus [\![\phi]\!]_e \\
[\![\phi_1 \wedge \phi_2]\!]_e &= [\![\phi_1]\!]_e \cap [\![\phi_2]\!]_e \\
[\![\phi_1 \vee \phi_2]\!]_e &= [\![\phi_1]\!]_e \cup [\![\phi_2]\!]_e \\
[\![\langle a \rangle \phi]\!]_e &= \{s \in S \mid \exists_{s' \in S} : (s, a, s') \in T \wedge s' \in [\![\phi]\!]_e\} \\
[\![[a]\phi]\!]_e &= \{s \in S \mid \forall_{s' \in S} : (s, a, s') \in T \Rightarrow s' \in [\![\phi]\!]_e\} \\
[\![\mu X.\phi]\!]_e &= \bigcap \{S' \subseteq S \mid [\![\phi]\!]_{e[X:=S']} \subseteq S'\} \\
[\![\nu X.\phi]\!]_e &= \bigcup \{S' \subseteq S \mid S' \subseteq [\![\phi]\!]_{e[X:=S']}\}
\end{aligned}$$

We call a variable X bound in a formula ϕ if X is preceded by a fixpoint operator $\sigma X.\phi'$ in ϕ, else it is free.
More intuitively, one can see $[\![\phi]\!]_e$ as the set of states where the formula ϕ holds. The formulas *false* and *true* are always false and true respectively, independent of the state. The formula $\neg\phi$ holds in a state if ϕ does not hold in this state. The formula $\phi_1 \wedge \phi_2$ holds in a state if both ϕ_1 and ϕ_2 hold in that state and dually $\phi_1 \vee \phi_2$ holds in a state if at least one of ϕ_1 and ϕ_2 holds in that state. The formula $\langle a \rangle \phi$ holds in a state if ϕ holds in at least one of its successor states reached via an a-step and dually, $[a]\phi$ holds in a state if ϕ holds in all of its successor states reached via an a-step.

4.2 Fixpoint Equation Systems

As mentioned before, one can find the least and greatest fixpoints of a function f using an iterative approach. However, when applying this to functions with nested fixpoints, as may occur in $L\mu$, this iterative method becomes quite tedious. Therefore, in 1997, Angelika Mader defined the Fixpoint Equation System (FES) [Mad97], which is a representation of a fixpoint formula by means of equations equipped with a least or greatest fixpoint symbol. In this section we will discuss some of the work from Mader's PhD thesis [Mad97] that will be useful later on. The proofs of the lemmas in this section can be found in that thesis.

Definition 4.3. Let $\langle Z, \leq \rangle$ be a complete lattice, let Var be a set of variables in Z and let $Op(f_1, ...f_n)$ be some n-ary operator on Z. Then the syntax of a FES over lattice $\langle Z, \leq \rangle$ is defined as

$$\mathcal{E} ::= \epsilon \mid (\mu X = f)\ \mathcal{E} \mid (\nu X = f)\ \mathcal{E}$$

where ϵ denotes the empty FES, $X \in Var$ is a variable and $f : Z^{|Var|} \to Z$ is a function with the syntax

$$f ::= z \mid X \mid f \sqcap f \mid f \sqcup f \mid Op(f_1, ...f_n)$$

where $z \in Z$ and $X \in Var$.

Definition 4.4. Let $\langle Z, \leq \rangle$ be a complete lattice and let $Op(g_1, ...g_n)$ be some n-ary operator on Z. Let g be some monotone formula on Z with the syntax

$$g ::= z \mid X \mid g \sqcap g \mid g \sqcup g \mid Op(g_1, ...g_n) \mid \mu X.g \mid \nu X.g$$

where $z \in Z$ and $X \in Var$. Then the FES $\mathbf{E}(g)$ of g is defined as:

$$\begin{aligned}
\mathbf{E}(z) &= \epsilon \\
\mathbf{E}(X) &= \epsilon \\
\mathbf{E}(g_1 \sqcap g_2) &= \mathbf{E}(g_1)\ \mathbf{E}(g_2) \\
\mathbf{E}(g_1 \sqcup g_2) &= \mathbf{E}(g_1)\ \mathbf{E}(g_2) \\
\mathbf{E}(Op(g_1, ...g_n)) &= \mathbf{E}(g_1)\ \ldots\ \mathbf{E}(g_n) \\
\mathbf{E}(\sigma X.g) &= (\sigma X = \mathbf{E}'(g))\ \mathbf{E}(g)
\end{aligned}$$

$$\begin{aligned}
\mathbf{E}'(z) &= z \\
\mathbf{E}'(X) &= X \\
\mathbf{E}'(g_1 \sqcap g_2) &= \mathbf{E}'(g_1) \sqcap \mathbf{E}'(g_2) \\
\mathbf{E}'(g_1 \sqcup g_2) &= \mathbf{E}'(g_1) \sqcup \mathbf{E}'(g_2) \\
\mathbf{E}'(Op(g_1, ...g_n)) &= Op(\mathbf{E}'(g_1), ...\mathbf{E}'(g_n)) \\
\mathbf{E}'(\sigma X.g) &= X
\end{aligned}$$

where $z \in Z$ and $X \in Var$.

The resulting FES is a list of equations of the form $(\sigma X = f)$ where $f : Z^{|Var|} \to Z$ for every variable X bound by a fixpoint operator in the original fixpoint formula. Similarly as for fixpoint formulas, a variable X is bound in a FES $\mathcal{E}$ if

$\mathcal{E}$ contains an equation of the form $(\sigma X = f)$, else it is free. If in an equation $(\sigma X = f)$ f does not contain any free variables, we call this equation closed. We call the FES closed if all equations in the FES are closed.

A FES can be solved by seeing each equation as its own fixpoint expression.

Definition 4.5. Let $\mathcal{E}$ be a FES over the complete lattice $\langle Z, \leq \rangle$ and let $e : Var \to Z$ be an environment, which assigns a value to each variable in the FES. Then $[\mathcal{E}]e$ is the *solution of* $\mathcal{E}$ *with respect to* e defined as:

$$\begin{aligned} [\epsilon]e &= e \\ [(\mu X = f)\ \mathcal{E}]e &= [\mathcal{E}]e[X := \mu X.f([\mathcal{E}]e)] \\ [(\nu X = f)\ \mathcal{E}]e &= [\mathcal{E}]e[X := \nu X.f([\mathcal{E}]e)] \end{aligned}$$

where

$$\begin{aligned} \mu X.f([\mathcal{E}]e) &= \bigsqcap \{z \in Z \mid f([\mathcal{E}]e[X := z]) \leq z\} \\ \nu X.f([\mathcal{E}]e) &= \bigsqcup \{z \in Z \mid f([\mathcal{E}]e[X := z]) \geq z\} \end{aligned}$$

The "with respect to e" can be dropped when the FES $\mathcal{E}$ is closed, since then the solution is unique for all (bound) variables.

Most importantly, Mader shows that this solution of a FES encodes the solution of the original fixpoint expression.

Lemma 4.1. Let $\langle Z, \leq \rangle$ be a complete lattice, $f : Z \to Z$ a monotone function and e an arbitrary environment. Then

$$[\![\sigma X.f]\!]_e = ([\mathbf{E}(\sigma X.f)]e)(X)$$

where $[\![\sigma X.f]\!]_e$ is the value of $\sigma X.f$ with respect to environment e.

Mader also describes an alternate definition of the solution of a FES.

Definition 4.6. Let $\mathcal{E}$ be a FES. Then we define a *lexicographical ordering on environments with respect to* $\mathcal{E}$, denoted as $\leq_{\mathcal{E}}$, for environments e_1 and e_2 such that $e_1 \leq_{\mathcal{E}} e_2$ iff

- if $\mathcal{E} = \epsilon$, then $e_1 = e_2$.
- if $\mathcal{E} = (\mu X = f)\ \mathcal{E}'$, then $e_1(X) < e_2(X)$ or both $e_1(X) = e_2(X)$ and $e_1 \leq_{\mathcal{E}'} e_2$.
- if $\mathcal{E} = (\nu X = f)\ \mathcal{E}'$, then $e_1(X) > e_2(X)$ or both $e_1(X) = e_2(X)$ and $e_1 \leq_{\mathcal{E}'} e_2$.

Lemma 4.2. Let $\mathcal{E}$ be a FES and e an environment. If $\mathcal{E} = \epsilon$, then $[\mathcal{E}]e = e$. If $\mathcal{E} = (\sigma X = f)\ \mathcal{E}'$ then $[\mathcal{E}]e$ is the lexicographically least (with respect to $\mathcal{E}$) environment e' such that:

- $f(e') = e'(X)$ and
- $[\mathcal{E}']e[X := e'(X)] = e'$

From this lemma we can derive the following:

Lemma 4.3. Let $\mathcal{E} = (\sigma X_1 = f_1) \ ... \ (\sigma X_n = f_n)$ be a FES and let $\mathcal{E}^i = (\sigma X_i = f_i) \ ... \ (\sigma X_n = f_n)$ for $1 \leq i \leq n$. Then $[\mathcal{E}]e = e'$ implies $[\mathcal{E}^i]e' = e'$ for $1 \leq i \leq n$.

Another useful definition is the equivalence on FESs.

Definition 4.7. Two FESs $\mathcal{E}_1$ and $\mathcal{E}_2$ are *equivalent*, denoted by $\mathcal{E}_1 \sim \mathcal{E}_2$, iff for all environments e it holds that $[\mathcal{E}_1]e = [\mathcal{E}_2]e$.

Lemma 4.4. Let $\mathcal{E}$, $\mathcal{E}_1$ and $\mathcal{E}_2$ be FESs. Then $\mathcal{E}_1 \sim \mathcal{E}_2$ implies $\mathcal{E}\mathcal{E}_1 \sim \mathcal{E}\mathcal{E}_2$.

Note that the ordering of equations in a FES is important: the less deeply nested a fixpoint expression is, the higher (more to the left) its corresponding equation occurs. However, it is allowed to swap consecutive equations as long as they have the same fixpoint sign.

Lemma 4.5. If

$$e_1 = [\mathcal{E}_1 \ (\sigma X_1 = f_1) \ (\sigma X_2 = f_2) \ \mathcal{E}_2]e$$
$$e_2 = [\mathcal{E}_1 \ (\sigma X_2 = f_2) \ (\sigma X_1 = f_1) \ \mathcal{E}_2]e$$

then $e_1 = e_2$.

4.2.1 Boolean Equation Systems

In practice the FES is used in model checking in the form of a Boolean Equation System (BES), which is an instance of a FES over the lattice $\langle \mathbb{B}, \leq \rangle$ where $\mathit{false} \leq \mathit{true}$. A BES can be used to solve the model checking problem of $L\mu$ on an LTS. See below for the translation of an $L\mu$ model checking problem to a BES.

Definition 4.8. Let $\mathcal{M} = \langle S, s_0, A, T \rangle$ be an LTS and $\sigma X.\phi$ an $L\mu$-formula. Then the BES $\mathbf{E}_B(\sigma X.\phi)$ of $\sigma X.\phi$ on $\mathcal{M}$ is defined as follows:

$$\begin{aligned}
\mathbf{E}_B(\mathit{false}) &= \epsilon \\
\mathbf{E}_B(\mathit{true}) &= \epsilon \\
\mathbf{E}_B(X) &= \epsilon \\
\mathbf{E}_B(\phi_1 \wedge \phi_2) &= \mathbf{E}_B(\phi_1)\mathbf{E}_B(\phi_2) \\
\mathbf{E}_B(\phi_1 \vee \phi_2) &= \mathbf{E}_B(\phi_1)\mathbf{E}_B(\phi_2) \\
\mathbf{E}_B(\langle a \rangle \phi) &= \mathbf{E}_B(\phi) \\
\mathbf{E}_B([a]\phi) &= \mathbf{E}_B(\phi) \\
\mathbf{E}_B(\sigma X.\phi) &= ((\sigma X_s = \mathbf{E}'_B(s, \phi)) \ \mathit{for\ each}\ s \in S) \ \mathbf{E}_B(\phi)
\end{aligned}$$

$$
\begin{aligned}
\mathbf{E}'_B(s, \mathit{false}) &= \mathit{false} \\
\mathbf{E}'_B(s, \mathit{true}) &= \mathit{true} \\
\mathbf{E}'_B(s, X) &= X_s \\
\mathbf{E}'_B(s, \phi_1 \wedge \phi_2) &= \mathbf{E}'_B(s, \phi_1) \wedge \mathbf{E}'_B(s, \phi_2) \\
\mathbf{E}'_B(s, \phi_1 \vee \phi_2) &= \mathbf{E}'_B(s, \phi_1) \vee \mathbf{E}'_B(s, \phi_2) \\
\mathbf{E}'_B(s, \langle a \rangle \phi) &= \bigvee_{(s,a,s') \in T} \mathbf{E}'_B(s', \phi) \\
\mathbf{E}'_B(s, [a] \phi) &= \bigwedge_{(s,a,s') \in T} \mathbf{E}'_B(s', \phi) \\
\mathbf{E}'_B(s, \sigma X.\phi) &= X_s
\end{aligned}
$$

Note that this translation $\mathbf{E}_B$ is slightly different than the translation $\mathbf{E}$ as in definition 4.4. The translation $\mathbf{E}$ creates a FES over the same lattice as the original formula, whereas translation $\mathbf{E}_B$ translates a formula over the lattice $\langle 2^S, \subseteq \rangle$ to a the BES over the lattice $\langle \mathbb{B}, \leq \rangle$. This is because the fixpoint operator is translated to $|S|$ equations instead of only one. Due to this, there are $|S|$ times more variables. Let Var be the set of variables in the $L\mu$ model checking problem, then the set of variables Var' in the resulting BES equals $\{X_s \mid X \in Var, s \in S\}$.
Because of the change in lattice it is necessary to prove correctness of the solution again, which is done in [Mad97].

Theorem 4.1. Let $\langle S, s_0, A, T \rangle$ be an LTS, $\sigma X.\phi$ an $L\mu$-formula, Var a set of boolean variables, $\sigma X.\phi$ an $L\mu$-formula and $e : \mathit{Var} \to 2^S$ and $e' : \mathit{Var}' \to \mathbb{B}$ environments such that $s \in e(Y)$ iff $e'(Y_s) = \mathit{true}$ for all $Y \in \mathit{Var}$ that are free in $\sigma X.\phi$. Then

$$s \in [\![\sigma X.\phi]\!]_e \Leftrightarrow ([\mathbf{E}(\sigma X.\phi)]e')(X_s)$$

As Mader shows in [Mad97], a (closed) BES $\mathcal{E} = (\sigma X_1 = \phi_1) \ldots (\sigma X_n = \phi_n)$ can be solved by means of an algorithm similar to Gauss-elimination, see algorithm 1. The algorithm consists of two parts: the elimination step (lines 2 to 6) and the substitution step (lines 7 to 9).

Algorithm 1 *GaussElimination*($\mathcal{E}$)

```
1:  for i := n downto 0 do
2:    if X_i is bound by a least fixpoint then
3:      φ_i = φ_i[X_i := false]
4:    else
5:      φ_i = φ_i[X_i := true]
6:    end if
7:    for j := 0 to i − 1 do
8:      φ_j = φ_j[X_i := φ_i]
9:    end for
10: end for
```

The operations in the algorithm can be carried out in polynomial time, but due to the substitution step the formulas can grow exponentially, requiring an exponential number of substitutions. Therefore, the *GaussElimination* algorithm

has an exponential worst case running time.
However, one can improve the running time somewhat by applying simplifications on the altered formulas after elimination and substitution. Some examples of such simplifications are

- absorbing element: $\phi \wedge \mathit{false} = \mathit{false}$ and $\phi \vee \mathit{true} = \mathit{true}$,
- identity: $\phi \wedge \mathit{true} = \phi$ and $\phi \vee \mathit{false} = \phi$,
- idempotence: $\phi \wedge \phi = \phi$ and $\phi \vee \phi = \phi$,
- absorption: $\phi_1 \wedge (\phi_1 \vee \phi_2) = \phi_1$ and $\phi_1 \vee (\phi_1 \wedge \phi_2) = \phi_1$.

5 Previous work

In the past there has been quite a lot of work done on verification on probabilistic systems. In this section we will summarize most of this work. We will first shortly look into approaches that precede the use of a quantitative μ-calculus. Afterwards we will focus on the work done in the field of using a quantitative μ-calculus. Lastly, we will look shortly into existing tool-sets that can do model checking on probabilistic models.

5.1 Approaches before quantitative μ-calculus

Temporal logic

One of the first attempts of doing model checking using temporal logic on probabilistic systems is done in [Var85]. Here it is shown how to prove a property ϕ expressed in temporal logic on a labeled Markov chain Π, by first creating the Büchi automaton B_ϕ of ϕ. In this paper the notions probabilistic universality and emptiness are defined, which denote the properties whether the probability that there is a path in Π that is accepting in B_ϕ equals 1 and 0 respectively. It is shown that this problem can be reduced to checking non-probabilistic universality and emptiness.
In 1995 the temporal logic pCTL* and its sublogic pCTL were devised [ASB95], which extend the state formula of CTL* (and CTL) with probabilistic threshold operators. These operators are $P_{<p}\psi$ and $P_{>p}\psi$, where p is a real number in the interval $[0, 1]$. The meaning of these operators is "the probability to take a path from s where ψ holds is less/greater than p" when evaluated in a state s.

Qualitative μ-calculi

In [LS89] the first attempt is made to use a μ-calculus to apply to probabilistic systems. Their μ-calculus for PTSs however does not contain negation, fixpoint operators or the box operator when compared to $L\mu$. The diamond operator has changed to $\langle a \rangle_p \phi$, which is true in a state iff there exists an a-transition from this state with probability at least p to a state where ϕ holds. To still be able to test for deadlocks, the Δ_a operator is added, which is true in a state iff there is no a-action possible in this state. Also, in this paper the notion of probabilistic bisimilation is defined.
A qualitative adaptation of $L\mu$ for probabilistic systems including fixed point operators is defined in [NCI99]. Similar to pCTL, their logic GPL distinguishes between state (ϕ) and path (ψ) formulas and adds probabilistic threshold operators $Æ_{>p}\psi$ and $Æ_{\geq p}\psi$, which are true iff the measure of ψ is greater than (or equal to) p. The authors also show that GPL is a generalization of pCTL.

Theorem proving

The first attempt to get a quantitative answer in the verification on probabilistic models is done in [MMS96, MM97]. They interpret temporal logic operators as programming statements and use the weakest precondition approach to calculate the expected value of a variable of a program. Probability is added to these programs by the $s \;{}_p\!\oplus\, t$ operator, meaning that statement s will be executed with probability p or else t (with probability $1 - p$).

5.2 Quantitative μ-calculi

The first adaptation of $L\mu$ that returns a quantitative answer when applied to a probabilistic system is the μ-calculus defined by Huth and Kwiatkowska in [HK97]. The model used in this paper is the PTS. The syntax of this μ-calculus is the same as that of $L\mu$; it is the semantics where the differences reside. Instead of returning a function from states to booleans (or a set of states for which the formula is true), these semantics return a function from states to probabilities. The semantics of a formula ϕ given some environment e is again given by $[\![\phi]\!]_e$. Then a formula ϕ checked in a state s ($[\![\phi]\!]_e(s)$) gives the probability that ϕ will happen in state s.
The booleans *true* and *false* are mapped to 1 and 0 respectively. The negation $\neg\phi$ of a formula ϕ corresponds to $1 - [\![\phi]\!]_e$. For the formulas $\phi \wedge \psi$ and $\phi \vee \psi$ the authors propose multiple possible semantics. This has to do with a situation we sketched earlier in section 2.1. We know the probabilities $[\![\phi]\!]_e(s)$ and $[\![\psi]\!]_e(s)$ for some state s and environment e, but we also need the dependence between these two to be able to know the exact values of $[\![\phi \wedge \psi]\!]_e(s)$ and $[\![\phi \vee \psi]\!]_e(s)$. Since we do not know this dependence, a range of values is possible. For $[\![\phi \wedge \psi]\!]_e(s)$ the authors propose three options:

- $\min([\![\phi]\!]_e(s), [\![\psi]\!]_e(s))$, which is the upper bound,
- $[\![\phi]\!]_e(s) \cdot [\![\psi]\!]_e(s)$, which is the exact probability if ϕ and ψ are independent,
- $\max(0, [\![\phi]\!]_e(s) + [\![\psi]\!]_e(s) - 1)$, which is the lower bound.

Similarly, there are three options for $[\![\phi \vee \psi]\!]_e(s)$ which are the duals of the options above.
The diamond operator $\langle a \rangle \phi$ acts as a weighted average of all transitions a where the weights are the probabilities to go to a state with an a-action. The box operator $[a]\phi$ is simply the dual of the diamond operator. Note that since same-action non-determinism is not possible in a PTS the diamond and box operators $\langle a \rangle \phi$ and $[a]\phi$ are actually equal when checked on a state with at least one outgoing a-transition.
The fixpoint operators also exist in the logic of Huth and Kwiatkowska, but over the lattice $\langle [0,1]^S, \dot{\leq} \rangle$ of functions from states to probabilities.
Huth and Kwiatkowska also give an algorithm to solve formulas, which directly uses the semantics and solves fixpoint formulas $\sigma X.\phi$ by computing $[\![\phi]\!] = X$, but it does not allow nested fixpoint operators.

One year later, Baier and Clarke take the quantitative μ-calculus to a more general level in [BC98]. By allowing any real number and any arithmetical operator their algebraic μ-calculus allows much more quantitative properties than those based on probabilities. Also, the underlying model does not need to be probabilistic. For example, the authors also mention applications such as solving graph problems and doing operations on matrices. They also give a way of computing the resulting values of formulas expressed in this μ-calculus, which uses Multi-Terminal Binary Decision Diagrams (MTBDDs).

De Alfaro [dAM01, dA03], inspired by the μ-calculus of Huth and Kwiatkowska, creates a quantitative game μ-calculus applied in the context of concurrent game structures. In his μ-calculus this is reflected by having the operators $Ppre_i(\phi)$

for $i \in \{1, 2\}$ instead of the diamond and box operator. These new operators simulate player i trying to maximize its own value while trying to minimize the other's.

Lluch-Lafuante and Montanari take a more theoretical approach by defining a μ-calculus on constraint semirings [LM05]. As Baier and Clarke, the authors aim for a more general setting than probabilistic systems. They focus mainly on Quality of Service properties, such as network availability, bandwidth, performance and access rights. They also give a variation on CTL based on constraint semirings and for both semiring logics the authors give algorithms to solve them.

After their work on quantitative verification on probabilistic systems using the weakest precondition, McIver and Morgan also come up with a quantitative adaptation of the μ-calculus [MM07, MM06], independently of the work of Huth and Kwiatkowska. The logic of McIver and Morgan, called $pL\mu$, is very similar to the logic of Huth and Kwiatkowska. One of the differences is that for the $\vee$ and $\wedge$ the options $\sqcup$ (maximum) and $\sqcap$ (minimum) are chosen respectively. Also McIver and Morgan use the more expressive PLTS as model instead, but then without action labels. The diamond and box operators are therefore changed to accept a set of possible transitions instead of an action. Their semantics are changed such that the diamond operator takes the maximum resulting value over all the given transitions and dually the box operator takes the minimum. In [MM06], the authors give a translation from their logic $pL\mu$ to a $2\frac{1}{2}$-player parity game.

The latest work is done by Mio. In his PhD thesis [Mio12] he extends the logic $pL\mu$ of McIver and Morgan to the quantitative μ-calculus $pL\mu^{\odot}_{\oplus}$, which adds the other options proposed by Huth and Kwiatkowska and an additional weighted sum operator. He also defines a new type of parity game that fits this logic. In the following sections of this paper, we will focus on understanding this logic and using it to solve quantitative model checking problems on PLTSs.

5.3 Practical application

In the theoretic work above, except for the work by Huth and Kwiatkowska [HK97] and Baier and Clarke [BC98], no algorithm or procedure is given to solve quantitative μ-calculus formulas (except for some specific examples). However, there are tool-sets available, namely PRISM [KNP02] and MRMC [KKZ05] that are able to qualitative model checking using pCTL for labeled discrete time Markov chains and CSL for labeled continuous time Markov chains. They also have extensions on both logics so that they can answer quantitative questions such as "what is the probability that ..." or "what is the expected number of transitions to get to a state where ...".
The PRISM tool-set also specializes in models that include time information. With this time information, one can check for instance the probability of something happening within a unit of time.

6 Syntax, semantics and intuitive meaning of $pL\mu^{\odot}_{\oplus}$-formulas

In this section we will look into the meaning of $pL\mu^{\odot}_{\oplus}$-formulas, both formally and intuitively. This will be done by relating the $pL\mu^{\odot}_{\oplus}$-formulas back to probability theory and by going through practical examples.
First we shortly look into the resulting value of a $pL\mu^{\odot}_{\oplus}$-formula. Afterwards the syntax of $pL\mu^{\odot}_{\oplus}$ is given, followed by both the semantical and intuitive meaning of each element of this syntax. Then we analyze the meaning of combinations of these elements (formulas) by focusing on subjects like dependence, fixpoint formulas and the difference between probabilistic and non-deterministic choice. This will be summarized by giving some guidelines on how to create meaningful formulas. Afterwards we will dive into additional subjects surrounding $pL\mu^{\odot}_{\oplus}$ as given in Mio's PhD thesis [Mio12], which are model checking qualitative properties using $pL\mu^{\odot}_{\oplus}$ and an alternative representation of a $pL\mu^{\odot}_{\oplus}$ model checking problem, namely the $2\frac{1}{2}$-player meta-parity game.

6.1 The resulting value

In $L\mu$ the resulting value of a formula ϕ is either true or false. When checking such a formula on a state s, it will return whether ϕ holds in state s or rather whether behaviour that adheres to ϕ will happen when starting in state s. In $pL\mu^{\odot}_{\oplus}$ however, the resulting value of a $pL\mu^{\odot}_{\oplus}$-formula ϕ lies within the interval $[0,1]$ of real numbers. When checking such a formula on a state s, it will return the expected probability that ϕ holds in state s or rather the (expected) probability that behaviour that adheres to ϕ will happen when starting from state s. More formally, let $[0,1]^S$ be the set of all mappings $S \to [0,1]$, which is a complete lattice under the pointwise order $\dot{\leq}$ as shown in [Mio12]. Let $\langle S, s_0, A, T\rangle$ be a PLTS and Var be a set of variables. Then we define $[\![\phi]\!]_e \in [0,1]^S$ to be the denotational semantics of some $pL\mu^{\odot}_{\oplus}$-formula ϕ, where $e : Var \to [0,1]^S$ is an environment that assigns values to variables.

Now we would like to relate the resulting value $[\![\phi]\!]_e(s)$ of a $pL\mu^{\odot}_{\oplus}$-formula checked on a state s in a PLTS to probability theory. As done in [Sto02] and [AW06], we can introduce a scheduler σ to remove the non-determinism. We can then define a probability space $\langle \Omega, \mathcal{F}, P\rangle$ where

- Ω is the set of maximal paths through a PLTS under the scheduler σ,
- $\mathcal{F}$ contains a cylinder set of maximal paths for each finite path in the PLTS,
- P is the product of all probabilistic choices made in the finite path that corresponds to the cylinder set.

This will give a well defined probability space for the model, but we would like to associate this to the semantics of a $pL\mu^{\odot}_{\oplus}$-formula ϕ. To do so, we want to find an event that corresponds to ϕ. However, since $\mathcal{F}$ contains sets of linear paths and $pL\mu^{\odot}_{\oplus}$ is a branching time logic, this is not possible.
So we could instead decide to keep the branching behaviour of the PLTS by defining a "branching path", which is a path that fully branches in case of

non-probabilistic choice (but still only branches once in case of probabilistic choice). However, using such (maximal) branching paths as outcomes it is not sure whether we can come up with a well defined probability space.
Even if we would be able to do so, this probability space would only capture the probabilistic behaviour of the model. As will be shown later on, it is also possible for a $pL\mu^{\odot}_{\oplus}$-formula to contain some probabilistic behaviour.

All in all it does not seem to be possible to make a relation between the semantics and $pL\mu^{\odot}_{\oplus}$ and probability theory using the concepts of outcomes and events although the resulting value $[\![\phi]\!]_e(s)$ of a $pL\mu^{\odot}_{\oplus}$-formula ϕ checked in a state s does share some properties with a probability function P, such as duality and dependence. Nevertheless, in the coming sections we will try to create intuition for the $pL\mu^{\odot}_{\oplus}$-formulas themselves, but first we will elaborate on the atoms and operators that make up $pL\mu^{\odot}_{\oplus}$.

6.2 Atoms and operators

The syntax of a $pL\mu^{\odot}_{\oplus}$-formula ϕ is given by

$$\phi ::= X \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \cdot \phi \mid \phi \odot \phi \mid \phi \ominus \phi \mid \phi \oplus \phi \mid \phi +_\lambda \phi \mid \langle a \rangle \phi \mid [a]\phi \mid \mu X.\phi \mid \nu X.\phi$$

where $X \in Var$ is a variable, $\lambda \in [0,1]$ a probability and $a \in A$ an action. We will however consider more atoms as mentioned in [Mio12], namely $\underline{0}$, $\underline{1}$ and $\underline{\lambda}$, which are defined as $\mu X.X$, $\nu X.X$ and $\underline{1} +_\lambda \underline{0}$ respectively. Also, we will sometimes refer to sublogics of $pL\mu^{\odot}_{\oplus}$:

- $pL\mu$, which is $pL\mu^{\odot}_{\oplus}$ without the operators $\cdot$, $\odot$, $\ominus$ and $\oplus$,
- $pL\mu^{\odot}$, which is $pL\mu^{\odot}_{\oplus}$ without the operators $\ominus$ and $\oplus$ and
- $pL\mu_{\oplus}$, which is $pL\mu^{\odot}_{\oplus}$ without the operators $\cdot$ and $\odot$.

We will go through all syntactic elements one by one, giving their formal definition and their intuitive meaning. Let $\langle S, s_0, A, T \rangle$ be a PLTS, let $s \in S$ be an arbitrary state in this PLTS and let $e : Var \to (S \to [0,1])$ be an arbitrary environment.

Atom $\underline{0}$

Semantically, the atom $\underline{0}$ is defined as

$$[\![\underline{0}]\!]_e(s) = 0$$

It is the probability value 0 independent of the state it is checked on. If a (sub)formula ϕ_0 ends with $\underline{0}$, the behaviour that is expressed in ϕ_0 by the operators that precede $\underline{0}$ is undesired: we aim for the probability that this behaviour does not happen.

Atom $\underline{1}$

Semantically, the atom $\underline{1}$ is defined as

$$[\![\underline{1}]\!]_e(s) = 1$$

It is the probability value 1 independent of the state it is checked on. If a (sub)formula ϕ_1 ends with $\underline{1}$, the behaviour expressed in ϕ_1 by the operators that precede $\underline{1}$ is desired: we aim for the probability that this behaviour happens.

Atom $\underline{\lambda}$

Semantically, for any λ in the interval $[0, 1]$ the atom $\underline{\lambda}$ is defined as

$$[\![\underline{\lambda}]\!]_e(s) = \lambda$$

It is the probability value λ independent of the state it is checked on. If a (sub)formula ϕ_λ ends with $\underline{\lambda}$, the behaviour that is expressed in ϕ_1 by the operators that precede $\underline{1}$ is desired to a certain degree λ.

Atom X

Semantically, the atom X, representing a variable, is defined as

$$[\![X]\!]_e(s) = e(X)(s)$$

It corresponds to the probability value as given by the environment. If the atom X is within the scope of a corresponding fixpoint operator ($\sigma X.$), we say that the variable is bound, else it is free. We will only consider $pL\mu^{\odot}_{\oplus}$-formulas that do not contain free variables, which are called closed formulas. We will also assume that each variable will only occur at most once in a $pL\mu^{\odot}_{\oplus}$-formula.

Operators $\wedge$, $\vee$, $\cdot$, $\odot$, $\ominus$ and $\oplus$

Semantically, the operators $\wedge$ (and), $\vee$ (or), $\cdot$ (product), $\odot$ (coproduct), $\ominus$ (truncated cosum) and $\oplus$ (truncated sum) are defined as

$$\begin{aligned}
[\![\phi_1 \wedge \phi_2]\!]_e(s) &= [\![\phi_1]\!]_e(s) \sqcap [\![\phi_2]\!]_e(s) \\
[\![\phi_1 \vee \phi_2]\!]_e(s) &= [\![\phi_1]\!]_e(s) \sqcup [\![\phi_2]\!]_e(s) \\
[\![\phi_1 \cdot \phi_2]\!]_e(s) &= [\![\phi_1]\!]_e(s) \cdot [\![\phi_2]\!]_e(s) \\
[\![\phi_1 \odot \phi_2]\!]_e(s) &= [\![\phi_1]\!]_e(s) \odot [\![\phi_2]\!]_e(s) \\
[\![\phi_1 \ominus \phi_2]\!]_e(s) &= [\![\phi_1]\!]_e(s) \ominus [\![\phi_2]\!]_e(s) \\
[\![\phi_1 \oplus \phi_2]\!]_e(s) &= [\![\phi_1]\!]_e(s) \oplus [\![\phi_2]\!]_e(s)
\end{aligned}$$

The formulas $\phi_1 \wedge \phi_2$, $\phi_1 \cdot \phi_2$ and $\phi_1 \ominus \phi_2$ correspond to the probability that both ϕ_1 and ϕ_2 happen and the formulas $\phi_1 \vee \phi_2$, $\phi_1 \odot \phi_2$ and $\phi_1 \oplus \phi_2$ correspond to the probability that ϕ_1 or ϕ_2 happens. The reason that here are three operators per situation is the same as observed by Huth and Kwiatkowska [HK97]. If we know the probabilities that ϕ_1 and ϕ_2 happen, we also need to know the dependence between ϕ_1 and ϕ_2 to be able to compute the probability that both ϕ_1 and ϕ_2 happen and the probability that ϕ_1 or ϕ_2 happens. If this dependence is unknown, there is a range of values possible.
For the probability that both ϕ_1 and ϕ_2 happen $\phi_1 \wedge \phi_2$ gives the upper bound of this range, $\phi_1 \cdot \phi_2$ gives the exact value in case of independence and $\phi_1 \ominus \phi_2$ gives the lower bound of this range. For the probability that ϕ_1 or ϕ_2 happens $\phi_1 \vee \phi_2$ gives the lower bound of this range, $\phi_1 \odot \phi_2$ gives the exact value in case of independence and $\phi_1 \oplus \phi_2$ gives the upper bound of this range.

In section 2.1 we sketched a similar situation. If we know $P(A)$ and $P(B)$ for events A and B but not the dependence between the two, there is a range of values possible for $P(A \cap B)$ (and $P(A \cup B)$). In case of the lower and upper bounds of this range we said that A and B are maximally dependent. We can apply the same to $pL\mu^{\odot}_{\oplus}$-formulas.
For the formulas $\phi_1 \wedge \phi_2$ and $\phi_1 \vee \phi_2$ it is assumed that ϕ_1 and ϕ_2 are maximally positively dependent. In the corresponding cases shown figures 2a and 2b ($A \subseteq B$ and $B \subseteq A$), the semantics of the operators $\wedge$ and $\vee$ are reflected. For instance, if we want to know $\phi_1 \wedge \phi_2$, we take the minimum of the two. Similarly in figure 2a we have that $P(A \cap B) = P(A)$ which is smaller than $P(B)$ since $A \subseteq B$.
For the formulas $\phi_1 \ominus \phi_2$ and $\phi_1 \oplus \phi_2$ it is assumed that ϕ_1 and ϕ_2 are maximally negatively dependent. In the corresponding cases shown figures 2c and 2d ($A \cap B = \emptyset$ and $A \cup B = \Omega$), the semantics of the operators $\ominus$ and $\oplus$ are reflected. For instance the formula $\phi_1 \oplus \phi_2$ equals the sum of the two if it does not exceed 1, else it is 1. Similarly in figure 2c we have that $P(A \cup B) = P(A) + P(B)$ which cannot exceed 1 since $A \cap B = \emptyset$ and in figure 2d we have that $P(A \cup B) = 1$ since $A \cup B = \Omega$ which causes $P(A) + P(B)$ to be at least 1.

Operator $+_\lambda$

Semantically, for any λ in the interval $[0, 1]$ the operator $+_\lambda$ is defined as

$$[\![\phi_1 +_\lambda \phi_2]\!]_e(s) = [\![\phi_1]\!]_e(s) \mathbin{+\!\!\boldsymbol{\lambda}} [\![\phi_2]\!]_e(s)$$

The formula $\phi_1 +_\lambda \phi_2$ corresponds to a probabilistically weighted sum between the behaviours expressed by ϕ_1 and ϕ_2. With probability λ we check for the behaviour expressed by ϕ_1 and with probability $1-\lambda$ we check for the behaviour expressed by ϕ_2. Note that this adds another layer of probabilistic behaviour on top of the probabilistic behaviour of the model.

Operators $\langle a \rangle$ and $[a]$

Semantically, for any action $a \in A$ the operators $\langle a \rangle$ and $[a]$ are defined as

$$[\![\langle a \rangle \phi]\!]_e(s) = \bigsqcup_{(s,a,\delta) \in T} \{ \sum_{s' \in S} \delta(s') \cdot [\![\phi]\!]_e(s') \}$$

$$[\![[a] \phi]\!]_e(s) = \bigsqcap_{(s,a,\delta) \in T} \{ \sum_{s' \in S} \delta(s') \cdot [\![\phi]\!]_e(s') \}$$

where $\bigsqcap$ and $\bigsqcup$ are defined over the lattice $\langle [0, 1], \leq \rangle$.
These are the only operators that actually use the probabilistic behaviour of the model. Both the diamond operator $\langle a \rangle \phi$ and the box operator $[a]\phi$ calculate the probabilistically weighted sum over all successor states for the probabilistic behaviour of a transition, but they differ in the non-deterministic behaviour. Whereas the diamond operator picks the maximum for all outgoing a-transitions, the box operator picks the minimum.
Intuitively, the formula $\langle a \rangle \phi$ is optimistic: it returns the maximum probability that after an a-step the behaviour desired by ϕ will happen. Dually, the formula $[a]\phi$ is pessimistic: it returns the minimum probability that after an a-step the behaviour desired by ϕ will happen.

Operators $\mu X.$ and $\nu X.$

Semantically, for any variable $X \in Var$ the operators $\mu X.$ and $\nu X.$ are defined as

$$[\![\mu X.\phi]\!]_e(s) = \bigsqcap\{f \in [0,1]^S \mid [\![\phi]\!]_{e[X:=f]} \dot{\leq} f\}(s)$$
$$[\![\nu X.\phi]\!]_e(s) = \bigsqcup\{f \in [0,1]^S \mid f \dot{\leq} [\![\phi]\!]_{e[X:=f]}\}(s)$$

where $\bigsqcap$ and $\bigsqcup$ are defined over the lattice $\langle [0,1]^S, \dot{\leq}\rangle$.
Apart from the difference in lattice, the definitions of the fixpoint operators $\mu X.$ and $\nu X.$ for $pL\mu^{\odot}_{\oplus}$ are the same as for $L\mu$. In [Mio12] it is proven that the least (μ) and greatest (ν) fixpoints exist for $pL\mu^{\odot}_{\oplus}$-formulas of the form $\sigma X.\phi$ by the Knaster-Tarski theorem (theorem 2.1), since all other operators are monotone.

Since the fixpoints exist we could use the iterative method to calculate the fixpoint, where we start with $\lambda s.0$ for the least fixpoint and $\lambda s.1$ for the greatest fixpoint. However, unlike the lattice used for $L\mu$, the lattice $[0,1]^S$ used for $pL\mu^{\odot}_{\oplus}$ has an infinite size due to the infinite nature of the set of real numbers, even within an interval. Because of this, termination of the iterative method is often not the case. In section 6.4.1, we will look into this in more detail.

6.2.1 Duality

Every syntactic element (except the variable) has its dual. We define $\overline{\phi}$ to be the dual of a formula ϕ, such that $\overline{\overline{\phi}} \equiv \phi$. See below for the dual of each element. Proof of each duality is given in [Mio12].

$$\begin{aligned}
\overline{\underline{0}} &= \underline{1} \\
\overline{\underline{1}} &= \underline{0} \\
\overline{\underline{\lambda}} &= 1 - \underline{\lambda} \\
\overline{\overline{X}} &= X \\
\overline{\phi_1 \wedge \phi_2} &= \overline{\phi_1} \vee \overline{\phi_2} \\
\overline{\phi_1 \vee \phi_2} &= \overline{\phi_1} \wedge \overline{\phi_2} \\
\overline{\phi_1 \cdot \phi_2} &= \overline{\phi_1} \odot \overline{\phi_2} \\
\overline{\phi_1 \odot \phi_2} &= \overline{\phi_1} \cdot \overline{\phi_2} \\
\overline{\phi_1 \ominus \phi_2} &= \overline{\phi_1} \oplus \overline{\phi_2} \\
\overline{\phi_1 \oplus \phi_2} &= \overline{\phi_1} \ominus \overline{\phi_2} \\
\overline{\phi_1 +_\lambda \phi_2} &= \overline{\phi_1} +_\lambda \overline{\phi_2} \\
\overline{\langle a\rangle\phi} &= [a]\overline{\phi} \\
\overline{[a]\phi} &= \langle a\rangle\overline{\phi} \\
\overline{\mu X.\phi} &= \nu X.\overline{\phi[X := \overline{X}]} \\
\overline{\nu X.\phi} &= \mu X.\overline{\phi[X := \overline{X}]}
\end{aligned}$$

For any $pL\mu^{\odot}_{\oplus}$-formula ϕ and state s we have that $[\![\overline{\phi}]\!]_e(s) = 1 - [\![\phi]\!]_e(s)$.

6.3 Dependence

As mentioned before, it is not really possible to determine a probability space such that we can correspond events to $pL\mu^{\odot}_{\oplus}$-formulas. Nevertheless, it is possible to get some indication of dependence between formulas, especially in case of a maximal dependency.
For instance consider the formulas $\phi_1 = \langle a \rangle \underline{1}$ and $\phi_2 = \langle a \rangle \langle b \rangle \underline{1}$ checked in the same state s. One can say that ϕ_2 expresses a stricter property than ϕ_1 does, since ϕ_2 does not only require an a-step but also a b-step afterwards. This suggests that ϕ_1 and ϕ_2 are maximally positively dependent. Therefore it only makes sense to use the operator $\wedge$ instead of $\cdot$ or $\ominus$ or the operator $\vee$ instead of $\odot$ or $\oplus$. A more trivial case would be the case where $\phi_1 \equiv \phi_2$.
We can do a similar thing with maximal negative dependence. For instance consider the formulas $\phi_1 = \langle a \rangle \langle a \rangle \underline{1}$ and $\phi_2 = [a]\underline{0}$ checked in the same state s. These two formulas express mutually exclusive properties: either we can do two a-steps or no a-step at all. This suggests that ϕ_1 and ϕ_2 are maximally negatively dependent. Therefore it only makes sense to use the operator $\ominus$ instead of $\wedge$ or $\cdot$ or the operator $\oplus$ instead of $\vee$ or $\odot$. A more trivial case would be the case where $\phi_1 \equiv \overline{\phi_2}$.

However, dependency is usually not only derived from the formulas. Take for instance the formulas $\phi_1 = \langle a \rangle \underline{1}$ and $\phi_2 = \langle b \rangle \underline{1}$. The properties expressed by the formulas seem to be independent, but this dependence actually depends on the model these formulas are checked on. If in each state where an a-action is possible, a b-action is also possible or vice versa, ϕ_1 and ϕ_2 are maximally positively dependent. On the other extreme, if in each state it's impossible to do both an a and a b-action or in each state it's possible to do at least one of the two, ϕ_1 and ϕ_2 are maximally negatively dependent.
It is also possible that it does not matter what operator to use to combine formulas ϕ_1 and ϕ_2. This is true when $[\![\phi_1]\!]_e(s) = 0$, $[\![\phi_1]\!]_e(s) = 1$, $[\![\phi_2]\!]_e(s) = 0$ or $[\![\phi_2]\!]_e(s) = 1$ for a state s. In these cases it always holds that $[\![\phi_1 \vee \phi_2]\!]_e(s) = [\![\phi_1 \cdot \phi_2]\!]_e(s) = [\![\phi_1 \ominus \phi_2]\!]_e(s)$ and $[\![\phi_1 \wedge \phi_2]\!]_e(s) = [\![\phi_1 \odot \phi_2]\!]_e(s) = [\![\phi_1 \oplus \phi_2]\!]_e(s)$.

Note that the examples shown above are rather simple and intuitive. For arbitrary formulas and models the degree of dependency is usually not easy to see, especially since both the formula and the model play a role in determining the dependence.
However, if one knows what formulas ϕ_1 and ϕ_2 intuitively mean, it is sometimes possible to find the right operator. For instance, consider the PLTS in figure 3a. Let $\phi_1 = \langle throwA \rangle \langle v2 \rangle \underline{1}$ and $\phi_2 = \langle throwB \rangle \langle v2 \rangle \underline{1}$. As shown before, $[\![\phi_1]\!]_e(s_0) = \frac{1}{2}$ and $[\![\phi_2]\!]_e(s_0) = \frac{1}{3}$. Intuitively, ϕ_1 denotes the probability of throwing a 2 with die A and ϕ_2 denotes the probability of throwing a 2 with die B. Realistically, we can say that the result of throwing die A is independent of the result of throwing die B. Therefore, it makes sense to use the operator $\cdot$ for the probability that both happen, resulting in a probability of $\frac{1}{6}$. We conclude that it is not always as easy to know whether to use the operator $\wedge$, $\cdot$ or $\ominus$ for and whether to use the operator $\vee$, $\odot$ or $\oplus$. However, one can always get the upper bound using the operators $\wedge$ or $\oplus$ and the lower bound using the operators $\vee$ or $\ominus$ to get the range of possible values. Optionally, the operator $+_\lambda$ can then be used to select a certain spot within this range.

6.4 Fixpoints

As mentioned before, the fixpoint operators $\mu X.$ and $\nu X.$ for some variable X are defined similarly as those in $L\mu$. Therefore they have a similar intuitive meaning: the least fixpoint operator $\mu X.$ concerns finite behaviour or eventuality, whereas the greatest fixpoint operator $\nu X.$ concerns infinite behaviour or globality. For instance, the formula $\nu X.\langle a\rangle X$ gives the probability of an infinite a-path, $\mu X.(\langle a\rangle X \vee \langle b\rangle \underline{1})$ gives the probability that there is an a-path where eventually a b-step is possible and the formula $\nu X.([a]X \wedge \langle b\rangle \underline{1})$ gives the probability that in any a-path it is globally (in any state of the path) possible to do a b-step. In this section we will use examples of least fixed points only, but the reasoning and results hold for the greatest fixpoint dually.

6.4.1 Computation of fixpoints

The main difference between the fixpoint operators used in $pL\mu^{\odot}_{\oplus}$ and $L\mu$ is the lattice that is used, which for $pL\mu^{\odot}_{\oplus}$ is $[0,1]^S$ where S is the set of states. Since the interval $[0,1]$ has the same cardinality as $\mathbb{R}$, the size of the lattice $\langle [0,1]^S, \dot{\leq}\rangle$ is infinite. Consequently, computing a fixpoint with the iterative method may not terminate. This however does not imply that it is impossible to get the exact solution. We will show this using an example.

Example 6.1. Again, let A be a two-sided die and B be a three-sided die and assume that we would like to use die B, but we have lost it. Fortunately, we can model a three sided die using die A, as seen in the PLTS in figure 6. Now to be sure that this model is correct, we want to assert that throwing a value i (in the PLTS represented by the action vi) happens with probability $\frac{1}{3}$. Let us focus on the probability of throwing the value 1. This can be computed from the formula $\mu X.(\langle \mathit{throwA}\rangle X \vee \langle \mathit{v1}\rangle \underline{1})$, which denotes the probability of eventually having value 1 after a finite number of throws. We will try to do this iteratively. Let $f(U) = [\![\langle \mathit{throwA}\rangle X \vee \langle \mathit{v1}\rangle \underline{1}]\!]_{e[X:=U]}$ where $U \in [0,1]^S$. Since we are looking for the least fixpoint, let $f_0 = \lambda s.0$. See the below table for the values of f_α for a number of iterations α.

α	0	1	2	3	4	5	6	7	8	9
$f_\alpha(s_0)$	0	0	0	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{5}{16}$	$\frac{5}{16}$	$\frac{21}{64}$	$\frac{21}{64}$	$\frac{85}{256}$
$f_\alpha(s_1)$	0	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{5}{8}$	$\frac{5}{8}$	$\frac{21}{32}$	$\frac{21}{32}$	$\frac{85}{128}$	$\frac{85}{128}$
$f_\alpha(s_2)$	0	0	0	0	0	0	0	0	0	0
$f_\alpha(s_3)$	0	1	1	1	1	1	1	1	1	1
$f_\alpha(s_4)$	0	0	0	0	0	0	0	0	0	0
$f_\alpha(s_5)$	0	0	0	0	0	0	0	0	0	0

We do not show all iterations simply because this does not terminate. However, since the least fixpoint exists, we know that there is an ordinal β such that $f_\beta(s) = f_\gamma(s)$ for any state s and any ordinal γ where $\beta < \gamma$.
Note that a repeating pattern occurs in the table. Due to the probabilistic transitions and the loop in the PLTS, we have that $f_{\alpha+2}(s_0) = (\frac{1}{2}\cdot f_\alpha(s_0) + \frac{1}{2})\cdot\frac{1}{2}$.

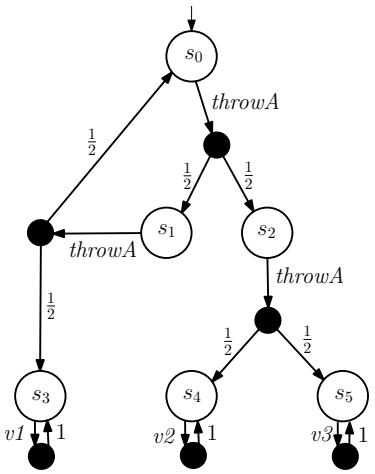


Figure 6: A PLTS that models a three sided die with a two sided die. Each state is given a name s_i so that they can be referred to.

Again let β be the ordinal such that f_β is the fixpoint, then we know that $f_\beta(s_0) = f_{\beta+2}(s_0) = (\frac{1}{2} \cdot f_\beta(s_0) + \frac{1}{2}) \cdot \frac{1}{2}$. Solving this equation will result in $f_\beta(s_0) = \frac{1}{3}$, which is the desired result. In a similar way, one can verify that throwing a 2 and throwing a 3 both happen with probability $\frac{1}{3}$. Note that it does not matter whether we use the operator $\vee$, $\odot$ or $\oplus$ in the $pL\mu^{\odot}_{\oplus}$-formula in this case, since $[\![\langle v1\rangle \underline{1}]\!](s)$ can only be 0 or 1 for any state s.

The above example shows a rather repetitive behaviour. Every other iteration either the value of $f_\alpha(s)$ does not change since it is waiting for a successor to change or $f_\alpha(s)$ does change, but less than the last time it changed. This behaviour however is not representative. For instance, the value of $f_\alpha(s)$ may also change more compared to the previous iteration it changed. This mainly depends on the model, which will be shown in the below example.

Example 6.2. See figure 7 for an example PLTS that contains two linear sequences of states which we will refer to as arms. We want to check the probability of eventually a b-step, which for this model can be described with the $pL\mu^{\odot}_{\oplus}$-formula $\phi = \mu X.(\langle a\rangle X \vee \langle b\rangle \underline{1})$. Then let $f(U) = [\![\langle a\rangle X \vee \langle b\rangle \underline{1}]\!]_{e[X:=U]}$ where $U \in [0,1]^S$ for which we want to know the least fixpoint. Using the iterative method in each arm the value 1 is propagated to the left one state per iteration. Assuming $n < m$, the value from the upper arm will have its effect on $f_\alpha(s_0)$ first, resulting in $f_n(s_0) = \mu$. Later when $\alpha = m$ is reached, we will have that $f_n(s_0) = 1$. Now if we pick $\mu > \frac{1}{2}$, the second increase of $f_\alpha(s_0)$ is smaller than the first increase, but if we would pick $\mu < \frac{1}{2}$, the second increase is greater instead.

6.4.2 Infinite behaviour

In the dice example above, we mentioned that the probabilities of eventually reaching states s_3, s_4 and s_5 are all equal to $\frac{1}{3}$, summing to 1 as expected for a three sided die. However, one could say that it is also possible to loop through states s_0 and s_1 indefinitely, never reaching any of the aforementioned states.

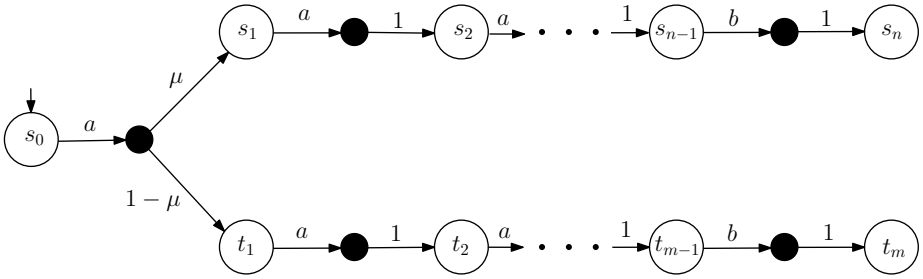


Figure 7: A PLTS that depending on $\mu \in [0, 1]$ probabilistically branches into two finite arms of length $m, n \in \mathbb{N}$.

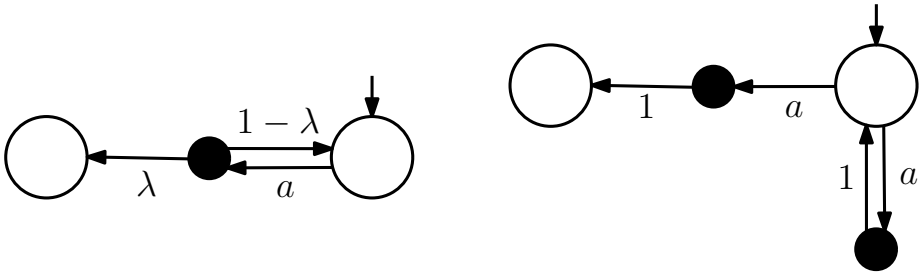


Figure 8: Two similar PLTSs except for the choice concerning the action a. The left PLTS has a probabilistic choice whereas the right PLTS has a non-deterministic choice.

To show why this behaviour appears to be negligible, we will use the two PLTSs as shown in figure 8. Both are the same except that one chooses the state after an a-action probabilistically whereas the other does this non-deterministically. Let us consider the $pL\mu^{\odot}_{\oplus}$-formula $\phi_1 = \mu X.[a]X$ and the syntactically equivalent $L\mu$-formula $\phi_2 = \mu X.[a]X$. Both formulas require all a-paths to be finite, but ϕ_1 is concerned with the probability that this is the case whereas ϕ_2 is concerned with whether it is true. On the right PLTS $[\![\phi_1]\!]_e(s_0) = 0$ because it is possible to non-deterministically choose to take an infinite a-loop. Converting the PLTS to an LTS we have that $[\![\phi_2]\!]_e(s_0) = \mathit{false}$ for the same reason. When ϕ_1 is applied to the left PLTS however, it returns the value 1, while there is still an infinite a-loop possible.
The main difference that causes this is how the different choices are handled. In case of non-deterministic choice, we can't say anything about the likeliness the two possibilities, so we either choose to be optimistic using the diamond operator or pessimistic using the box operator. In the above example we chose to be pessimistic, which resulted in 0 since an infinite a-path is possible. In case of probabilistic choice however we do know how likely the two possibilities will appear. In $pL\mu^{\odot}_{\oplus}$ we use this to our advantage. The probability of the only outcome that is undesired by the given formula, namely the infinite a-loop, equals λ^{∞}, which approaches 0.
From this example we can see that a resulting value of 1 does not always mean that the property specified will always happen. There may be an outcome undesired by the property, but its measure is infinitesimal. Therefore, this value

1 is usually interpreted as that it *almost always* happens. Dually, we can apply the same reasoning for the value 0, interpreting it as that a property *almost never* happens.

6.4.3 Dependence and non-determinism in fixpoints

In the reachability properties in section 6.4.1 it didn't matter whether we used the operator $\vee$, $\odot$ or $\oplus$, since formulas of the form $\langle b \rangle \underline{1}$ always result in either 0 or 1. However if this is not the case, the resulting values using different operators may differ.

In $L\mu$ a formula $\mu X.(\langle a \rangle X \vee \phi)$, which expresses whether possibly eventually ϕ will hold, consists of two parts: the possibility of getting to a state via a-steps and question whether ϕ holds on this state. Similarly, the $pL\mu^{\odot}_{\oplus}$-formula $\mu X.(\langle a \rangle X \bigtriangledown \phi)$ where $\bigtriangledown \in \{\vee, \odot, \oplus\}$, which expresses the probability that possibly eventually ϕ happens, consists of two parts: the probability of going to a state using a-steps and the probability of ϕ in that state. Intuitively, one can say that these two parts are independent, so we should use $\bigtriangledown = \odot$. However, the subformula $\langle a \rangle X$ does not only contain the probability of getting to a state using a-steps.
Formally, as described in theorem 2.1 the least fixpoint of a function f can be expressed as $\bigsqcup\{f_\alpha \mid \mathit{ordinal}\ \alpha\}$ where $f_0 = \bot$, $f_{\alpha+1} = f(f_\alpha)$ and $f_\lambda = \bigsqcup\{f_\alpha \mid \alpha < \lambda\}$ for all limit ordinals λ. For the fixpoint formula above we can define the corresponding function $f(U) = [\![\langle a \rangle X \bigtriangledown \phi]\!]_{e[X:=U]}$ where $U \in [0,1]^S$. In $L\mu$, where $\bot = \emptyset$, we can interpret the value of $f_\alpha(X)$ as the set of states that are capable of reaching a state where ϕ holds within $\alpha - 1$ a-steps. Similarly in $pL\mu^{\odot}_{\oplus}$, where $\bot = \lambda s.0$, we can interpret the value of $f_\alpha(X)$ as the probability of possibly eventually ϕ within $\alpha - 1$ a-steps.
For an ordinal α, the subformula $\langle a \rangle X$ corresponds to $\langle a \rangle f_{\alpha-1}$, which expresses the probability of doing an a-step to a state times the probability that possibly eventually ϕ happens within $\alpha - 2$ a-steps from that state. So actually, the dependence between $\langle a \rangle X$ and ϕ is the dependence between the event that possibly eventually ϕ happens in at least 1 (and at most α) a-steps from a state s and the event of ϕ in s.
In case $\bigtriangledown = \vee$, the maximum of the two options is picked. In case $\bigtriangledown = \oplus$, the probabilities of the two options are summed. In case $\bigtriangledown = \odot$, the two options are assumed to be independent of each other.
Sometimes however, one does not only want to check for eventuality via a single action but via multiple actions, such as with the formula $\mu X.(\langle a \rangle X \bigtriangledown \langle b \rangle X \bigtriangledown' \phi)$. Here a similar thing holds as above: $\bigtriangledown = \vee$ takes the path resulting in the highest probability, $\bigtriangledown = \oplus$ sums the possible paths and $\bigtriangledown = \odot$ chooses the paths independently.

To show the difference of effect on the values of f_α between using $\vee$, $\odot$ or $\oplus$ in a fixpoint formula we will use the example below.

Example 6.3. We define a WxH game board of squares. One starts at the bottom row on the middle square and from each square one can choose to move left (action *moveLeft*) or right (action *moveRight*). When moving left, with probability $\frac{1}{2}$ one will actually move to the square on the left, with probability

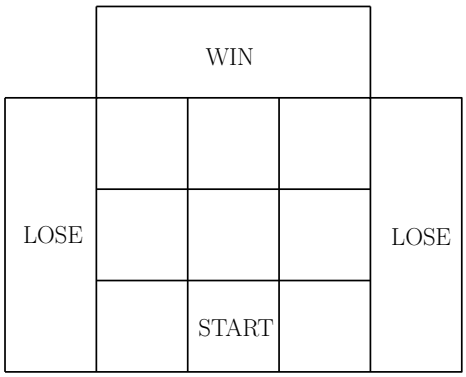


Figure 9: The 3x3 board the game is played on.

$\frac{1}{3}$ to the square in front and otherwise to the square to the right. Moving to the right is symmetric. If one moves off the board on the side the game is lost and if one moves off the board on the top the game is won. See figure 9 for a 3x3 board and see figure 10 for the corresponding PLTS. The layout of the states of the PLTS is the same as layout of the tiles on the board.
The probability to win the game can be expressed by

$$\phi_\nabla = \mu X.((\langle \mathit{moveLeft} \rangle X \ \nabla \ \langle \mathit{moveRight} \rangle X) \vee \langle \mathit{won} \rangle \underline{1})$$

where ∇ can be either $\vee$, $\odot$ or $\oplus$. Let

$$f^\nabla(U) = [\![(\langle \mathit{moveLeft} \rangle X \ \nabla \ \langle \mathit{moveRight} \rangle X) \vee \langle \mathit{won} \rangle \underline{1}]\!]_{e[X:=U]},$$

then f^∇_α expresses the probability of eventually winning the game within $\alpha - 1$ steps. See figure 11 for the values of $f^\vee_\alpha$ for a few iterations α. The values per state are given in the same layout as the states in figure 10, except for the state s_{lost}. Applying f^∇_α to s_{lost} results in 0 for any α, therefore it is omitted.
From these values, we can indeed see that $f^\vee_\alpha$ expresses the probability of reaching the winning state within $\alpha - 1$ moves given a state *when choosing the move that gives the best result.* Since the operator $\vee$ only allows us to take one choice, we can assign each state a move that will lead to the best result (a strategy). For states s_{tl}, s_l and s_{bl} this is *moveRight*, for states s_{tr}, s_r and s_{br} this is *moveLeft* and for states s_t, s_m and s_b it does not matter due to symmetry of the two actions. Note that if we would change one of the two actions such that going forward has a greater probability, this action is likely the best move for states s_t, s_m and s_b.
As example of this we will look at the state s_t for which $f^\vee_3(s_t) = \frac{5}{9}$ as shown in figure 11c. Note that $f^\vee_3$ denotes the probability of reaching the winning state within two steps. Due to the probabilistic choices this leaves us with three possible paths: going up, going left and then up or going right and then up. Let *moveLeft* be the strategy in this state s_t. Then the path going up occurs with probability $\frac{1}{3}$, the path going left and then up occurs with probability $\frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6}$ and the path going right and then up occurs with probability $\frac{1}{6} \cdot \frac{1}{3} = \frac{1}{18}$. If we sum these probabilities (by definition of the diamond operator), we get the probability of $\frac{5}{9}$ as is given by $f^\vee_3(s_t)$.

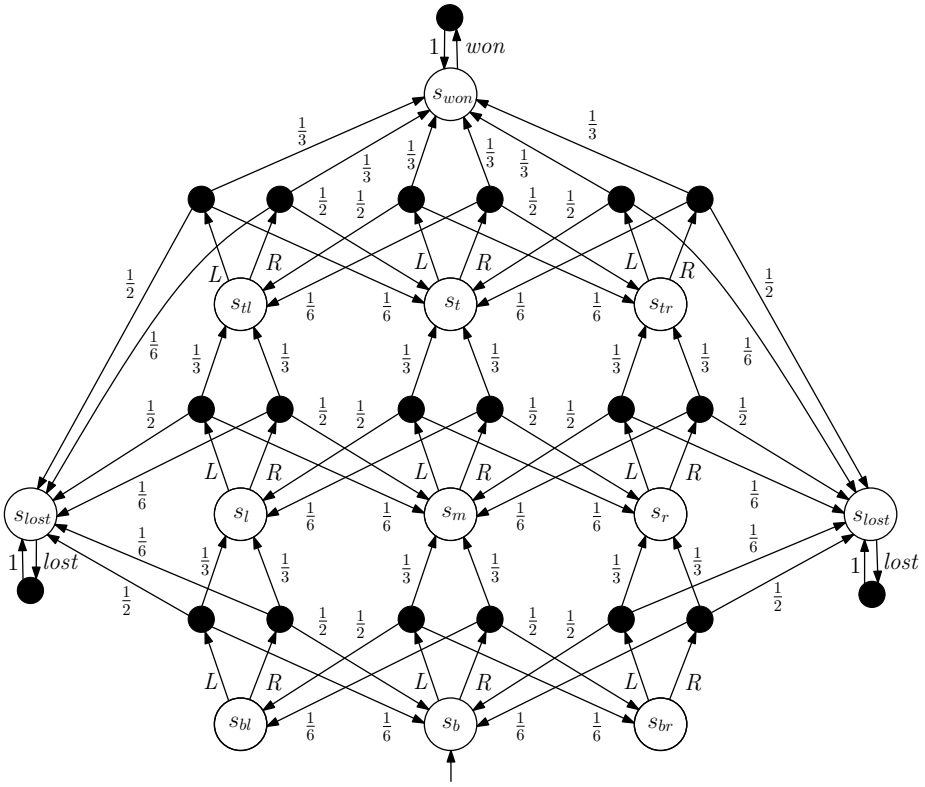


Figure 10: The PLTS of a 3x3 board game described in section 6.4.3. The actions *moveLeft* and *moveRight* are abbreviated to L and R respectively. The state s_{lost} is given twice to make it visually more pleasing.

	1	
0	0	0
0	0	0
0	0	0

(a) Values of $f_1^\vee$.

	1	
$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$
0	0	0
0	0	0

(b) Values of $f_2^\vee$.

	1	
$\frac{1}{2}$	$\frac{5}{9}$	$\frac{1}{2}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
0	0	0

(c) Values of $f_3^\vee$.

	1	
$\frac{11}{18}$	$\frac{2}{3}$	$\frac{11}{18}$
$\frac{2}{9}$	$\frac{7}{27}$	$\frac{2}{9}$
$\frac{1}{27}$	$\frac{1}{27}$	$\frac{1}{27}$

(d) Values of $f_4^\vee$.

	1	
$\frac{2}{3}$	$\frac{20}{27}$	$\frac{2}{3}$
$\frac{1}{3}$	$\frac{10}{27}$	$\frac{1}{3}$
$\frac{5}{54}$	$\frac{1}{9}$	$\frac{5}{54}$

(e) Values of $f_5^\vee$.

	1	
$\frac{19}{27}$	$\frac{7}{9}$	$\frac{19}{27}$
$\frac{11}{27}$	$\frac{38}{81}$	$\frac{11}{27}$
$\frac{1}{6}$	$\frac{5}{27}$	$\frac{1}{6}$

(f) Values of $f_6^\vee$.

Figure 11: Values of $f_\alpha^\vee$ for the first few α arranged like the states in the PLTS in figure 10. The two losing states are left out since their value is always 0.

1			1			1		
$\frac{3}{4}$	$\frac{5}{6}$	$\frac{3}{4}$	0.916	0.997	0.916	1	1	1
$\frac{7}{12}$	$\frac{2}{3}$	$\frac{7}{12}$	0.896	0.995	0.896	1	1	1
$\frac{11}{24}$	$\frac{19}{36}$	$\frac{11}{24}$	0.891	0.994	0.891	1	1	1

Figure 12: Values of the least fixpoint of $f^{\vee}$, $f^{\odot}$ and $f^{\oplus}$ respectively.

See figure 12 for the values of the least fixpoint of f^{∇}, which are also the resulting values of $[\![\phi_{\nabla}]\!]_e$ when checked in each state. When we take $\nabla = \vee$, the probability to win the game making optimal choices equals $\frac{19}{36}$. This shows that choosing the optimal moves really helps: although the winning edge is further away than the losing edges, the probability to win is greater than a half. When using the operators $\odot$ or $\oplus$ the probabilities are far greater. This is because, as mentioned before, all possible deterministic choices are explored and used in the result.

All in all, using the operators $\vee$, $\odot$ and $\oplus$ we can alter how we want to interpret the non-deterministic choice in reachability formulas. However, this only applies to non-deterministic choice between different actions. Same-action non-deterministic choice is handled by the diamond (or box) operator which always only picks the optimal choice like the operator $\vee$ does. So if we would change the above 3x3 board game example such that the actions *moveLeft* and *moveRight* are changed to the same action *move* and then use the $pL\mu^{\odot}_{\oplus}$-formula $\phi_1 = \mu X.(\langle move \rangle X \vee \langle won \rangle 1)$, we will end up with the same values as for $\phi_{\vee}$. On the other hand, we could choose the worst *move* possible at every non-deterministic choice using the $pL\mu^{\odot}_{\oplus}$-formula $\phi_2 = \mu X.(([move]X \vee \langle won \rangle 1) \wedge [lost]0)$. Note the added term $\wedge[lost]0$: this is to make sure that ϕ_2 evaluates to 0 when checked in s_{lost}. Like with ϕ_1, we can make a formula ϕ_{∇} that is similar to ϕ_2 by taking $\nabla = \wedge$.

6.5 Guidelines for quantitative properties

Before moving on to subjects not related to $pL\mu^{\odot}_{\oplus}$-formulas expressing quantitative properties, we will summarize what is discussed above by giving a few guidelines for creating meaningful formulas. See the table below for these guidelines, which maps $pL\mu^{\odot}_{\oplus}$-formulas to their intuitive meaning. Note that this table is not complete in the sense that every possible $pL\mu^{\odot}_{\oplus}$-formula is covered. Also note that these guidelines are recursively defined, by using ϕ, ϕ_1 and ϕ_2 as subformulas. For instance, combining rows 1 and 7 from the table below can give the formula $\langle a \rangle \underline{1} \cdot \langle b \rangle \underline{1}$, which intuitively means "the probability that the system can do both an a-step and a b-step, assuming both happen independently".

$pL\mu^{\odot}_{\oplus}$-formula	Intuitive meaning: The probability that...
$\langle a\rangle \underline{1}$	the system can do an a-step
$[a]\underline{0}$	the system cannot do an a-step
$\langle a\rangle \phi$	ϕ after an a-step
$[a]\phi$	ϕ after any a-step
$\phi_1 \wedge \phi_2$	both ϕ_1 and ϕ_2, assuming both are maximally positively dependent
$\phi_1 \vee \phi_2$	ϕ_1 or ϕ_2, assuming both are maximally positively dependent
$\phi_1 \cdot \phi_2$	both ϕ_1 and ϕ_2, assuming both are independent
$\phi_1 \odot \phi_2$	ϕ_1 or ϕ_2, assuming both are independent
$\phi_1 \ominus \phi_2$	both ϕ_1 and ϕ_2, assuming both are maximally negatively dependent
$\phi_1 \oplus \phi_2$	ϕ_1 or ϕ_2, assuming both are maximally negatively dependent
$\phi_1 +_\lambda \phi_2$	ϕ_1 with weight λ plus ϕ_2 with weight $1-\lambda$
$\mu X.[a]X$	there is a finite a-path while choosing pessimally
$\nu X.\langle a\rangle X$	there is an infinite a-path, while choosing optimally
$\mu X.(\langle a\rangle X \vee \phi)$	eventually ϕ after a path of a-actions while choosing optimally
$\mu X.((\langle a\rangle X \vee \langle b\rangle X) \vee \phi)$	eventually ϕ after a path of a and b-actions while choosing optimally
$\nu X.([a]X \wedge \phi)$	globally ϕ along a path of a-actions while choosing pessimally
$\nu X.(([a]X \wedge [b]X) \wedge \phi)$	globally ϕ along a path of a and b-actions while choosing pessimally

6.6 Qualitative properties

As Mio shows in [Mio12], $pL\mu^{\odot}_{\oplus}$ is a generalization of $L\mu$. When we do not use the atom $\underline{\lambda}$ or the operator $+_\lambda$, $pL\mu^{\odot}_{\oplus}$ can be used to check qualitative formulas on non-probabilistic PLTSs (LTSs) by using $\underline{0}$ for *false* and $\underline{1}$ for *true*. It does not matter whether one uses the operator $\wedge$, $\cdot$ or $\ominus$ to represent the logical operator $\wedge$. As mentioned before, all three operators are semantically equal when applied to values that are 0 or 1, which is always the case in this context. Similarly, it does not matter whether one uses the operator $\vee$, $\odot$ or $\oplus$ to represent the logical operator $\vee$.

Qualitative threshold modalities

The logic $pL\mu^{\odot}_{\oplus}$ is not only a generalization of $L\mu$, but also of PCTL. As shown in [Mio12], it is possible to translate any PCTL-formula to a $pL\mu^{\odot}_{\oplus}$-formula. To this end, Mio introduces some qualitative operators on formulas that concern probability thresholds.
The property whether a $pL\mu^{\odot}_{\oplus}$-formula ϕ has some positive probability, denoted by $P_{>0}(\phi)$, and the property whether a $pL\mu^{\odot}_{\oplus}$-formula ϕ almost always happens, denoted by $P_{=1}(\phi)$, are defined as follows:

$$P_{>0}(\phi) = \mu X.(\phi \odot X) = \mu X.(\phi \oplus X)$$
$$P_{=1}(\phi) = \nu X.(\phi \cdot X) = \nu X.(\phi \ominus X)$$

Although both options of the above operators are mathematically equivalent, the rightmost option converges more quickly when using the iterative method. Semantically, for any state s the operators can be defined as:

$$[\![P_{>0}(\phi)]\!]_e(s) = \begin{cases} 1 & \textit{if } [\![\phi]\!]_e(s) > 0 \\ 0 & \textit{else} \end{cases}$$
$$[\![P_{=1}(\phi)]\!]_e(s) = \begin{cases} 1 & \textit{if } [\![\phi]\!]_e(s) = 1 \\ 0 & \textit{else} \end{cases}$$

The duals of these operators, $P_{=0}(\phi)$ and $P_{<1}(\phi)$, can be expressed as $P_{=1}(\overline{\phi})$ and $P_{>0}(\overline{\phi})$ respectively. To compare the probabilities of two $pL\mu^{\odot}_{\oplus}$-formulas, Mio introduces the operators $\geq$ and $>$ on $pL\mu^{\odot}_{\oplus}$-formulas defined as:

$$\phi_1 \geq \phi_2 = P_{=1}(\phi_1 \oplus \overline{\phi_2})$$
$$\phi_1 > \phi_2 = P_{>0}(\phi_1 \ominus \overline{\phi_2})$$

Semantically, for any state s they are defined as:

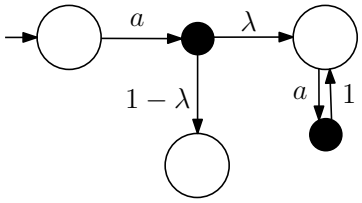
$$[\![\phi_1 \geq \phi_2]\!]_e(s) = \begin{cases} 1 & \textit{if } [\![\phi_1]\!]_e(s) \geq [\![\phi_2]\!]_e(s) \\ 0 & \textit{else} \end{cases}$$
$$[\![\phi_1 > \phi_2]\!]_e(s) = \begin{cases} 1 & \textit{if } [\![\phi_1]\!]_e(s) > [\![\phi_2]\!]_e(s) \\ 0 & \textit{else} \end{cases}$$

If we take $\phi_2 = \underline{\lambda}$, we can write the operators as $P_{\geq\lambda}(\phi_1)$ and $P_{>\lambda}(\phi_1)$ respectively instead, which are very similar to the operators that PCTL adds. Since all introduced operators in this section can be considered as qualitative as they return either 0 (false) or 1 (true), we can say that a formula ϕ that has such a qualitative operator as topmost operator either *holds in a state s* if $[\![\phi]\!]_e(s) = 1$ or *does not hold in a state s* if $[\![\phi]\!]_e(s) = 0$.

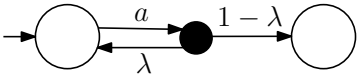
However, these operators need not be the topmost operator. For instance, the formula $\mu X.(\langle a\rangle X \vee P_{=1}(\phi))$ expresses the probability that in a path of a-steps eventually ϕ will hold. Another interesting example is the comparison between the formulas $\phi_1 = P_{\geq\lambda}(\nu X.\langle a\rangle X)$ and $\phi_2 = \nu X.P_{\geq\lambda}(\langle a\rangle X)$.
The formula ϕ_1 holds in a state s if the probability of an infinite a-path starting in s is at least λ. The meaning of ϕ_2 is somewhat more involved. Let $f(X) = [\![P_{\geq\lambda}(\langle a\rangle X)]\!]_e$, from which follows that $f_\alpha = [\![P_{\geq\lambda}(\langle a\rangle f_{\alpha-1})]\!]_e$ for some ordinal α. Then $f_\alpha(s)$ for some state s can only hold if there is an a-step possible from s such that with probability at least λ $f_{\alpha-1}(s')$ holds for successor state(s) s'. This shows that ϕ_2 holds iff there is an infinite a-path where each step occurs with probability at least λ. Note that the formula ϕ_1 is stricter than ϕ_2, since ϕ_1 requires the product of all probabilistic choices to be at least λ. See figure 13 for some examples PLTSs where either ϕ_2 holds or both ϕ_1 and ϕ_2 hold.

6.7 Game semantics of $pL\mu^{\odot}_{\oplus}$

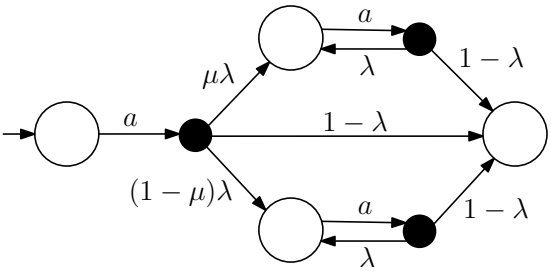
In the past subsections we have tried to find the intuition of $pL\mu^{\odot}_{\oplus}$ by looking at the logic itself, but it is also possible to transform a $pL\mu^{\odot}_{\oplus}$ model checking problem to another type of problem. In Mio's PhD thesis [Mio12], a great part



(a) A PLTS for which both ϕ_1 and ϕ_2 hold.



(b) A PLTS for which only ϕ_2 holds.



(c) A PLTS for which only ϕ_2 holds. The constant μ is a value in the interval $[0, 1]$.

Figure 13: Some example PLTSs that show the semantics of ϕ_1 and ϕ_2 as defined in section 6.6 and the difference between the two.

of the work consists of defining a type of parity game that corresponds to a model checking problems in $pL\mu^{\odot}_{\oplus}$. This type of parity game is called the $2\frac{1}{2}$-player meta-parity game and is a generalisation of the $2\frac{1}{2}$-player parity game [Zie04]. We will explain this type of parity game and give a translation from a $pL\mu^{\odot}_{\oplus}$ model checking problem which can both be found in [Mio12].
A $2\frac{1}{2}$-player meta-parity game is a game played between two players named even ($\diamond$) and odd ($\Box$) and a third "player" called nature ($\circ$). The game is played on a graph of vertices and edge by placing a token on a vertex and then moving this token around. How this token is moved depends on what vertex it is on. There are five types of vertices.

- Regular vertices owned by player even or odd. If the token is on such a vertex, the player that owns the vertex may move the token to a successor vertex.
- Vertices owned by nature. If the token is on such a vertex, the token moves to a successor vertex depending on a probability distribution.
- Branching vertices owned by player even or odd. If the token is on such a vertex, the token is copied to each successor vertex, resulting in a number of independent subgames.

Each vertex also has a priority, which will be explained later.
Formally, let $V_\diamond$ be the set of regular vertices owned by player even, $V_\Box$ be the set of regular vertices owned by player odd, $V_\circ$ be the set of vertices owned by nature ($\circ$), $B_\diamond$ be the set of branching vertices owned by player even, $B_\Box$ be the set of branching vertices owned by player odd and $V = V_\diamond \cup V_\Box \cup V_\circ \cup B_\diamond \cup B_\Box$. Then a $2\frac{1}{2}$-player meta-parity game is defined as follows:

Definition 6.1. A $2\frac{1}{2}$*-player meta-parity game* is a graph defined as a tuple $\langle V_\diamond, V_\square, V_\circ, B_\diamond, B_\square, Pr, E, E_\circ \rangle$ where $Pr : V \to \mathbb{N}$ assigns a priority to a vertex, $E : V \setminus V_\circ \to 2^V$ assigns to a vertex owned by a player its successors where $E(v) \neq \emptyset$ for every $v \in V$ and $E_\circ : V_\circ \to (V \to [0,1])$ assigns to a vertex owned by nature a probability distribution over its successors.

A play in this game is the path that the token takes through the graph. A play is winning for a player according to the parity condition:

Definition 6.2. Let π be a play and let $\mathit{infp}(\pi)$ be the set of priorities that occur infinitely often in π. Then a play π is *winning* for player even if $\min(\mathit{infp}(\pi))$ is even. A play π is *winning* for player odd if $\min(\mathit{infp}(\pi))$ is odd.

Both players even and odd can form a strategy $\sigma_\diamond$ and $\sigma_\square$ that says, given a vertex, to what successor vertex they will move the token. If we only look at plays that follow these strategies, we could say that a strategy is winning in a vertex if all plays from the vertex are winning.
However, a play may also cross vertices owned by nature. Since nature moves the token according to a probability distribution, such a play (following a strategy) can only happen with a certain probability. Therefore instead of a winning strategy we have an optimal strategy: the strategy that results in the greatest probability of winning in a vertex. For branching vertices however this works differently. The probability that a player wins in a branching vertex owned by him/herself equals the probability that player even wins any subgame. The probability that a player wins in a branching vertex owned by the other player equals the probability that player even wins all subgames.
For the formal definitions of plays, strategies and winning we refer to [Mio12].

Mio also gives a translation from a $pL\mu^{\odot}_{\oplus}$ model checking problem to a $2\frac{1}{2}$-player meta-parity game. This translation is based on the translation of McIver and Morgan from the sublogic $pL\mu$ to a $2\frac{1}{2}$-player parity game [MM06].
Th translation is as follows. Every vertex $v \in V \setminus V_\circ$ owned by a player is a tuple (s, ϕ') where $s \in S$ is a state and ϕ' is a subformula of ϕ. Every vertex $v \in V_\circ$ owned by nature is either a tuple (s, ϕ') or a tuple (δ, ϕ') where $s \in S$ is a state, $\delta : S \to [0,1]$ is a probability distribution over states and ϕ' is a subformula of ϕ. In the resulting $2\frac{1}{2}$-player meta-parity game player even aims to maximize the probability that ϕ happens whereas player odd aims to minimize the probability that ϕ happens.
The translation below will be somewhat informal; for a more formal translation see Mio's PhD thesis [Mio12].

Let ϕ be a closed $pL\mu^{\odot}_{\oplus}$-formula and $\mathcal{M} = \langle S, s_0, A, T \rangle$ a PLTS. Then we can create a $2\frac{1}{2}$-player meta-parity game by creating a vertex for every $s \in S$ and every subformula of ϕ. Below we will give the properties of each vertex (s, ϕ') in a case distinction on ϕ'. Note that in case a vertex owned by a player only has one successor vertex it does not matter which player owns the vertex.

- If $\phi' = \underline{0}$, the vertex (s, ϕ') is a regular vertex won by player odd. This can be modeled by giving it an odd priority and by giving it itself as its only successor.
- If $\phi' = \underline{1}$, the vertex (s, ϕ') is a regular vertex won by player even. This

can be modeled by giving it an even priority and by giving it itself as its only successor.

- If $\phi' = \underline{\lambda}$, the vertex (s, ϕ') is owned by nature. From this vertex the token moves to vertex $(s, \underline{1})$ with probability λ and to vertex $(s, \underline{0})$ with probability $1 - \lambda$.
- If $\phi' = X$, the vertex (s, ϕ') is a regular vertex with only one successor, namely the vertex $(s, \sigma X.\phi'')$ where $\sigma X.\phi''$ is a subformula of ϕ.
- If $\phi' = \phi_1 \wedge \phi_2$, the vertex (s, ϕ') is a regular vertex owned by player odd, who can choose to move the token to (s, ϕ_1) or (s, ϕ_2).
- If $\phi' = \phi_1 \vee \phi_2$, the vertex (s, ϕ') is a regular vertex owned by player even, who can choose to move the token to (s, ϕ_1) or (s, ϕ_2).
- If $\phi' = \phi_1 \cdot \phi_2$, the vertex (s, ϕ') is a branching vertex owned by player odd, who can move the token to create two subgames starting in (s, ϕ_1) and (s, ϕ_2).
- If $\phi' = \phi_1 \odot \phi_2$, the vertex (s, ϕ') is a branching vertex owned by player even, who can move the token to create two subgames starting in (s, ϕ_1) and (s, ϕ_2).
- If $\phi' = \phi_1 \ominus \phi_2$, the vertex (s, ϕ') is a branching vertex owned by player odd, who can move the token to create subgames starting in additional vertices $(s, B_n^{\phi_1,\phi_2})$ for all $n \in \mathbb{N}$, where $A_0^{\phi_1,\phi_2} = \phi'$, $B_0^{\phi_1,\phi_2} = \phi''$, $A_{i+1}^{\phi_1,\phi_2} = A^i \odot B^i$ and $B_{i+1}^{\phi_1,\phi_2} = A^i \cdot B^i$ for $i \in \mathbb{N}$.
- If $\phi' = \phi_1 \oplus \phi_2$, the vertex (s, ϕ') is a branching vertex owned by player even, who can move the token to create subgames starting in additional vertices $(s, A_n^{\phi_1,\phi_2})$ for all $n \in \mathbb{N}$, where $A_0^{\phi_1,\phi_2} = \phi'$, $B_0^{\phi_1,\phi_2} = \phi''$, $A_{i+1}^{\phi_1,\phi_2} = A^i \odot B^i$ and $B_{i+1}^{\phi_1,\phi_2} = A^i \cdot B^i$ for $i \in \mathbb{N}$.
- If $\phi' = \phi_1 +_\lambda \phi_2$, the vertex (s, ϕ') is a vertex owned by nature. From this vertex the token moves to vertex (s, ϕ_1) with probability λ and to vertex (s, ϕ_2) with probability $1 - \lambda$.
- If $\phi' = \langle a \rangle \phi''$, we distinguish between two cases. If there is no a-step possible from s, the vertex (s, ϕ') is the same as the vertex $(s, \underline{0})$. Else, the vertex (s, ϕ') is a regular vertex owned by player even, who can choose to move the token to any of the additional vertices (δ, ϕ'') for all $(s, a, \delta) \in T$. These vertices (δ, ϕ'') are owned by nature and from such a vertex the token moves to vertex (s', ϕ'') with probability $\delta(s')$.
- If $\phi' = [a]\phi''$, we distinguish between two cases. If there is no a-step possible from s, the vertex (s, ϕ') is the same as the vertex $(s, \underline{1})$. Else, the vertex (s, ϕ') is a regular vertex owned by player odd, who can choose to move the token to any of the additional vertices (δ, ϕ'') for all $(s, a, \delta) \in T$. These vertices (δ, ϕ'') are owned by nature and from such a vertex the token moves to vertex (s', ϕ'') with probability $\delta(s')$.
- If $\phi' = \mu X.\phi''$, the vertex (s, ϕ') is a regular vertex with an odd priority and only one successor, namely (s, ϕ'').

- If $\phi' = \nu X.\phi''$, the vertex (s, ϕ') is a regular vertex with an even priority and only one successor, namely (s, ϕ'').

The priorities can then be given to the vertices such that $Pr((s,\phi)) \leq Pr((s,\phi'))$ if ϕ' is a subformula of ϕ for any $s \in S$, while adhering to the odd/even restrictions for $\underline{0}$, $\underline{1}$, $\mu X.\phi''$ and $\nu X.\phi''$.
Note that if we would take a finite PLTS, the resulting $2\frac{1}{2}$-player meta-parity game may be countably infinite due to the translation for the operators $\ominus$ and $\oplus$. If these operators are not present in ϕ however, the resulting $2\frac{1}{2}$-player meta-parity game is finite too.
In Mio's PhD thesis, he shows the equivalence between the result of checking a $pL\mu^{\odot}_{\oplus}$-formula ϕ on a PLTS $\langle S, s_0, A, T\rangle$ and the probability that player even wins vertex (s_0, ϕ) when using optimal strategies in the $2\frac{1}{2}$-player meta-parity game that results from the above translation.

7 An equation system approach for solving $pL\mu^{\odot}_{\oplus}$ model checking problems

Let $\langle S, s_0, A, T\rangle$ be a PLTS and *Var* a set of variables in $S \to [0,1]$. As mentioned before, the syntax of a $pL\mu^{\odot}_{\oplus}$-formula is given by

$$\phi ::= \underline{\lambda} \mid X \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \cdot \phi \mid \phi \odot \phi \mid \phi \ominus \phi \mid \phi \oplus \phi \mid \phi +_\lambda \phi \mid \langle a \rangle \phi \mid [a]\phi \mid \mu X.\phi \mid \nu X.\phi$$

where $\lambda \in [0,1]$ is a probability, $X \in \mathit{Var}$ is a variable and $a \in A$ is an action. The semantics $[\![\phi]\!]_e(s)$ of a $pL\mu^{\odot}_{\oplus}$-formula ϕ checked on state $s \in S$ given an environment $e : \mathit{Var} \to (S \to [0,1])$ is defined as

$$\begin{aligned}
[\![\underline{\lambda}]\!]_e(s) &= \lambda \\
[\![X]\!]_e(s) &= e(X)(s) \\
[\![\phi_1 \wedge \phi_2]\!]_e(s) &= [\![\phi_1]\!]_e(s) \sqcap [\![\phi_2]\!]_e(s) \\
[\![\phi_1 \vee \phi_2]\!]_e(s) &= [\![\phi_1]\!]_e(s) \sqcup [\![\phi_2]\!]_e(s) \\
[\![\phi_1 \cdot \phi_2]\!]_e(s) &= [\![\phi_1]\!]_e(s) \cdot [\![\phi_2]\!]_e(s) \\
[\![\phi_1 \odot \phi_2]\!]_e(s) &= [\![\phi_1]\!]_e(s) \odot [\![\phi_2]\!]_e(s) \\
[\![\phi_1 \ominus \phi_2]\!]_e(s) &= [\![\phi_1]\!]_e(s) \ominus [\![\phi_2]\!]_e(s) \\
[\![\phi_1 \oplus \phi_2]\!]_e(s) &= [\![\phi_1]\!]_e(s) \oplus [\![\phi_2]\!]_e(s) \\
[\![\phi_1 +_\lambda \phi_2]\!]_e(s) &= \lambda \cdot [\![\phi_1]\!]_e(s) + (1-\lambda) \cdot [\![\phi_2]\!]_e(s) \\
[\![\langle a \rangle \phi]\!]_e(s) &= \bigsqcup_{(s,a,\delta) \in T} \{ \sum_{s' \in S} \delta(s') \cdot [\![\phi]\!]_e(s') \} \\
[\![[a]\phi]\!]_e(s) &= \bigsqcap_{(s,a,\delta) \in T} \{ \sum_{s' \in S} \delta(s') \cdot [\![\phi]\!]_e(s') \} \\
[\![\mu X.\phi]\!]_e(s) &= \bigsqcap \{ f \in [0,1]^S \mid [\![\phi]\!]_{e[X:=f]} \dot{\leq} f \}(s) \\
[\![\nu X.\phi]\!]_e(s) &= \bigsqcup \{ f \in [0,1]^S \mid f \dot{\leq} [\![\phi]\!]_{e[X:=f]} \}(s)
\end{aligned}$$

From these semantics we can easily derive a procedure that recurses over the structure of the formula and directly applies the semantics of the formula to the model. See algorithm 2 for pseudo code.

For solving a fixpoint formula this procedure uses the iterative method. This method however does not terminate in the worst case due to the interval $[0,1]$ being infinite. To tackle this we set a maximum number of iterations *MAX-ITER*, but this will lead to an approximation of the real answer. By adjusting this maximum number of iterations we can trade precision for running time. The precision of this approximation however is not easy to determine. As shown in section 6.4.1, the progression of the function values during the iterative method can vary a lot, strongly dependent on the operators used on the $pL\mu^{\odot}_{\oplus}$-formula and on the model the formula is checked on.
In the rest of this section we will introduce a way of solving $pL\mu^{\odot}_{\oplus}$ model checking problems exactly inspired by the work of Mader [Mad97].

Algorithm 2 $naiveChecker(s, \phi)$

if $\phi == \underline{\lambda}$ **then**
  return λ
else if $\phi == X$ **then**
  return $e(X)(s)$
else if $\phi == \phi_1 \wedge \phi_2$ **then**
  return $naiveChecker(s, \phi_1) \sqcap naiveChecker(s, \phi_2)$
else if $\phi == \phi_1 \vee \phi_2$ **then**
  return $naiveChecker(s, \phi_1) \sqcup naiveChecker(s, \phi_2)$
else if $\phi == \phi_1 \cdot \phi_2$ **then**
  return $naiveChecker(s, \phi_1) \cdot naiveChecker(s, \phi_2)$
else if $\phi == \phi_1 \odot \phi_2$ **then**
  return $naiveChecker(s, \phi_1) \odot naiveChecker(s, \phi_2)$
else if $\phi == \phi_1 \ominus \phi_2$ **then**
  return $naiveChecker(s, \phi_1) \ominus naiveChecker(s, \phi_2)$
else if $\phi == \phi_1 \oplus \phi_2$ **then**
  return $naiveChecker(s, \phi_1) \oplus naiveChecker(s, \phi_2)$
else if $\phi == \phi_1 +_\lambda \phi_2$ **then**
  return $naiveChecker(s, \phi_1) +_{\boldsymbol{\lambda}} naiveChecker(s, \phi_2)$
else if $\phi == \langle a \rangle \phi'$ **then**
  if $\neg \exists_\delta : (s, a, \delta) \in T$ **then**
    return 0
  else
    return $\bigsqcup_{(s,a,\delta) \in T} \{ \sum_{s' \in S} \delta(s') \cdot naiveChecker(s', \phi') \}$
  end if
else if $\phi == [a] \phi'$ **then**
  if $\neg \exists_\delta : (s, a, \delta) \in T$ **then**
    return 1
  else
    return $\bigsqcap_{(s,a,\delta) \in T} \{ \sum_{s' \in S} \delta(s') \cdot naiveChecker(s', \phi') \}$
  end if
else if $\phi == \mu X.\phi'$ **or** $\phi = \nu X.\phi'$ **then**
  if $\phi == \mu X.\phi'$ **then**
    $newValues = \lambda s.0$
  else
    $newValues = \lambda s.1$
  end if
  $i = 0$
  while $e(X) \neq newValues$ **and** $i < MAXITER$ **do**
    $e(X) = newValues$
    for $s' \in S$ **do**
      $newValues(s') = naiveChecker(s', \phi')$
    end for
    $i = i + 1$
  end while
end if

7.1 Real equation systems

Since the lattice $\langle [0,1], \leq \rangle$ is a complete lattice and since all operators we have defined on this lattice are monotone, we can create an instance of a FES on this lattice. We will call such a FES over the lattice $\langle [0,1], \leq \rangle$ a Real Equation System (with independent product and truncated sum), denoted as $\text{RES}^{\odot}_{\oplus}$.

Definition 7.1. Let Var be a set of variables in $[0,1]$, then the syntax of a $\text{RES}^{\odot}_{\oplus}$ $\mathcal{E}$ is defined as

$$\mathcal{E} ::= \epsilon \mid (\mu X = f)\ \mathcal{E} \mid (\nu X = f)\ \mathcal{E}$$

where ϵ denotes the empty $\text{RES}^{\odot}_{\oplus}$, $X \in Var$ and $f : [0,1]^{|Var|} \to [0,1]$ is a real formula generated by the syntax

$$f ::= \lambda \mid X \mid f \sqcap f \mid f \sqcup f \mid f \cdot f \mid f \odot f \mid f \ominus f \mid f \oplus f \mid f +_{\boldsymbol{\lambda}} f$$

where $\lambda \in [0,1]$ and $X \in Var$.

For simplicity of notation we will denote multiple applications of $f \sqcap f$, $f \sqcup f$ and $f +_{\boldsymbol{\lambda}} f$ as $\bigsqcap_i f_i$, $\bigsqcup_i f_i$ and $\sum_i c_i \cdot f_i$ respectively. Note that for the latter $\sum_i c_i = 1$ by definition of $+_{\boldsymbol{\lambda}}$. Since the $\text{RES}^{\odot}_{\oplus}$ is an instance of the FES, all definitions and lemmas on FESs defined in section 4.2 also hold for the $\text{RES}^{\odot}_{\oplus}$. A $\text{RES}^{\odot}_{\oplus}$ can be used to represent a model checking problem in $pL\mu^{\odot}_{\oplus}$, just like a BES can be used to represent a model checking problem in $L\mu$. See below for the translation from a model checking problem in $pL\mu^{\odot}_{\oplus}$ to a $\text{RES}^{\odot}_{\oplus}$. Note that we will define this translation only for $pL\mu^{\odot}_{\oplus}$-formulas of the form $\sigma X.\phi$. However, any $pL\mu^{\odot}_{\oplus}$-formula can be given in this form while keeping equivalent semantics by prepending a fixpoint operator with a fresh variable.

Definition 7.2. Let $\mathcal{M} = \langle S, s_0, A, T \rangle$ be a PLTS and $\sigma X.\phi$ a $pL\mu^{\odot}_{\oplus}$-formula. Then the $\text{RES}^{\odot}_{\oplus}$ $\mathbf{E}_R(\sigma X.\phi)$ corresponding to the model checking problem of $\sigma X.\phi$ on $\mathcal{M}$ is created as follows:

$$\begin{aligned}
\mathbf{E}_R(\underline{\lambda}) &= \epsilon \\
\mathbf{E}_R(X) &= \epsilon \\
\mathbf{E}_R(\phi_1 \wedge \phi_2) &= \mathbf{E}_R(\phi_1)\ \mathbf{E}_R(\phi_2) \\
\mathbf{E}_R(\phi_1 \vee \phi_2) &= \mathbf{E}_R(\phi_1)\ \mathbf{E}_R(\phi_2) \\
\mathbf{E}_R(\phi_1 \cdot \phi_2) &= \mathbf{E}_R(\phi_1)\ \mathbf{E}_R(\phi_2) \\
\mathbf{E}_R(\phi_1 \odot \phi_2) &= \mathbf{E}_R(\phi_1)\ \mathbf{E}_R(\phi_2) \\
\mathbf{E}_R(\phi_1 \ominus \phi_2) &= \mathbf{E}_R(\phi_1)\ \mathbf{E}_R(\phi_2) \\
\mathbf{E}_R(\phi_1 \oplus \phi_2) &= \mathbf{E}_R(\phi_1)\ \mathbf{E}_R(\phi_2) \\
\mathbf{E}_R(\phi_1 +_\lambda \phi_2) &= \mathbf{E}_R(\phi_1)\ \mathbf{E}_R(\phi_2) \\
\mathbf{E}_R(\langle a \rangle \phi) &= \mathbf{E}_R(\phi) \\
\mathbf{E}_R([a]\phi) &= \mathbf{E}_R(\phi) \\
\mathbf{E}_R(\sigma X.\phi) &= ((\sigma X_s = \mathbf{E}'_R(s,\phi))\ \textit{for each}\ s \in S)\ \mathbf{E}_R(\phi)
\end{aligned}$$

$$
\begin{aligned}
\mathbf{E}'_R(s, \underline{\lambda}) &= \lambda \\
\mathbf{E}'_R(s, X) &= X_s \\
\mathbf{E}'_R(s, \phi_1 \wedge \phi_2) &= \mathbf{E}'_R(s, \phi_1) \sqcap \mathbf{E}'_R(s, \phi_2) \\
\mathbf{E}'_R(s, \phi_1 \vee \phi_2) &= \mathbf{E}'_R(s, \phi_1) \sqcup \mathbf{E}'_R(s, \phi_2) \\
\mathbf{E}'_R(s, \phi_1 \cdot \phi_2) &= \mathbf{E}'_R(s, \phi_1) \cdot \mathbf{E}'_R(s, \phi_2) \\
\mathbf{E}'_R(s, \phi_1 \odot \phi_2) &= \mathbf{E}'_R(s, \phi_1) \odot \mathbf{E}'_R(s, \phi_2) \\
\mathbf{E}'_R(s, \phi_1 \ominus \phi_2) &= \mathbf{E}'_R(s, \phi_1) \ominus \mathbf{E}'_R(s, \phi_2) \\
\mathbf{E}'_R(s, \phi_1 \oplus \phi_2) &= \mathbf{E}'_R(s, \phi_1) \oplus \mathbf{E}'_R(s, \phi_2) \\
\mathbf{E}'_R(s, \phi_1 +_\lambda \phi_2) &= \mathbf{E}'_R(s, \phi_1) +_{\boldsymbol{\lambda}} \mathbf{E}'_R(s, \phi_2) \\
\mathbf{E}'_R(s, \langle a \rangle \phi) &= \bigsqcup_{(s,a,\delta) \in T} \{ \sum_{s' \in S} \delta(s') \cdot \mathbf{E}'_R(s', \phi) \} \\
\mathbf{E}'_R(s, [a] \phi) &= \bigsqcap_{(s,a,\delta) \in T} \{ \sum_{s' \in S} \delta(s') \cdot \mathbf{E}'_R(s', \phi) \} \\
\mathbf{E}'_R(s, \sigma X.\phi) &= X_s
\end{aligned}
$$

Similar to the translation $\mathbf{E}_B$ to a BES as in definition 4.8, the translation $\mathbf{E}_R$ creates $|S|$ equations for each fixpoint operator. This is because we create a $\mathrm{RES}^{\odot}_{\oplus}$ over the lattice $\langle [0,1], \leq \rangle$ from a logic over the lattice $\langle [0,1]^S, \dot{\leq} \rangle$. This will also introduce a new set of variables Var' of type $[0,1]$, such that $Var' = \{X_s \mid X \in Var, s \in S\}$. Since this translation does not completely adhere to the usual translation $\mathbf{E}$ as in definition 4.4, we cannot use lemma 4.1 to claim the correctness of the solution. Instead, we claim the correctness of the solution of a $\mathrm{RES}^{\odot}_{\oplus}$ with the theorem below.

Theorem 7.1. Let $\langle S, s_0, A, T \rangle$ be a PLTS and $\sigma X.\phi$ be a $pL\mu^{\odot}_{\oplus}$-formula. Then for any variable $X \in Var$, state $s \in S$ and environments $e : Var \to (S \to [0,1])$ and $e' : Var' \to [0,1]$

$$[\![\sigma X.\phi]\!]_e(s) = ([\mathbf{E}_R(\sigma X.\phi)]e')(X_s)$$

if $e(Y)(t) = e'(Y_t)$ for all free variables $Y \in Var$ in $\sigma X.\phi$ and all $t \in S$.

Proof. We will take a similar approach as Mader used to prove theorem 5.1 in her PhD thesis on the correctness of the solution of the BES (in this thesis theorem 4.1). We will first translate a $pL\mu^{\odot}_{\oplus}$ model checking problem to a FES on the lattice $\langle [0,1]^S, \dot{\leq} \rangle$ which we will then to a $\mathrm{RES}^{\odot}_{\oplus}$.
Let $\sigma X.\phi$ be a $pL\mu^{\odot}_{\oplus}$-formula and $\mathcal{M} = \langle S, s_0, A, T \rangle$ a PLTS. Then we first translate the model checking problem of $\sigma X.\phi$ on $\mathcal{M}$ to a FES on the lattice

$\langle [0,1]^S, \dot{\leq} \rangle$ using translations $\mathbf{E}_\mu$ and $\mathbf{E}'_\mu$ as defined below:

$$
\begin{aligned}
\mathbf{E}_\mu(\underline{\lambda}) &= \epsilon \\
\mathbf{E}_\mu(X) &= \epsilon \\
\mathbf{E}_\mu(\phi_1 \wedge \phi_2) &= \mathbf{E}_\mu(\phi_1)\ \mathbf{E}_\mu(\phi_2) \\
\mathbf{E}_\mu(\phi_1 \vee \phi_2) &= \mathbf{E}_\mu(\phi_1)\ \mathbf{E}_\mu(\phi_2) \\
\mathbf{E}_\mu(\phi_1 \cdot \phi_2) &= \mathbf{E}_\mu(\phi_1)\ \mathbf{E}_\mu(\phi_2) \\
\mathbf{E}_\mu(\phi_1 \odot \phi_2) &= \mathbf{E}_\mu(\phi_1)\ \mathbf{E}_\mu(\phi_2) \\
\mathbf{E}_\mu(\phi_1 \ominus \phi_2) &= \mathbf{E}_\mu(\phi_1)\ \mathbf{E}_\mu(\phi_2) \\
\mathbf{E}_\mu(\phi_1 \oplus \phi_2) &= \mathbf{E}_\mu(\phi_1)\ \mathbf{E}_\mu(\phi_2) \\
\mathbf{E}_\mu(\phi_1 +_\lambda \phi_2) &= \mathbf{E}_\mu(\phi_1)\ \mathbf{E}_\mu(\phi_2) \\
\mathbf{E}_\mu(\langle a \rangle \phi) &= \mathbf{E}_\mu(\phi) \\
\mathbf{E}_\mu([a]\phi) &= \mathbf{E}_\mu(\phi) \\
\mathbf{E}_\mu(\sigma X.\phi) &= (\sigma X = \mathbf{E}'_\mu(\phi))\ \mathbf{E}_\mu(\phi)
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{E}'_\mu(\underline{\lambda}) &= \underline{\lambda} \\
\mathbf{E}'_\mu(X) &= X \\
\mathbf{E}'_\mu(\phi_1 \wedge \phi_2) &= \mathbf{E}'_\mu(\phi_1) \wedge \mathbf{E}'_\mu(\phi_2) \\
\mathbf{E}'_\mu(\phi_1 \vee \phi_2) &= \mathbf{E}'_\mu(\phi_1) \vee \mathbf{E}'_\mu(\phi_2) \\
\mathbf{E}'_\mu(\phi_1 \cdot \phi_2) &= \mathbf{E}'_\mu(\phi_1) \cdot \mathbf{E}'_\mu(\phi_2) \\
\mathbf{E}'_\mu(\phi_1 \odot \phi_2) &= \mathbf{E}'_\mu(\phi_1) \odot \mathbf{E}'_\mu(\phi_2) \\
\mathbf{E}'_\mu(\phi_1 \ominus \phi_2) &= \mathbf{E}'_\mu(\phi_1) \ominus \mathbf{E}'_\mu(\phi_2) \\
\mathbf{E}'_\mu(\phi_1 \oplus \phi_2) &= \mathbf{E}'_\mu(\phi_1) \oplus \mathbf{E}'_\mu(\phi_2) \\
\mathbf{E}'_\mu(\phi_1 +_\lambda \phi_2) &= \mathbf{E}'_\mu(\phi_1) +_\lambda \mathbf{E}'_\mu(\phi_2) \\
\mathbf{E}'_\mu(\langle a \rangle \phi) &= \langle a \rangle \mathbf{E}'_\mu(\phi) \\
\mathbf{E}'_\mu([a]\phi) &= [a]\mathbf{E}'_\mu(\phi) \\
\mathbf{E}'_\mu(\sigma X.\phi) &= X
\end{aligned}
$$

Note that this translation is the same as the usual translation $\mathbf{E}$ as in definition 4.4. Therefore, we can use lemma 4.1 to see that $[\![\sigma X.\phi]\!]_e(s) = ([\mathbf{E}_\mu(\sigma X.\phi)]e)(X)(s)$. Next we translate from this FES to a $\mathrm{RES}^{\odot}_{\oplus}$ using translations $\mathbf{E}_M$ and $\mathbf{E}'_M$ as defined below

$$
\begin{aligned}
\mathbf{E}_M(\epsilon) &= \epsilon \\
\mathbf{E}_M((\sigma X = f)\ \mathcal{E}) &= ((\sigma X_s = \mathbf{E}'_M(s,f))\ \textit{for each}\ s \in S)\ \mathbf{E}_M(\mathcal{E})
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{E}'_M(s, \underline{\lambda}) &= \lambda \\
\mathbf{E}'_M(s, X) &= X_s \\
\mathbf{E}'_M(s, f_1 \wedge f_2) &= \mathbf{E}'_M(s, f_1) \sqcap \mathbf{E}'_M(s, f_2) \\
\mathbf{E}'_M(s, f_1 \vee f_2) &= \mathbf{E}'_M(s, f_1) \sqcup \mathbf{E}'_M(s, f_2) \\
\mathbf{E}'_M(s, f_1 \cdot f_2) &= \mathbf{E}'_M(s, f_1) \cdot \mathbf{E}'_M(s, f_2) \\
\mathbf{E}'_M(s, f_1 \odot f_2) &= \mathbf{E}'_M(s, f_1) \odot \mathbf{E}'_M(s, f_2) \\
\mathbf{E}'_M(s, f_1 \ominus f_2) &= \mathbf{E}'_M(s, f_1) \ominus \mathbf{E}'_M(s, f_2) \\
\mathbf{E}'_M(s, f_1 \oplus f_2) &= \mathbf{E}'_M(s, f_1) \oplus \mathbf{E}'_M(s, f_2) \\
\mathbf{E}'_M(s, f_1 +_\lambda f_2) &= \mathbf{E}'_M(s, f_1) +_{\boldsymbol{\lambda}} \mathbf{E}'_M(s, f_2) \\
\mathbf{E}'_M(s, \langle a \rangle f) &= \bigsqcup_{(s,a,\delta) \in T} \{ \sum_{s' \in S} \delta(s') \cdot \mathbf{E}'_M(s', f) \} \\
\mathbf{E}'_M(s, [a] f) &= \bigsqcap_{(s,a,\delta) \in T} \{ \sum_{s' \in S} \delta(s') \cdot \mathbf{E}'_M(s', f) \}
\end{aligned}
$$

Let $\mathcal{E}_\mu$ be an equation system created using the translation $\mathbf{E}_\mu$. What is left to prove is the correctness of the translation $\mathbf{E}_M$. To do so, we first prove that the translation $\mathbf{E}'_M$ preserves the semantic value of a formula f checked on a state s as shown below.

Lemma 7.1. For any formula f that can occur as the right hand side in an equation in $\mathcal{E}_\mu$, any state $s \in S$, any environment $e : Var \to (S \to [0,1])$ and any environment $e' : Var' \to [0,1]$:

$$[\![f]\!]_e(s) = [\![\mathbf{E}'_M(s, f)]\!]_{e'}$$

if $e(X)(s') = e'(X_{s'})$ for all variables $X \in Var$ in f and all states $s' \in S$.

Proof. We will prove this using structural induction on f with the induction hypothesis (IH) that the above holds for the subformulas f_i of f.

- If $f = \underline{\lambda}$: $[\![f]\!]_e(s) = \lambda = [\![\mathbf{E}'_M(s, f)]\!]_{e'}$
- If $f = X$: $[\![f]\!]_e(s) = e(X)(s) = e'(X_s) = [\![\mathbf{E}'_M(s, f)]\!]_{e'}$
- If $f = f_1 \wedge f_2$: $[\![f]\!]_e(s) = [\![f_1]\!]_e(s) \sqcap [\![f_2]\!]_e(s) \overset{IH}{=} [\![\mathbf{E}'_M(s, f_1)]\!]_{e'} \sqcap [\![\mathbf{E}'_M(s, f_2)]\!]_{e'} = [\![\mathbf{E}'_M(s, f)]\!]_{e'}$
- If $f = f_1 \vee f_2$: $[\![f]\!]_e(s) = [\![f_1]\!]_e(s) \sqcup [\![f_2]\!]_e(s) \overset{IH}{=} [\![\mathbf{E}'_M(s, f_1)]\!]_{e'} \sqcup [\![\mathbf{E}'_M(s, f_2)]\!]_{e'} = [\![\mathbf{E}'_M(s, f)]\!]_{e'}$
- If $f = f_1 \cdot f_2$: $[\![f]\!]_e(s) = [\![f_1]\!]_e(s) \cdot [\![f_2]\!]_e(s) \overset{IH}{=} [\![\mathbf{E}'_M(s, f_1)]\!]_{e'} \cdot [\![\mathbf{E}'_M(s, f_2)]\!]_{e'} = [\![\mathbf{E}'_M(s, f)]\!]_{e'}$
- If $f = f_1 \odot f_2$: $[\![f]\!]_e(s) = [\![f_1]\!]_e(s) \odot [\![f_2]\!]_e(s) \overset{IH}{=} [\![\mathbf{E}'_M(s, f_1)]\!]_{e'} \odot [\![\mathbf{E}'_M(s, f_2)]\!]_{e'} = [\![\mathbf{E}'_M(s, f)]\!]_{e'}$
- If $f = f_1 \ominus f_2$: $[\![f]\!]_e(s) = [\![f_1]\!]_e(s) \ominus [\![f_2]\!]_e(s) \overset{IH}{=} [\![\mathbf{E}'_M(s, f_1)]\!]_{e'} \ominus [\![\mathbf{E}'_M(s, f_2)]\!]_{e'} = [\![\mathbf{E}'_M(s, f)]\!]_{e'}$
- If $f = f_1 \oplus f_2$: $[\![f]\!]_e(s) = [\![f_1]\!]_e(s) \oplus [\![f_2]\!]_e(s) \overset{IH}{=} [\![\mathbf{E}'_M(s, f_1)]\!]_{e'} \oplus [\![\mathbf{E}'_M(s, f_2)]\!]_{e'} = [\![\mathbf{E}'_M(s, f)]\!]_{e'}$

- If $f = f_1 +_\lambda f_2$: $[\![f]\!]_e(s) = [\![f_1]\!]_e(s) +_{\boldsymbol{\lambda}} [\![f_2]\!]_e(s) \overset{IH}{=} [\![\mathbf{E}'_M(s, f_1)]\!]_{e'} +_{\boldsymbol{\lambda}} [\![\mathbf{E}'_M(s, f_2)]\!]_{e'} = [\![\mathbf{E}'_M(s, f)]\!]_{e'}$
- If $f = \langle a \rangle f_1$: $[\![f]\!]_e(s) = \bigsqcup_{(s,a,\delta)\in T} \{ \sum_{s'\in S} \delta(s') \cdot [\![f_1]\!]_e(s') \} \overset{IH}{=} \bigsqcup_{(s,a,\delta)\in T} \{ \sum_{s'\in S} \delta(s') \cdot [\![\mathbf{E}'_M(s, f_1)]\!]_{e'} \} = [\![\mathbf{E}'_M(s, f)]\!]_{e'}$
- If $f = [a] f_1$: $[\![f]\!]_e(s) = \bigsqcap_{(s,a,\delta)\in T} \{ \sum_{s'\in S} \delta(s') \cdot [\![f_1]\!]_e(s') \} \overset{IH}{=} \bigsqcap_{(s,a,\delta)\in T} \{ \sum_{s'\in S} \delta(s') \cdot [\![\mathbf{E}'_M(s, f_1)]\!]_{e'} \} = [\![\mathbf{E}'_M(s, f)]\!]_{e'}$

□

To prove the correctness of the translation $\mathbf{E}_M$ we need to prove that for all bound variables $X \in Var$ in $\mathcal{E}_\mu$, states $s \in S$ and environments $e : Var \to (S \to [0, 1])$ and $e' : Var' \to [0, 1]$

$$([\mathcal{E}_\mu]e)(X)(s) = ([\mathbf{E}_M(\mathcal{E}_\mu)]e')(X_s)$$

if $e(Y)(t) = e'(Y_t)$ for all free variables $Y \in Var$ in $\mathcal{E}_\mu$ and all $t \in S$.
We will prove this using structural induction.
Base case: Since there are no bound variables in the empty equation system, it trivially holds that

$$([\epsilon]e)(X)(s) = ([\mathbf{E}_M(\epsilon)]e')(X_s)$$

Inductive step: Let all states in S be indexed by an $0 \leq i \leq n$ where $n = |S| - 1$. Then we need to prove for bound variables $X \in Var$ in $(\sigma Y = f)\ \mathcal{E}$ that

$$([(\sigma Y = f)\ \mathcal{E}]e)(X)(s_i) = ([\mathbf{E}_M((\sigma Y = f)\ \mathcal{E})]e')(X_{s_i})$$

if $e(Z)(t) = e'(Z_t)$ for all free variables $Z \in Var$ in $(\sigma Y = f)\ \mathcal{E}$ and all $t \in S$. The corresponding induction hypothesis (IH) is

$$([\mathcal{E}]e)(X')(s_i) = ([\mathbf{E}_M(\mathcal{E})]e')(X'_{s_i})$$

for all bound variables X' if $e(Z)(t) = e'(Z_t)$ for all free variables $Z \in Var$ in $\mathcal{E}$ and all $t \in S$.

Applying the semantics of the solution (definition 4.5) gives

$$([(\sigma Y = f)\mathcal{E}]e)(X)(s_i) = ([\mathcal{E}]e[Y := \sigma Y.f([\mathcal{E}]e)])(X)(s_i)$$

Now let $Proj$ be a projection function that given a tuple $(x_0, \ldots x_n)$ returns a function that maps state s_i to x_i for all i such that $Proj(x_0, \ldots x_n)(s_i) = x_i$. Then we vectorize the fixpoint formula $\sigma Y.f([\mathcal{E}]e)$, creating a variable of type $[0, 1]$ for every state, which results in:

$$\sigma Y.f([\mathcal{E}]e) = Proj(\sigma(Y_{s_0}, \ldots Y_{s_n}).(f([\mathcal{E}]e)(s_0), \cdots f([\mathcal{E}]e)(s_n)))$$

Note that with $f([\mathcal{E}]e)(s_i)$ we mean the same as $[\![f]\!]_{[\mathcal{E}]e}(s_i)$. Then to be able to apply lemma 7.1, we need to have that $([\mathcal{E}]e)(X)(s') = ([\mathbf{E}_M(\mathcal{E})]e')(X_{s'})$ for all

variables $X \in Var$ in f and all states $s' \in S$, which is true by the induction hypothesis. This means that we can apply lemma 7.1 to transform the above to

$$Proj(\sigma(Y_{s_0}, \dots Y_{s_n}).(\mathbf{E}'_M(s_0, f)([\mathbf{E}_M(\mathcal{E})]e'), \cdots \mathbf{E}'_M(s_n, f)([\mathbf{E}_M(\mathcal{E})]e')))$$

Applying theorem 2.3 will create fixpoint formulas $y_0, \dots y_n$ such that the above is equivalent to $Proj((y_0, \dots y_n))$. When we fill this in in the original problem we get:

$$([\mathcal{E}]e[Y := Proj((y_0, \dots y_n))])(X)(s_i)$$

Then to be able to apply the induction hypothesis, we need to find an environment e'' such that $e(Z)(t) = e''(Z_t)$ for all free variables $Z \in Var$ in $\mathcal{E}$ and all $t \in S$. By assumption, we already know this is the case for all free variables except Y if we pick e'' to be e'. To make this true for Y too, we adjust e' to $e'[Y_{s_0} := y_0, \dots Y_{s_n} := y_n]$. Then applying the induction hypothesis with this adjusted e' we get

$$([\mathbf{E}_M(\mathcal{E})]e'[Y_{s_0} := y_0, \dots Y_{s_n} := y_n])(X)(s_i)$$

which is equivalent to the desired $([\mathbf{E}_M((\sigma Y = f)\mathcal{E})]e')(X_{s_i})$. Since it is very tedious work to show this last step in the general case, we will show how it works using an example:

Example 7.1. Let $S = \{s_0, s_1\}$. For simplicity of notation, let $f'_i = \mathbf{E}'_M(s_i, f)$. Then

$$\begin{aligned} &([\mathcal{E}]e[Y := Proj(\sigma(Y_{s_0}, Y_{s_1}).(f'_0([\mathbf{E}_M(\mathcal{E})]e'), f'_1([\mathbf{E}_M(\mathcal{E})]e')))])(X)(s_i) \\ \{Thrm\ 2.3\} &= ([\mathcal{E}]e[Y := Proj((y_0, y_1))])(X)(s_i) \end{aligned}$$

where

$$\begin{aligned} y_1 &= \sigma Y_{s_1}.f'_1([\mathbf{E}_M(\mathcal{E})]e'[Y_{s_0} := \sigma Y_{s_0}.f'_0([\mathbf{E}_M(\mathcal{E})]e')]) \\ \{Def\ 4.5\} &= \sigma Y_{s_1}.f'_1([(\sigma Y_{s_0} = f'_0)\ \mathbf{E}_M(\mathcal{E})]e') \end{aligned}$$

$$\begin{aligned} y_0 &= \sigma Y_{s_0}.f'_0([\mathbf{E}_M(\mathcal{E})]e'[Y_{s_1} := y_1]) \\ &= \sigma Y_{s_0}.f'_0([\mathbf{E}_M(\mathcal{E})]e'[Y_{s_1} := \sigma Y_{s_1}.f'_1([(\sigma Y_{s_0} = f'_0)\ \mathbf{E}_M(\mathcal{E})]e')]) \end{aligned}$$

Then applying the induction hypothesis gives

$$\begin{aligned} &([\mathbf{E}_M(\mathcal{E})]e[Y_{s_0} := y_0, Y_{s_1} := y_1])(X)(s_i) \\ \{Def\ 4.5\} &= ([(\sigma Y_{s_0} = f'_0)\ \mathbf{E}_M(\mathcal{E})]e[Y_{s_1} := y_1])(X)(s_i) \\ \{Def\ 4.5\} &= ([(\sigma Y_{s_1} = f'_1)\ (\sigma Y_{s_0} = f'_0)\ \mathbf{E}_M(\mathcal{E})]e)(X)(s_i) \\ \{Lemma\ 4.5\} &= ([(\sigma Y_{s_0} = f'_0)\ (\sigma Y_{s_1} = f'_1)\ \mathbf{E}_M(\mathcal{E})]e)(X)(s_i) \\ &= ([\mathbf{E}_M((\sigma Y = f)\ \mathcal{E})]e)(X)(s_i) \end{aligned}$$

which is the desired result.

Since we have proven correctness of both translations, we know that $[\![\sigma X.\phi]\!]_e(s) = ([\mathbf{E}_M(\mathbf{E}_\mu(\sigma X.\phi))]e)(X_s)$. Since $\mathbf{E}_M(\mathbf{E}_\mu(\sigma X.\phi)) = \mathbf{E}_R(\sigma X.\phi)$, we can therefore conclude that $[\![\sigma X.\phi]\!]_e(s) = ([\mathbf{E}_R(\sigma X.\phi)]e')(X_s)$. □

Since closed $pL\mu^{\odot}_{\oplus}$-formulas do not contain any free variables, we can simplify the above theorem for such $pL\mu^{\odot}_{\oplus}$-formulas.

Corollary 7.1. Let $\langle S, s_0, A, T\rangle$ be a PLTS and $\sigma X.\phi$ be a closed $pL\mu^{\odot}_{\oplus}$-formula. Then for any variable $X \in Var$, state $s \in S$ and environments $e : Var \to (S \to [0,1])$ and $e' : Var' \to [0,1]$

$$[\![\sigma X.\phi]\!]_e(s) = ([\mathbf{E}_R(\sigma X.\phi)]e')(X_s)$$

7.2 Solving a RES$^{\odot}_{\oplus}$

As described in [Mad97] and in section 4.2.1, a BES can be solved by means of an algorithm similar to Gauss elimination. We can apply the same method for RES$^{\odot}_{\oplus}$'s, but with a solving step instead of an elimination step. See algorithm 3 for the pseudo code given a RES$^{\odot}_{\oplus}$ $\mathcal{E} = (\sigma X_1 = f_1)$ ... $(\sigma X_n = f_n)$.

Algorithm 3 *SolveRES*($\mathcal{E}$)

```
1: for i := n downto 0 do
2:    solve X_i = f_i for X_i
3:    for j := 0 to i − 1 do
4:       f_j = f_j[X_i := f_i]
5:    end for
6: end for
```

The correctness of substitution in *SolveRES* follows from the correctness of substitution in *GaussElimination* for BESs (algorithm 1) as described in [Mad97] as lemma 6.3, since the corresponding proof does not rely on the lattice used. We will give this lemma and its proof below

Lemma 7.2. Let $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{E}_3$ be RES$^{\odot}_{\oplus}$'s, let $\sigma_1 X_1 = f$, $\sigma_1 X_1 = f'$ and $\sigma_2 X_2 = g$ be real equations such that $f' = f[X_2 := g]$ and let e be some environment. Then

$$[\mathcal{E}_1\ (\sigma_1 X_1 = f)\ \mathcal{E}_2\ (\sigma_2 X_2 = g)\ \mathcal{E}_3]e = [\mathcal{E}_1\ (\sigma_1 X_1 = f')\ \mathcal{E}_2\ (\sigma_2 X_2 = g)\ \mathcal{E}_3]e$$

Proof. Using lemma 4.4 it is only necessary to check that

$$[(\sigma_1 X_1 = f)\ \mathcal{E}_2\ (\sigma_2 X_2 = g)\ \mathcal{E}_3]e = [(\sigma_1 X_1 = f')\ \mathcal{E}_2\ (\sigma_2 X_2 = g)\ \mathcal{E}_3]e$$

We define

$$[\mathcal{E}]e = [(\sigma_1 X_1 = f)\ \mathcal{E}_2\ (\sigma_2 X_2 = g)\ \mathcal{E}_3]e = e_1$$
$$[\mathcal{E}']e = [(\sigma_1 X_1 = f')\ \mathcal{E}_2\ (\sigma_2 X_2 = g)\ \mathcal{E}_3]e = e_2$$

We will first show that e_1 fulfills both conditions of lemma 4.2 for the solution of $\mathcal{E}'$ to show that $e_1 \leq_{\mathcal{E}'} e_2$ since e_2 is the solution of $\mathcal{E}'$.
Condition 1: show that $f'(e_1) = e_1(X_1)$. By definition of the solution we already know that $f(e_1) = e_1(X_1)$ and $g(e_1) = e_1(X_2)$. Using this we can derive that

$$\begin{aligned} e_1(X_1) &= f(e_1) \\ &= f(e_1[X_2 := e_1(X_2)]) \\ &= f(e_1[X_2 := g(e_1)]) \\ &= f'(e_1) \end{aligned}$$

Condition 2: show that $[\mathcal{E}_2\ (\sigma_2 X_2 = g)\ \mathcal{E}_3]e_1 = e_1$. This follows from lemma 4.3 using that $[\mathcal{E}]e = e_1$.
In the same way we can prove that e_2 fulfills both conditions of lemma 4.2 for the solution of $\mathcal{E}$, from which it follows that $e_2 \leq_{\mathcal{E}} e_1$. Since $\mathcal{E}$ and $\mathcal{E}'$ have the same sequence of fixpoint signs, we can say that $e_2 \leq_{\mathcal{E}} e_1$ iff $e_2 \leq_{\mathcal{E}'} e_1$ by definition 4.6.
Since we have shown that both $e_1 \leq_{\mathcal{E}'} e_2$ and $e_2 \leq_{\mathcal{E}'} e_1$, we can conclude that $e_1 = e_2$. □

With solving $X = f$ for X we mean that we want to find a function f' such that $X = f'$ and f' does not contain X. If f already did not contain X, we simply have that $f' = f$. If f does contain X, we need to do some additional steps. For this case we will discuss two possibilities.

The straightforward way, directly taken from the definition of a $\text{RES}^{\odot}_{\oplus}$, is by fixpoint iteration. Let $\sigma X = f$, then we can "solve" it by computing $\sigma X.f(X)$ iteratively. However, as was the case in the naive algorithm, this will not always terminate due to the infinite size of the interval $[0, 1]$ of real numbers. For instance, while computing the solution of the equation $\nu X = X +_{\frac{1}{2}} \underline{0}$ the approximated solution will be halved every iteration, starting at 1, never actually reaching the fixpoint 0. The iterative method is also very inefficient, since the syntactic representation of the approximands may grow every iteration due to variables that remain in the solution.

However, since f is a function on $[0, 1]$, one can see f as a function on the field of reals. Then computing the fixpoints of f is the same as finding all intersections of f with the line $f(X) = X$. This is done by solving the equation $X = f(X)$ for X. By theorem 2.1, the collection of intersections is a complete lattice. In case $\sigma = \mu$, the desired solution of $X = f(X)$ is the bottom of this lattice of intersections, otherwise it is the top.
Actually solving the equation $X = f(X)$ is however not always so trivial, especially when the operators $\cdot$, $\odot$, $\ominus$ and $\oplus$ are involved. Below we will discuss solving methods for sub-equation systems of a $\text{RES}^{\odot}_{\oplus}$:

- RES which is a $\text{RES}^{\odot}_{\oplus}$ without the operators $\cdot$, $\odot$, $\ominus$ and $\oplus$
- $\text{RES}^{\odot}$ which is a $\text{RES}^{\odot}_{\oplus}$ without the operators $\ominus$ and $\oplus$
- $\text{RES}_{\oplus}$ which is a $\text{RES}^{\odot}_{\oplus}$ without the operators $\cdot$ and $\odot$

Note that translating a $pL\mu$-formula results in a RES, a $pL\mu^{\odot}$-formula in a $\text{RES}^{\odot}$ and a $pL\mu_{\oplus}$-formula in a $\text{RES}_{\oplus}$.
The solving method for $\text{RES}^{\odot}_{\oplus}$'s is a combination of the described methods.

Solving a RES-equation

To solve $X = f(X)$, rewrite each normal formula to a normal form.

Lemma 7.3. Let $\mathcal{E}$ be a RES. Then for each equation $(\sigma X = f)$ in $\mathcal{E}$ the real formula f can be rewritten to the following normal form:

$$\bigsqcup_i \{ \bigsqcap_j \{ \sum_k p_{ijk} \cdot X_{ijk} + p_{ij} \} \}$$

Proof. We will prove this using structural induction on f.
Base case: If $f = \lambda$ for some $\lambda \in [0,1]$ or if $f = X$ for some $x \in Var$ then f is in normal form.
Inductive step assuming that f_1 and f_2 are real formulas in normal form:

- If $f = f_1 \sqcup f_2$ then f is in normal form.
- If $f = f_1 \sqcap f_2$ then by the induction hypothesis $f_1 = \bigsqcup_i f_i$ and $f_2 = \bigsqcup_j f_j$. Then by distributivity of $\sqcap$ over $\sqcup$ we can change f to $\bigsqcup_i \bigsqcup_j (f_i \sqcap f_j)$, which is in normal form.
- If $f = f_1 +_{\boldsymbol{\lambda}} f_2$ then by the induction hypothesis $f_1 = \bigsqcup_i \bigsqcap_j f_{ij}$ and $f_2 = \bigsqcup_k \bigsqcap_l f_{kl}$. Then by lemma 2.4 we can change f to $\bigsqcup_i \bigsqcup_k \bigsqcap_j \bigsqcap_l (f_{ij} +_{\boldsymbol{\lambda}} f_{kl})$ which is in normal form.

□

In practice we may also consider the possibility that the multiary operators $\bigsqcup$ and $\bigsqcap$ are unary, which allows normal forms from which these operators can be omitted.
As mentioned before, the solution of X in the equation $\sigma X = f$ equals $\sigma X.f(X)$. If we first rewrite f to normal form, we can use lemmas 2.3a, 2.3b and 2.5 to conclude that

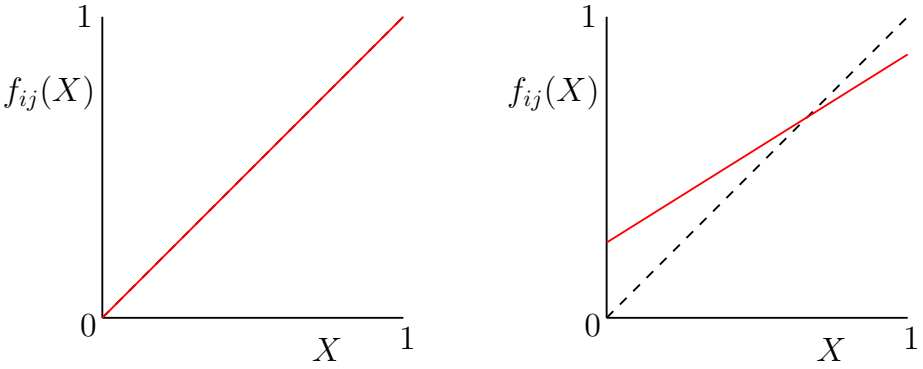
$$\sigma X.f(X) = \bigsqcup_i \{\bigsqcap_j \{\sigma X.f_{ij}(X)\}\}$$

where all f_{ij} are of the form

$$\sum_k p_{ijk} \cdot X_{ijk} + p_{ij}.$$

where $\sum_k p_{ijk} + p_{ij} \leq 1$ for all i and j.
The reason that we can use lemma 2.5 is as follows. This lemma requires functions f_1 and f_2 in $\mu X.(f_1(X) \sqcup f_2(X))$ and $\nu X.(f_1(X) \sqcap f_2(X))$ to have intervals as prefixpoints and postfixpoints such that both prefixpoints intervals overlap and both postfixpoints intervals overlap. In the case where it is used above the functions f_1 and f_2 are either of the form $\sum_k p_{ijk} \cdot X_{ijk} + p_{ij}$ or $\bigsqcap_j \{\sum_k p_{ijk} \cdot X_{ijk} + p_{ij}\}$. For both forms we can use the same argument. Both f_1 and f_2 are total and they only intersect the fixpoint line ($f(X) = X$) once. Due to this, the prefixpoints are intervals that always start at 0 and the postfixpoints are intervals that always end at 1 and are therefore sure to overlap.
Now all that is left to get the solution of X is to solve $\sigma X.f_{ij}(X)$ for every i and j. We differentiate between two cases, which are visualized in figure 14:

- $f_{ij}(X) = X$: In this case every $p \in [0,1]$ is a fixpoint. Therefore if $\sigma = \mu$ the solution equals 0, else if $\sigma = \nu$ the solution equals 1.



(a) Case $f_{ij}(X) = X$ (b) Case $f_{ij}(X) \neq X$

Figure 14: The two cases for $f_{ij}(X)$. The dashed line is the line of all fixpoints.

- $f_{ij}(X) \neq X$: Since $f_{ij}(X)$ is linear only one fixpoint can exist. If $f_{ij}(X)$ does not contain X, the solution is simply $f_{ij}(X)$. If $f_{ij}(X)$ does contain X we can solve $X = f_{ij}(X)$ using simple linear algebra. Let $f_{ij}(X) = \sum_k p_{ijk} \cdot Y_{ijk} + p \cdot X + p_{ij}$, then the solution f'_{ij} equals $\sum_k \frac{p_{ijk}}{1-p} \cdot Y_{ijk} + \frac{p_{ij}}{1-p}$. After resolving the fractions the solution f'_{ij} is in the same form as f_{ij}.

Now we can replace every f_{ij} with its solution f'_{ij} resulting in

$$\bigsqcup_i \{ \bigsqcap_j \{\sigma X. f_{ij}(X)\}\} = \bigsqcup_i \{ \bigsqcap_j \{f'_{ij}\}\}$$

which is the solution of the original equation $\sigma X = f$.

Solving a RES$^{\odot}$-equation

If we want to create a normal form while the operators $\cdot$ and $\odot$ are included we need to flatten them according to their definitions, since both operators do not distribute over one another. If we do this, the right hand side of an equation may contain products of variables. This may cause the solution of this equation to be a fraction with variables in the denominator. Since this solution will be substituted, we need to be able to solve equations with variables in the denominator as well. However, the solution(s) of such an equation may contain square roots, so similarly we need to be able to solve square roots with variables inside as well. Now already the problem arises that most operators on real numbers do not distribute over the square root or vice versa, so it is not possible to come up with a normal form. Other than that, it is not clear whether the conclusion of lemma 2.5 also holds in this situation.
Therefore, the solving method for RES's cannot be used when considering the operators $\cdot$ and $\odot$. The solution of an equation does still exist, but to find it one might need external equation solving libraries. Even then, the definition of a RES$^{\odot}$ has to be altered to allow operators as division and square root for the equations system to remain a well-defined RES$^{\odot}$ during execution of the *SolveRES* algorithm.

Solving a RES$_\oplus$-equation

If we want to create a normal form while the operators $\ominus$ and $\oplus$ are included we need to flatten them according to their definitions too, since both operators do not distribute over one another. But then a different problem arises. If we would create a normal form of the form $\bigsqcup\bigsqcap f_{ij}$ like we did for RES's, the range of the formulas f_{ij} may not be $[0,1]$. One application of the operator $\ominus$ results in a $f_{ij} = f_1 + f_2 - 1$, which has the range $[-1,1]$. Similarly, one application of the operator $\oplus$ results in a $f_{ij} = f_1 + f_2$, which has the range $[0,2]$. Multiple applications of these operators will create f_{ij}'s with further increased ranges. These subformulas f_{ij} with increased ranges may not have a fixpoint in the domain $[0,1]$.
It could even be that a f_{ij} does not have a fixpoint at all. For instance the equation $\mu X = X \oplus f = (X + f) \sqcap 1$ would result in solving $\mu X.(X + f)$, but this only has a solution if $f = 0$.
Therefore, the solving method described for RES's cannot be used when considering the operators $\ominus$ and $\oplus$. The solution of an equation does still exist, but again one might need to use external equation solving libraries to find it.

Worst case running time complexity

The worst case running time complexity of *SolveRES* is similar to the worst case running time complexity of Gauss-elimination for BESs. Every equation $\sigma X = f$ needs to be solved only once, which for RES's can be done in polynomial time in the size of f when using the method explained above. A substitution can be done in constant time, but substitutions will make real formulas grow exponentially in size in the worst case. Therefore, the number of substitutions may be exponential in the size of the RES$^{\odot}_{\oplus}$, which causes the whole *SolveRES* algorithm to be exponential in the worst case.

However, like with Gauss-elimination, we can improve the absolute running time of *SolveRES* by applying simplifications to the real formulas to make them smaller. Such simplifications can be done after bringing a formula to normal form and after substituting. Possible simplifications are for instance:

- Resolve zero values, such as $0 \cdot f = 0$.
- Resolve unit values, such as $0 + f = f$ and $1 \cdot f = f$.
- Apply operators, for instance $\frac{1}{2} \cdot \frac{1}{4} = \frac{1}{8}$ or $\frac{1}{2}X + \frac{1}{4}X = \frac{3}{4}X$.
- Remove terms from min and max that are definitely worse, for instance $\min(\frac{1}{2}X + \frac{1}{3}, \frac{1}{4}X + \frac{1}{3}) = \frac{1}{4}X + \frac{1}{3}$.

The latter is done as follows. Let $f_1 = \sum a_i X_i + \sum b_j Y_j + c$ and $f_2 = \sum d_i X_i + \sum e_j Z_j + f$ be real formulas, where the X_i represent the variables f_1 and f_2 have in common and the Y_j and Z_j represent variables f_1 and f_2 do not have in common. Then f_1 is definitely less than f_2 if both $a_i \leq d_i$ for all i and $\sum b_j + c \leq f$.

See below for an example of using this equation system approach to solve a model checking problem.

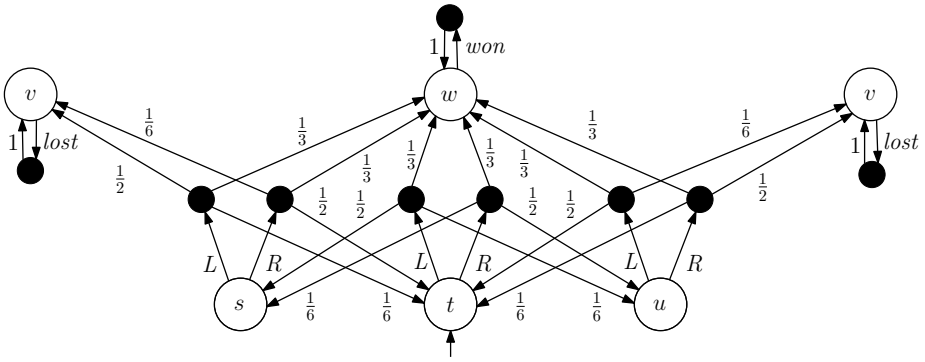


Figure 15: The PLTS of a 3x1 board game as described in section 6.4.3. The actions *moveLeft* and *moveRight* are abbreviated to L and R respectively. Note that the state u occurs twice. These are the same state, but split into two to make it visually more pleasing.

Example 7.2. As an example for the creation and solving of a RES we will pick the board game example as described in section 6.4.3, but then with a 3x1 board. The corresponding PLTS is shown in figure 15, which we will refer to as $\mathcal{M}$. Again we want to check the formula $\phi = \mu X.((\langle moveLeft\rangle X \vee \langle moveRight\rangle X) \vee \langle won\rangle \underline{1})$, which checks what the probability is of winning the game when playing optimally.
When we apply the translation $\mathbf{E}_R$ as in definition 7.2 to formula ϕ and model $\mathcal{M}$, we get the following RES after applying the simplifications described above:

$$\begin{aligned}
\mu X_s &= \max(\frac{1}{6}X_t + \frac{1}{2}X_v + \frac{1}{3}X_w, \frac{1}{2}X_t + \frac{1}{6}X_v + \frac{1}{3}X_w)\\
\mu X_t &= \max(\frac{1}{2}X_s + \frac{1}{6}X_u + \frac{1}{3}X_w, \frac{1}{6}X_s + \frac{1}{2}X_u + \frac{1}{3}X_w)\\
\mu X_u &= \max(\frac{1}{2}X_t + \frac{1}{6}X_v + \frac{1}{3}X_w, \frac{1}{6}X_t + \frac{1}{2}X_v + \frac{1}{3}X_w)\\
\mu X_v &= 0\\
\mu X_w &= 1
\end{aligned}$$

Now we can extract the solution by applying the *SolveRES* algorithm as described in algorithm 3 combined with the solving method for RES's. After each solving and substitution step we will apply the simplifications described above. The equation of X_w does not contain X_w on the right-hand side, so we only need to do the distribution step. The same can be said for X_v. After substituting both we get the following RES:

$$\begin{aligned}
\mu X_s &= \frac{1}{2}X_t + \frac{1}{3}\\
\mu X_t &= \max(\frac{1}{2}X_s + \frac{1}{6}X_u + \frac{1}{3}, \frac{1}{6}X_s + \frac{1}{2}X_u + \frac{1}{3})\\
\mu X_u &= \frac{1}{2}X_t + \frac{1}{3}\\
\mu X_v &= 0\\
\mu X_w &= 1
\end{aligned}$$

Likewise, the right-hand side of the equation of X_u does not contain X_u, so we only need to do the can substitute upwards resulting in the following RES:

$$\begin{aligned}
\mu X_s &= \frac{1}{2}X_t + \frac{1}{3} \\
\mu X_t &= \max(\frac{1}{2}X_s + \frac{1}{12}X_t + \frac{7}{18}, \frac{1}{6}X_s + \frac{1}{4}X_t + \frac{1}{2}) \\
\mu X_u &= \frac{1}{2}X_t + \frac{1}{3} \\
\mu X_v &= 0 \\
\mu X_w &= 1
\end{aligned}$$

The right-hand side of the equation for X_t does contain X_t itself, so we will need to solve it. Using the solving method for $pL\mu$, we need to solve the equations

$$\begin{aligned}
X_t = \frac{1}{2}X_s + \frac{1}{12}X_t + \frac{7}{18} \text{ which results in } X_t = \frac{6}{11}X_s + \frac{14}{33} \\
X_t = \frac{1}{6}X_s + \frac{1}{4}X_t + \frac{1}{2} \text{ which results in } X_t = \frac{2}{9}X_s + \frac{2}{3}
\end{aligned}$$

Substituting these solutions in the equation gives $\mu X_t = \max(\frac{6}{11}X_s + \frac{14}{33}, \frac{2}{9}X_s + \frac{2}{3})$. Then after the substitution step we get the following RES:

$$\begin{aligned}
\mu X_s &= \max(\frac{3}{11}X_s + \frac{18}{33}, \frac{1}{9}X_s + \frac{2}{3}) \\
\mu X_t &= \max(\frac{6}{11}X_s + \frac{14}{33}, \frac{2}{9}X_s + \frac{2}{3}) \\
\mu X_u &= \frac{1}{2}X_t + \frac{1}{3} \\
\mu X_v &= 0 \\
\mu X_w &= 1
\end{aligned}$$

Lastly, we need to solve the equation for X_s. Again using the solving method for $pL\mu$, we have to solve the equations

$$\begin{aligned}
X_s = \frac{3}{11}X_s + \frac{18}{33} \text{ which results in } X_s = \frac{3}{4} \\
X_s = \frac{1}{9}X_s + \frac{2}{3} \text{ which results in } X_s = \frac{3}{4}
\end{aligned}$$

Substituting these solutions back in the equation gives $\mu X_s = \frac{3}{4}$. Note however that state s in the given PLTS $\mathcal{M}$ is actually not the initial state. To get the value for the initial state t, we can substitute the solution of X_s downwards. This is allowed since this solution does not contain any variables. If we would do this iteratively from top to bottom for every variable we will end up with the following RES:

$$\begin{aligned}
\mu X_s &= \frac{3}{4} \\
\mu X_t &= \frac{5}{6} \\
\mu X_u &= \frac{3}{4} \\
\mu X_v &= 0 \\
\mu X_w &= 1
\end{aligned}$$

which gives the solution of ϕ for every state. Since t is the initial state, we can conclude that the probability of winning the game on a 3x1 board equals $\frac{5}{6}$.

8 Experiments and improvements

In this section we will evaluate the performance of the $\text{RES}^{\odot}_{\oplus}$ solving algorithm *SolveRES* (algorithm 3 in section 7.2) by applying it to some use cases and reflect on these results and the algorithm itself. Afterwards we will try out some changes to the *SolveRES* algorithm to improve the running time. Lastly we will shortly experiment with a small addition to the logic to let the algorithm be able to return more than a probability.
Most models will be represented in the mCRL2 format [GM14]. The mCRL2 models can be found in appendix A and using the mCRL2 tool-set [CGK+13] they can be transformed to PLTSs. Note that some actions in these models carry data parameters. If such an action appears in a $pL\mu^{\odot}_{\oplus}$-formula without parameters, we mean to check for this action with any values for its parameters.
We will apply and analyze both the naive (algorithm 2) and the *SolveRES* algorithm (algorithm 3), which have been implemented in Python. The source code of the implementation can be found at `https://github.com/Valo13/plmuChecker`. The naive algorithm is set to have an unlimited number of iterations so that it stops when the difference between two iterations is less than Python is able to represent with a double. Therefore the resulting value of the naive algorithm may be an approximation, although with a difference to the exact result that is not respresentable. For the *SolveRES* algorithm we will only consider $pL\mu$-formulas, since we have not given a complete solving method when the operators $\cdot$, $\odot$, $\ominus$ and/or $\oplus$ are involved.
The timing results will be given in seconds unless appended by a letter (m for minutes, h for hours) and rounded to a significance of two. Every timing is the average of 10 runs, except when the timing surpasses 1 hour in which case it is only run once. The timings of the RES approach are split into two timings: the running time of the creation of the RES denoted by *CreateRES* and the running time of the *SolveRES* algorithm.
The resulting values are taken from the result of the *SolveRES* algorithm. They are rounded to a significance of three, except when the answer can be represented exactly in fewer digits.
The experiments are run on a PC running Windows 10 with a i7-7700K (4.2GHz) processor and 16 GB DDR4 RAM.

8.1 Use cases

Below we introduce five use cases and show the results of applying the naive and the *SolveRES* algorithm to check meaningful $pL\mu$-formulas on these use cases.

Ant on a grid

The ant on a grid problem [Ant13] is as follows. We have a grid of 8 by 8 lines, with an ant located on the intersection of the third horizontal and fifth vertical line. The ant takes a step to a neighbouring intersection (either up, down left or right) where all possibilities have equal probability ($\frac{1}{4}$). If the ant moves to a horizontal boundary line it survives, if the ant moves to a vertical boundary line it dies. Then the question is "What is the probability that the ant will survive?". See figure 16 for a visualization of the starting point of the ant on grid problem.

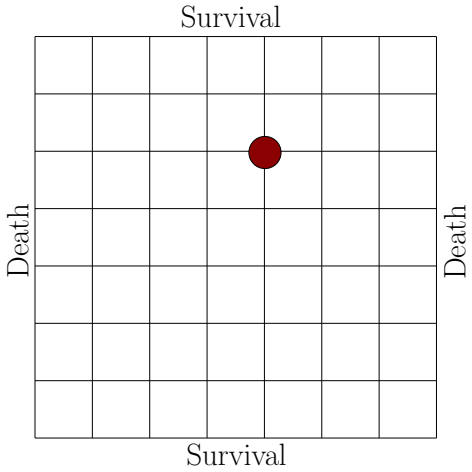


Figure 16: The starting point of the ant on grid problem where the ant is denoted with a red dot. Note that it actually does not matter whether the ant starts from the third horizontal line from the top or bottom and from the fifth vertical line from the left or right.

The mCRL2 model of this problem is taken from the mCRL2 tool-set and is given in appendix A.1. The action *step* models the ant making a step, the action *live* models that the ant has survived and the action *dead* models that the ant has died. The question "What is the probability that the ant will survive?" can be answered by the formula $\phi_s = \mu X.(\langle step \rangle X \vee \langle live \rangle \underline{1})$. Similarly the question "What is the probability that the ant will die?" can be answered with the formula $\phi_d = \mu X.(\langle step \rangle X \vee \langle dead \rangle \underline{1})$. The probability of the remaining option for the ant, namely never reaching a boundary line, can be answered with the formula $\phi_i = \nu X.([step]X \wedge [live]\underline{0} \wedge [dead]\underline{0})$. See the table below for the timings and resulting values.

	Naive	*CreateRES*	*SolveRES*	Result
ϕ_s	0.20	0.024	2.2	0.586
ϕ_d	0.20	0.023	2.2	0.414
ϕ_i	0.77	0.026	2.2	0

As one can see from the timings, the *SolveRES* algorithm is quite a lot slower than the naive algorithm, which is something that we will see more often. Another interesting thing to note is that while the naive algorithm has a greater running time for ϕ_i compared to the other formulas, The *SolveRES* algorithm does not have this difference at all (except slightly in *CreateRES*). This shows that the naive algorithm's running time heavily depends on the size of the formula, whereas this is not the case for *SolveRES*. The latter can be explained by the fact that one can only do one action in a state, which makes it possible to simplify the semantics of the box operators of the other actions already during *CreateRES*.
The probability for the ant to survive is greater than the probability to die which makes sense when looking at the initial position: the ant is closer to a boundary where it survives than to a boundary where it dies while the probability to move

in any of the four directions is uniformly distributed. The probability for the ant to reach no border equals 0. This is due to the fact that the steps are made probabilistically and that all traces that do not reach a border are infinite. The probability of any such trace happening is $(\frac{1}{4})^\infty = 0$ that is, it will almost never happen that the ant will not end up at a border.
The dual of ϕ_i, the probability that the ant will reach any border, can also be created using ϕ_s and ϕ_d by saying that ϕ_s or ϕ_d should happen. Since surviving and dying are two mutually exclusive events this combination should be made with the $\oplus$ operator, resulting in $\overline{\phi_i} \equiv \phi_s \oplus \phi_d$.

Airplane seats

In the airplane seats problem, a number N of people need to board a plane with N seats. The first passenger however has forgotten his seat number, so he/she decides to sit on a random seat. The following passengers, one by one, will sit on their own seat, but if it is already taken they will sit on a random seat instead. Now the question is "What is the probability that the last passenger will sit on his own seat?".
The mCRL2 model of this problem is taken from the mCRL2 tool-set and is given in appendix A.2. The action *enter* models arriving at the plane, the action *enter_plane* models entering the plane and picking a seat and the action *last_passenger_has_his_own_seat* shows whether the last passenger has his own seat using a boolean argument. The $pL\mu$-formula that answers the above question is

$$\mu X.(\langle enter \rangle X \vee \langle enter\_plane \rangle X \vee \langle last\_passenger\_has\_his\_own\_seat(true) \rangle \underline{1})$$

See the table below for the size of the PLTS $\langle S, s_0, A, T \rangle$, the timings and the resulting value for different values of N.

N	$\|S\|$	$\|T\|$	Naive	*CreateRES*	*SolveRES*	Result
2	8	7	0.00014	0.00038	0.00041	0.5
5	32	31	0.0013	0.0020	0.0083	0.5
10	72	71	0.0056	0.0049	0.039	0.5
25	192	191	0.036	0.013	0.27	0.5
50	392	391	0.15	0.027	1.1	0.5
75	592	591	0.33	0.041	2.5	0.5
100	792	791	0.59	0.055	4.5	0.5

The generated PLTS is a linear acyclic chain where every shackle is a passenger arriving and picking a seat. This linearity can be seen in the number of states and transitions shown in the above table where $|S|, |T| = \Theta(N)$ ($|S|$ and $|T|$ are linear in N) and $|T| = |S| - 1$ (one transition for each state except the final state). Note that due to these properties, the naive algorithm will terminate within $\Theta(|S|)$ iterations with an exact result. Also, as we order the states from start to end, the *SolveRES* algorithm will only substitute solutions without variables. Because of this no equation will grow during solving, which is very beneficial for the running time.
The running times of the two algorithms are also shown as plots in figure 17. This data shows that although the sizes of the PLTSs are linear in N, the running time complexity of both algorithms on these PLTSs seems to be quadratic

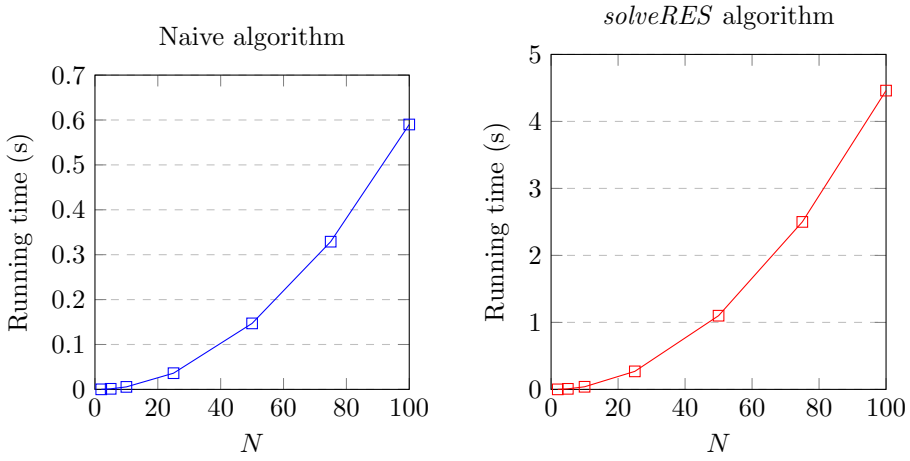


Figure 17: Running times of the naive and the *SolveRES* algorithm on the airplane seats use case for different number of passengers N.

in N. For the naive algorithm this is because every iteration a new value for X is calculated for each state. For the *SolveRES* algorithm this is likely due to the substitution step that for each solved equation tries to substitute the result in all equations above. The running time of *CreateRES* seems to behave linearly in N.
Note that the resulting probability is independent of N: the probability that the last passenger will sit on his own seat is independent of the number of passengers that need to board the plane and it equals 0.5.

Board game

We had already defined the board game example in section 6.4.3, but we will repeat it here again.
The game is played on a WxH board of squares. One starts at the bottom row on the middle square and from each square one can choose to move left or right. When moving left, with probability $\frac{1}{2}$ one will actually move to the square on the left, with probability $\frac{1}{3}$ to the square in front and otherwise to the square to the right. Moving to the right is symmetric. If one moves off the board on the side the game is lost and if one moves off the board on the top the game is won. Note the similarity to the ant on grid problem. In the board game example however the player has a choice that influences the result. See figure 9 in section 6.4.3 for a board where $W = H = 3$.
See figures 10 (section 6.4.3) and 15 (section 7.2) for some example PLTSs that model this board game. The action *moveLeft* models moving left, the action *moveRight* models moving right, the action *won* models winning the game and the action *lost* models losing the game. The states are ordered from left to right, bottom to top except that the losing and the winning state are the second last and last state respectively.
We will consider two $pL\mu$-formulas. The formula

$$\phi_w = \mu X.(\langle moveLeft \rangle X \vee \langle moveRight \rangle X \vee \langle won \rangle \underline{1})$$

answers "What is the probability to win when choosing optimally?" and the formula

$$\phi_i = \nu X.([\mathit{moveLeft}]X \wedge [\mathit{moveRight}]X \wedge \langle \mathit{moveRight} \rangle \underline{1})$$

answers "What is the probability to be always able to do a step when choosing optimally?". Note that in the latter formula the part $\langle \mathit{moveRight} \rangle \underline{1}$ is enough to test whether a step is possible, since if the action *moveLeft* is possible the action *moveRight* is possible as well (and vice versa). See the tables below for the timings and resulting values for different W and H for the formulas ϕ_w and ϕ_i respectively.

W	H	Naive	*CreateRES*	*SolveRES*	Result
1	1	0.000040	0.00032	0.000088	0.333
1	3	0.00012	0.000089	0.00031	0.0370
1	5	0.00024	0.0015	0.00056	0.00412
3	1	0.0018	0.0013	0.0028	0.833
5	1	0.0033	0.0022	0.014	0.968
3	3	0.0053	0.0039	0.020	0.528
5	5	0.019	0.010	2.7	0.771

W	H	Naive	*CreateRES*	*SolveRES*	Result
1	1	0.000040	0.00037	0.000076	0
1	3	0.00015	0.00093	0.00029	0
1	5	0.00027	0.0016	0.00054	0
3	1	0.020	0.0013	0.0018	0
5	1	0.042	0.0022	0.0044	0
3	3	0.053	0.0040	0.010	0
5	5	0.19	0.011	0.079	0

From the timing results we can see a few interesting things. Although the PLTSs for the vertical (1x3, 1x5) and horizontal (3x1, 5x1) boards have the same amount of states (respectively), solving the above formulas for a vertical board is around 8 times faster than for a horizontal board. This has to do with the transitions between the states: In case of a vertical board the states are connected linearly, while the horizontal board has transitions back and forth between the states, allowing cyclical behaviour. These extra dependencies cause the naive algorithm to need more iterations and the *SolveRES* algorithm to create bigger equations while substituting. Note that for ϕ_i with the naive algorithm it is even around 100 times slower. This however mainly has to do with how doubles are represented in Python and with the resulting value. When the program is approaching 0 it represents the value in the form $x \cdot 10^y$, which makes it possible to show even smaller changes than when approaching 0.5 for instance. As example, for the 5x1 board the value of X in the initial state in the second last iteration equals $5 \cdot 10^{-324}$.
Due to this the *SolveRES* algorithm is faster than the naive algorithm. However when the board becomes bigger, as can be seen for the 5x5 board, the running time of the *SolveRES* algorithm increases faster than the running time of the naive algorithm likely due to the exponential worst case running time of the *SolveRES* algorithm.

In the resulting values one can see what effect it has to be able to choose what direction to go. For the 3x3 and the 5x5 board the starting position is closer to the losing sides than to the winning side. Also, when making a move there is a higher probability to move towards a losing side than to the winning side. Nevertheless, the probability of winning is greater than 0.5, because by choosing the optimal moves, which are to move towards the losing side that is furthest away, the player can try to stay away from the losing sides as much as possible.

Yahtzee

Another type of game we will look into is a simplified version of the yahtzee game. This yahtzee game is played with three three-sided dice. A basic turn consist of two steps: throw the dice and write down a score. One can choose three options when writing down a score, one for each possible outcome of a die (1, 2 or 3), where the value of the score is the sum of the dice of the chosen value. For instance, if one throws $\langle 2, 3, 3 \rangle$ he/she can choose to write down a score of 0 for number 1, a score of 2 for number 2 or a score of 6 for number 3. It is not allowed to fill in a score for a number that already has a score. Therefore, the game consists of three turns. The final score is the sum of all three filled in scores. Note that the minimum possible score is 0 and the maximum possible score is 18.
However, we may also allow the player to hold and rethrow after a throw, which means that the player is allowed to put some dice to the side and rethrow the others. This helps the player to get a more favorable score.
In appendix A.3 one can find two mCRL2 models: a model $\mathcal{M}_0$ of the yahtzee game without hold and rethrow and a model $\mathcal{M}_1$ that allows to hold and rethrow once per turn. The number of states $|S|$ of the resulting PLTS is 2550 and 4899 respectively and the number of transitions $|T|$ is 3042 and 21834 respectively. Note that both resulting PLTSs are tree structured (acyclic). All states are ordered in a breadth first search manner. The action *throw* models throwing the dice, the action *write* models writing down a score, the action *hold* models holding dice, the action *label* labels the final states of the game with the final score and the action *endOfGame* shows that we have reached such a final state. The question we are interested in is "What is the probability that a player will end up with a final score of i to j when choosing optimally?". The choices for the player in this game are for what number to fill in the score and which dice to hold. The $pL\mu$-formula $\phi_{i,j}$ that answers the above question is defined as

$$\mu X.(\langle throw \rangle X \vee \langle write \rangle X \vee \langle hold \rangle X \vee \bigvee_{k=i}^{j} \langle label(k) \rangle \underline{1})$$

See the tables below for the timings and resulting values of $\phi_{i,j}$ for different i, j when checked on the models $\mathcal{M}_0$ and $\mathcal{M}_1$ respectively.

	Naive	*CreateRES*	*SolveRES*	Result
$\phi_{18,18}$	0.10	0.26	71	0.000305
$\phi_{17,18}$	0.12	0.29	71	0.00213
$\phi_{0,0}$	0.10	0.27	71	0.128
$\phi_{1,18}$	0.47	0.61	71	1

	Naive	*CreateRES*	*SolveRES*	Result
$\phi_{18,18}$	1.1	31	9.8h	0.0168
$\phi_{17,18}$	1.1	30	9.9h	0.0519
$\phi_{0,0}$	1.1	31	9.9h	0.585
$\phi_{1,18}$	2.2	32	9.9h	1

When looking at the timings, one can see the influence of the size of the formula. When increasing $j - i$ from 0 to 1 ($\phi_{18,18}$ to $\phi_{17,18}$) there is a small increase in running time for the naive algorithm. When increasing $j - i$ from 0 to 17 ($\phi_{18,18}$ to $\phi_{1,18}$), which effectively results in 5 times more $\vee$-operands, the running time at least doubles.
The resulting values seem to contradict if one looks at the last two formulas: the sum of the probability to end up with a score of zero and a non-zero score add up to something greater than one. This however does not contradict, because "choosing optimally" is different between the formulas. With choosing optimally is meant that one chooses such that the resulting value of the formula is maximized. That means that for $\phi_{1,18}$ the player makes the choices to avoid a final score of 0 whereas for $\phi_{0,0}$ the player makes the choices to actually get a final score of 0.
When one compares the resulting values between the different models, one can see how much the ability to hold and rethrow improves the probability of achieving the score the player is aiming at. From $\mathcal{M}_0$ to $\mathcal{M}_1$, the probability of getting the maximum score is more than 50 times greater and the probability of getting the minimum score is more than 4 times greater.

Bounded retransmission protocol

The bounded retransmission protocol is a protocol used to send a file through lossy channels. The file is first split in N chunks, which will be sent separately. Each chunk is sent from the sender to the receiver over a channel K, which has a probability of $\frac{1}{10}$ to lose the chunk. When the receiver receives a chunk, it will send the sender an acknowledgement via channel L, which has a probability of $\frac{1}{20}$ to lose the acknowledgement. When either one of the losses occurs, the sender will retry sending the chunk. However, the sender will only retry sending a chunk up to M number of times. If this M is exceeded, the transmission has failed and the transmission will start over again from scratch.
The mCRL2 model of this protocol is taken from the mCRL2 tool-set and is given in appendix A.4. Note that a number of actions has been hidden by renaming them to *tau*. Out of the actions that are not hidden the action *c_aF* models the sender sending a chunk to channel K (the start of a send attempt), the action *success_frame* models that a chuck has been successfully sent and acknowledged, the action *c_success_file* models that the complete file has been sent successfully and the action *fail_transmission* models that the maximum number of tries has been reached and thus that the process starts over. In the mCRL2 model the number M is represented by `MAX`.
There are three formulas that we will check on this model. The formula

$$\phi_1 = \langle tau \rangle \langle c\_aF \rangle \mu X.(\langle tau \rangle X \vee \langle success\_frame \rangle \langle c\_aF \rangle X \vee \langle c\_success\_file \rangle \underline{1})$$

answers "What is the probability that the file will be sent successfully without losing any chunk?", the formula

$$\phi_2 = \mu X.(\langle tau \rangle X \vee \langle success\_frame \rangle X \vee \langle c\_aF \rangle X \vee \langle c\_success\_file \rangle \underline{1})$$

answers "What is the probability that the file will be sent successfully without losing a chunk more than M times?" and the formula

$$\phi_3 = \mu X.\nu Y.(\langle fail\_transmission \rangle X \vee \langle tau \rangle Y \vee \langle success\_frame \rangle Y \vee \langle c\_aF \rangle Y \vee \langle c\_success\_file \rangle \underline{1})$$

answers "What is the probability that the file will be sent successfully after a finite number of times starting over?". See the tables below for the number of states $|S|$, the number of transitions $|T|$, the timings and resulting values for different values of N and M for ϕ_1, ϕ_2 and ϕ_3 respectively. Note that $|S| = |T|$ since all states in this model only have a single outgoing transition.

N	M	$\|S\| = \|T\|$	Naive	*CreateRES*	*SolveRES*	Result
2	4	79	0.0050	0.0058	0.081	0.731
3	4	117	0.011	0.0087	0.18	0.625
4	4	155	0.019	0.012	0.31	0.534
4	3	127	0.015	0.0095	0.20	0.534
4	2	99	0.012	0.0074	0.12	0.534

N	M	$\|S\| = \|T\|$	Naive	*CreateRES*	*SolveRES*	Result
2	4	79	0.016	0.0052	0.038	1.00
3	4	117	0.036	0.0077	0.082	1.00
4	4	155	0.055	0.010	0.14	1.00
4	3	127	0.041	0.0084	0.094	0.998
4	2	99	0.025	0.0065	0.056	0.988

N	M	$\|S\| = \|T\|$	Naive	*CreateRES*	*SolveRES*	Result
2	4	79	5.4	0.0062	0.15	1
3	4	117	16	0.0092	0.43	1
4	4	155	32	0.012	0.95	1
4	3	127	27	0.0099	0.66	1
4	2	99	21	0.0078	0.41	1

The timings per algorithm when compared to the state space do not come as a surprise. More states implies a higher running time. However, it does seem that the size of the file N has a greater effect on the state space (and therefore the running time) than the maximum number of retransmissions M.
What is quite interesting however is that for formula ϕ_3 the running time of the naive algorithm is far greater than the running time of *SolveRES*. This is due to the nested alternating fixpoint that occurs in ϕ_3. The naive algorithm needs to (iteratively) compute a new value for Y for every iteration of X, while the *SolveRES* algorithm only needs to solve one equation per variable and state.
The resulting values of ϕ_2 when compared to those of ϕ_1 show the impact of allowing a chunk to be retransmitted. In all given examples the probability that the file will arrive successfully is almost equal to 1. The resulting values

of ϕ_3 show that if we would also allow complete retries after failed attempts to send the file, then the probability of the file never arriving at the receiver equals zero. The only behaviour that makes that the file can never arrive is infinite and contains probabilistic behaviour and therefore it happens with probability 0. However, since this behaviour does exist, we should say that it will *almost never* happen that the file will never arrive at the receiver.

8.2 Improvements to the RES approach

As one can see in the above tests, the naive algorithm is usually faster than the *SolveRES* algorithm. However, the naive algorithm is quite unpredictable because sometimes it (theoretically) will need an infinite number of iterations. Practically, this will lead to an approximation of the solution within a finite number of iterations (although the difference with the exact solution is not representable on the system). Since the *SolveRES* algorithm gives the exact solution and has a more predictable running time, we would rather make use of this algorithm. In this section we will give some improvements to the *SolveRES* algorithm that may lower its running time. Note however that these do not change the worst case running time complexity.

8.2.1 Locality

The *SolveRES* algorithm computes the global solution: it solves all equations. However, it is not always the case that all equations need to be solved in a RES to get the solution you are looking for. If so, we can instead solve the local solution by only solving the equations in the RES that are required for the final solution to the model checking problem.
We will give two examples based on the use cases where computing the local solution is favorable. The local RES is created iteratively by creating an equation for each variable in a queue. Each variable that occurs on the right-hand side of the resulting equation is put in the queue if it wasn't already. Initially, the queue contains the variable X_{s_0} when checking $pL\mu$-formula $\sigma X.\phi$ on PLTS $\langle S, s_0, A, T\rangle$. To properly compare the solving times of the global and local RES, the local RES will be sorted with the same ordering as the global RES after creation. The sorting time is excluded from the timing results. In the timing tables we will denote with *CreateRES-L* the creation of the local RES and with *SolveRES-L* the *SolveRES* algorithm that computes the solution of the local RES.

For the first example we will consider the board game. The $pL\mu$ formula that will be checked on the board game is

$$\mu X.(\langle \mathit{moveLeft}\rangle\langle \mathit{moveLeft}\rangle X \vee \langle \mathit{moveLeft}\rangle\langle \mathit{moveRight}\rangle X \\ \vee\langle \mathit{moveRight}\rangle\langle \mathit{moveLeft}\rangle X \vee \langle \mathit{moveRight}\rangle\langle \mathit{moveLeft}\rangle X \vee \langle \mathit{won}\rangle \underline{1})$$

which answers the question "What is the probability to win with an even number of steps when choosing optimally?". The requirement about an even number of steps is enforced in the formula by only checking the winning condition after every two steps. Because the game board has square tiles, the player cannot stop on all squares when always doing two steps at a time. Therefore, only

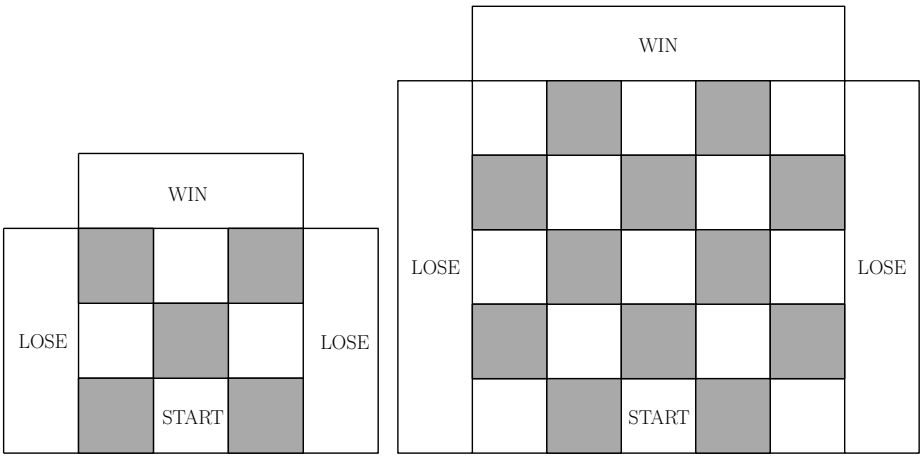


Figure 18: A 3x3 and a 5x5 board of the board game example. The player is not able to stop on any of the squares coloured grey if the player always takes two steps at a time.

about half of all states (and thus only about half of all equations) need to be considered. See figure 18 for some examples that show what squares cannot be stopped on. See the table below for the timings and resulting values.

	Naive	*CreateRES*	*SolveRES*	*CreateRES-L*	*SolveRES-L*	Result
3x3	0.013	0.034	0.024	0.016	0.0067	0.226
5x5	0.043	0.11	0.32	0.057	0.12	0.287

As the second example we consider the yahtzee game. When we introduced this use case we created a formula $\phi_{0,0}$ with which one can check the probability of having a final score of zero. However, we can check this differently. To get a score of zero, the *write* actions should write a score of 0 for each number. This way of checking can be done with the $pL\mu$-formula

$$\mu X.(\langle throw\rangle X \vee \langle write(1,0)\rangle X \vee \langle write(2,0)\rangle X \vee \langle write(3,0)\rangle X \vee \langle hold\rangle X \vee \langle endOfGame\rangle \underline{1})$$

By using parameters for the *write* action, we enforce that only paths are taken that write a score of 0. Therefore, a big part of the state space, namely the trees of states after *write* actions with a nonzero score, does not need to be considered when model checking the above formula. Whereas the global RES's of the above formula checked on the models $\mathcal{M}_0$ and $\mathcal{M}_1$ have respectively 2550 and 4899 equations, the local RES's have respectively 286 and 556 equations, which is about 9 times less. See the table below for the timings and resulting values.

	Naive	*CreateRES*	*SolveRES*	*CreateRES-L*	*SolveRES-L*	Result
$\mathcal{M}_0$	0.14	0.26	53	0.031	0.70	0.128
$\mathcal{M}_1$	2.5	31	9.8h	3.7	9.2m	0.585

As would be expected, the running time of both the *CreateRES-L* and the *SolveRES-L* algorithm are a lot faster than the *CreateRES* and the *SolveRES*

algorithm respectively. Actually, since the *SolveRES* and *SolveRES-L* algorithms are worst case exponential in the size of the RES, the ratio between the running times of both algorithms is far greater than the ratio between the size of the global RES and the local RES. The creation of the RES's on the other hand is linear to the number of states, so the ratio between the running times of *CreateRES* and *CreateRES-L* is about equal to the ratio between the size of the created global RES and the created local RES.

8.2.2 Dependency graph

The substitution step in the *SolveRES* algorithm is rather simplistic. If an equation $(\sigma X = f)$ is solved, it searches in all equations $(\sigma Y = g)$ above for any appearance of the variable X in g to substitute it for f. However, usually only a small number of these equations actually has a g that contains the variable X. If such a g contains X, we say the Y depends on X. We can make the substitution step more efficient if we would create a dependency graph beforehand. Then we can query the dependency graph to immediately get all equations where substitutions are possible.
Note however that this dependency graph needs to be maintained while solving the RES. Whenever an equation $(\sigma X = f)$ is solved, X is no longer dependent on itself. More importantly, if for an equation $(\sigma Y = g)$ the variable(s) X in g are substituted by f, Y no longer depends on X and instead Y now additionally depends on all variables that occur in f. Also, it is possible for dependencies to disappear due to simplifications.
To show the improvement in running time this method brings, we will use the ant on grid and the airplane seats examples. The *CreateRES* algorithm that also computes the dependency graph is denoted with *CreateRES-D*. The *SolveRES* algorithm using the dependency graph is denoted with *SolveRES-D*. See the table below for the timings of both the *SolveRES* and the *SolveRES-D* approach for the ant on grid problem.

	CreateRES	*SolveRES*	*CreateRES-D*	*SolveRES-D*
ϕ_s	0.024	2.2	0.025	1.4
ϕ_d	0.023	2.2	0.026	1.4
ϕ_i	0.026	2.2	0.027	1.4

See the table below for the timings of both the *SolveRES* and the *SolveRES-D* approach for the airplane seats problem. See also figure 19 for a plot of the running times of *SolveRES-D*.

N	*CreateRES*	*SolveRES*	*CreateRES-D*	*SolveRES-D*
2	0.00014	0.00041	0.00039	0.00019
5	0.0020	0.0083	0.0021	0.0016
10	0.0049	0.039	0.0056	0.0041
25	0.013	0.27	0.014	0.011
50	0.027	1.1	0.029	0.022
75	0.041	2.5	0.043	0.033
100	0.055	4.5	0.058	0.045

Since the *CreateRES-D* algorithm now additionally creates a dependency graph

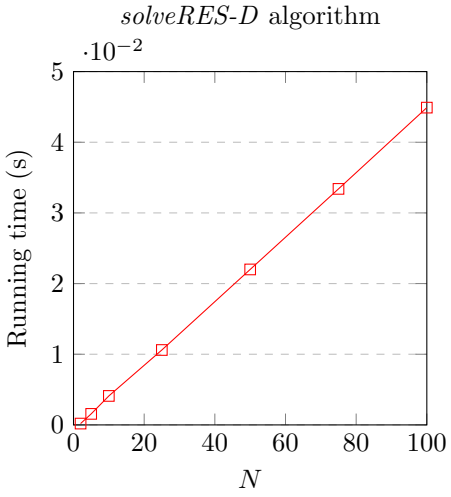


Figure 19: Running times of the *SolveRES-D* algorithm on the airplane seats use case for different number of passengers N.

when compared to *CreateRES*, the running time of *CreateRES-D* is slightly higher. However, when comparing the running times of *SolveRES-D* with *SolveRES*, one can see that this bit of extra creation time is certainly worth it. For the solving algorithms the ant on grid problem shows that using the dependency graph results in a 1.5 times lower running time. The airplane seats problem, where the substitution step dominated the running time complexity, shows that this complexity has reduced by a factor of N making it linear in N instead.

8.2.3 Ordering of equations

As mentioned before, the worst case running time complexity stems from the possible exponential growth of equations due to substitutions. Something that heavily influences the growth of equations is the order in which the equations are solved. We will illustrate this with an example.

Example 8.1. We will use the board game example where we want to check the formula ϕ_w (probability of winning when choosing optimally) on a 3x3 board. We consider two equivalent RES's to solve this problem as shown in figure 20:

- The RES that is created using the global approach, ordering the equations by variable and then state number.
- A RES that is created using the local approach without sorting afterwards, ordering the equations in a breadth first search manner.

The running time for solving the first RES is already given in section 8.1, namely 0.2 seconds. The time needed by *SolveRES* for solving the second RES however exceeds half a day. This is mainly caused by the position of equation $\mu X_9 = 0$. In the first RES, this equation is one of the first equations to be substituted upwards, which results in quite some reductions in equation size early on in the

solving process. Actually, the size of each equation never surpasses the size it started with.
In the second RES however, the equation $\mu X_9 = 0$ appears halfway. Before this equation is reached, the equations below and above have already grown greatly. When substituting the solution of X_5 upwards, which is a maximum of 16 terms each consisting of 3 summands, the right-hand side of X_4 is a maximum of 124 terms each consisting of 4 summands. Each maximum term in this right-hand side contains the variable X_5, which needs to be substituted for the solution of X_5.

This example shows how important the ordering of equations is for the running time. In this section we will give a method of ordering the equations such that the RES can be solved more efficiently. For this method we will also use the two improvements mentioned previously.
However, we are not free to order all equations any way we would want. Using lemma 4.5, we may only swap consecutive equations with the same fixpoint sign. Therefore, we will divide the equations into blocks, such that if two consecutive equations have the same fixpoint sign, they are in the same block. Then using lemma 4.5 we are allowed to order all equations within a block any way we want.

The creation of the local RES, including the partitioning into blocks and the creation of the dependency graph of all equations in the RES is done by *CreateRES-O*. The ordering and solving of the RES is done by *SolveRES-O*. The pseudo code of *SolveRES-O* is given in algorithm 4, where $\mathcal{E}$ and *depGraph* are the RES and dependency graph created by *CreateRES-O* respectively. The function $tarjan(G)$ refers to Tarjan's algorithm [Tar72] on graph G and the function $ROBFS(G, I)$ orders all vertices in G on descending depth by assigning a depth to all vertices in a breadth first search fashion, where all vertices in I have depth 0. We will explain the ordering method below.

Due to the limitations on how we can order the equations, we order the equations per block, from the last block to the first. The ordering is done in two layers.
First we create the dependency graph of all equations in this block. Then using Tarjan's algorithm [Tar72], we partition the vertices of this graph into strongly connected components (SCC's), such that if two vertices u and v are in an SCC, there is a path from u to v and a path from v to u such that only vertices in the SCC are visited. Due to this circular dependency within a SCC, it is possible to completely solve all equations within the SSC if there are no dependencies going out of the SSC. Such an SCC always exists, because if we would make a graph with the SCC's as vertices, this graph is always acyclic. By ordering the SCC's in reverse topological order, we will always consider an SCC when all its outgoing dependencies have already been processed. Fortunately, Tarjan's algorithm will return the SCC's in this order.
Secondly, the equations within the SCC are ordered using breadth first search, where either the vertex corresponding to the initial equation has depth 0 or the vertices in the SCC that have an incoming dependency from vertices outside this SCC have depth 0. Note these two cases cannot overlap. Since we use the local RES, the initial variable will be in the very last SCC that we will consider, which cannot have incoming dependencies. The equations in the SCC are then

$$\mu X_0 = \max(\frac{1}{3}X_3 + \frac{1}{6}X_1 + \frac{1}{2}X_9, \frac{1}{3}X_3 + \frac{1}{6}X_9 + \frac{1}{2}X_1)$$
$$\mu X_1 = \max(\frac{1}{6}X_2 + \frac{1}{2}X_0 + \frac{1}{3}X_4, \frac{1}{2}X_2 + \frac{1}{6}X_0 + \frac{1}{3}X_4)$$
$$\mu X_2 = \max(\frac{1}{6}X_9 + \frac{1}{2}X_1 + \frac{1}{3}X_5, \frac{1}{6}X_1 + \frac{1}{2}X_9 + \frac{1}{3}X_5)$$
$$\mu X_3 = \max(\frac{1}{2}X_9 + \frac{1}{3}X_6 + \frac{1}{6}X_4, \frac{1}{6}X_9 + \frac{1}{3}X_6 + \frac{1}{2}X_4)$$
$$\mu X_4 = \max(\frac{1}{2}X_3 + \frac{1}{3}X_7 + \frac{1}{6}X_5, \frac{1}{6}X_3 + \frac{1}{3}X_7 + \frac{1}{2}X_5)$$
$$\mu X_5 = \max(\frac{1}{3}X_8 + \frac{1}{6}X_9 + \frac{1}{2}X_4, \frac{1}{3}X_8 + \frac{1}{2}X_9 + \frac{1}{6}X_4)$$
$$\mu X_6 = \max(\frac{1}{3}X_{10} + \frac{1}{2}X_9 + \frac{1}{6}X_7, \frac{1}{3}X_{10} + \frac{1}{6}X_9 + \frac{1}{2}X_7)$$
$$\mu X_7 = \max(\frac{1}{3}X_{10} + \frac{1}{6}X_8 + \frac{1}{2}X_6, \frac{1}{3}X_{10} + \frac{1}{2}X_8 + \frac{1}{6}X_6)$$
$$\mu X_8 = \max(\frac{1}{3}X_{10} + \frac{1}{6}X_9 + \frac{1}{2}X_7, \frac{1}{3}X_{10} + \frac{1}{2}X_9 + \frac{1}{6}X_7)$$
$$\mu X_9 = 0$$
$$\mu X_{10} = 1$$

(a) The RES when using the global RES creation.

$$\mu X_1 = \max(\frac{1}{6}X_2 + \frac{1}{2}X_0 + \frac{1}{3}X_4, \frac{1}{2}X_2 + \frac{1}{6}X_0 + \frac{1}{3}X_4)$$
$$\mu X_0 = \max(\frac{1}{3}X_3 + \frac{1}{6}X_1 + \frac{1}{2}X_9, \frac{1}{3}X_3 + \frac{1}{6}X_9 + \frac{1}{2}X_1)$$
$$\mu X_2 = \max(\frac{1}{6}X_9 + \frac{1}{2}X_1 + \frac{1}{3}X_5, \frac{1}{6}X_1 + \frac{1}{2}X_9 + \frac{1}{3}X_5)$$
$$\mu X_4 = \max(\frac{1}{2}X_3 + \frac{1}{3}X_7 + \frac{1}{6}X_5, \frac{1}{6}X_3 + \frac{1}{3}X_7 + \frac{1}{2}X_5)$$
$$\mu X_9 = 0$$
$$\mu X_3 = \max(\frac{1}{2}X_9 + \frac{1}{3}X_6 + \frac{1}{6}X_4, \frac{1}{6}X_9 + \frac{1}{3}X_6 + \frac{1}{2}X_4)$$
$$\mu X_5 = \max(\frac{1}{3}X_8 + \frac{1}{6}X_9 + \frac{1}{2}X_4, \frac{1}{3}X_8 + \frac{1}{2}X_9 + \frac{1}{6}X_4)$$
$$\mu X_7 = \max(\frac{1}{3}X_{10} + \frac{1}{6}X_8 + \frac{1}{2}X_6, \frac{1}{3}X_{10} + \frac{1}{2}X_8 + \frac{1}{6}X_6)$$
$$\mu X_6 = \max(\frac{1}{3}X_{10} + \frac{1}{2}X_9 + \frac{1}{6}X_7, \frac{1}{3}X_{10} + \frac{1}{6}X_9 + \frac{1}{2}X_7)$$
$$\mu X_8 = \max(\frac{1}{3}X_{10} + \frac{1}{6}X_9 + \frac{1}{2}X_7, \frac{1}{3}X_{10} + \frac{1}{2}X_9 + \frac{1}{6}X_7)$$
$$\mu X_{10} = 1$$

(b) A RES when using the local RES creation without sorting afterwards.

Figure 20: Two equivalent RES's that solve ϕ_w on a 3x3 board.

Algorithm 4 *SolveRES-O*($\mathcal{E}$, *depGraph*)

```
1: for block in reversed(E.blocks) do
2:   blockDepGraph := dependency graph of all equations in block
3:   sccs := tarjan(blockDepGraph)
4:   for scc in sccs do
5:     if block is the last block and scc the last SCC to consider then
6:       scc := ROBFS(scc, initialVertex)
7:     else
8:       scc := ROBFS(scc, vertices in scc that have an incoming depen-
         dency in depGraph from a vertex outside scc)
9:     end if
10:    for v in scc do
11:      (σX = f) := v.equation
12:      if v has a self loop in depGraph then
13:        f := solve f for X
14:        update depGraph
15:      end if
16:      for u in parents of v in depGraph do
17:        (σ'Y = g) := u.equation
18:        substitute all X in g with f
19:        update depGraph
20:      end for
21:      remove v from depGraph
22:    end for
23:  end for
24: end for
```

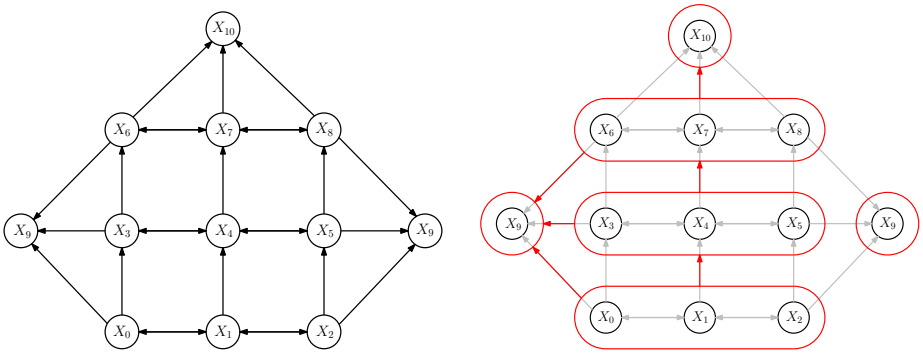


Figure 21: The graph on the left is the dependency graph of the RES given in figure 20. Each equation has a vertex represented by the variable on its left hand side. The graph on the right shows the SCC's of this graph in red. In both graphs the vertex X_9 is given twice to make it visually more pleasing.

solved and substituted in order of descending depth. This way the equations within the SCC are solved in order towards the final solution, which keeps the number of substitutions low.
Note that we remove a vertex from the dependency graph when it has been processed. If we would keep them in, there may still remain dependencies to this vertex from vertices that have not been processed yet. This would lead to extra substitutions that are not of any use for the answer to the original problem.

As an example of the ordering method, we will use the RES of the 3x3 board as shown in figure 20. The dependency graph that corresponds to this RES is shown in figure 21 on the left. When we partition this graph in SCC's, we get the SCC's as shown in figure 21 on the right. The final ordering of the equations will then be as follows, where every equation is represented by its left hand side and the symbol | is used to group equations for which their respective ordering does not matter:

$$\mu X_9|\mu X_{10}\ \mu X_6|\mu X_7|\mu X_8\ \mu X_3|\mu X_4|\mu X_5\ \mu X_0|\mu X_2\ \mu X_1$$

Since the ordering method uses both the locality and dependency graph improvements, we will compare *CreateRES-O* and *SolveRES-O* with *CreateRES-LD* and *SolveRES-LD* respectively, which use both locality and the dependency graph. We will show this comparison for the ant on grid problem and the bounded retransmission protocol. It does not have much use to test this for the airplane seats and the yahtzee problem, since the order that arises from the ordering method described in this section applied to these problems is the same ordering as if we would not use an ordering method.

For the ant on grid problem the ordering method may not seem better than the standard ordering, but due to the way the state space is generated some of the edge states (where the ant lives or dies) are closer to the initial state than some states that are dependent on these states. The timing comparisons are given in the table below.

	CreateRES-LD	*SolveRES-LD*	*CreateRES-O*	*SolveRES-O*
ϕ_s	0.025	1.4	0.025	1.1
ϕ_d	0.026	1.4	0.025	1.1
ϕ_i	0.026	1.4	0.027	1.1

For the bounded retransmission protocol the ordering method is beneficial due to the nature of the model. There are multiple paths possible (retransmissions) to the same state (successful transmission), which have different lengths. Using the standard breadth first search order from mCRL2, some equations that belong to states in the longer paths are solved before the equation that belongs to the state all paths lead to. The timing comparisons are given in the table below for $N = M = 4$.

	CreateRES-LD	*SolveRES-LD*	*CreateRES-O*	*SolveRES-O*
ϕ_1	0.0017	0.0015	0.0017	0.0011
ϕ_2	0.010	0.0091	0.011	0.0058
ϕ_3	0.012	0.030	0.012	0.040

As could be expected, the running times for most of the model checking problems are lower when using the ordering method. Only for ϕ_3 for the bounded retransmission protocol this is not the case. When looking at the RES of this problem during the solving problem, it seems that the average size of each equation when substituting it is slightly greater on average when using *SolveRES-O* (however, the maximum equation size between the two options is equal).
This can be explained as follows. The dependency graph for the νY-block is acyclic, so every vertex is its own SCC, while the complete dependency graph is actually cyclic. When solving the νY-equations in reverse topological order, most substitutions are made with a real formula that contains X-variables. When using the breadth first search order as used by *SolveRES-LD*, some equations in the νY-block may be substituted before their dependencies are processed. Since most of these equations only have a single dependency, the substitution that is done is very small. In the *SolveRES-O* ordering however these equations are handled later, causing a bigger substitution due to the X-variables.

8.3 Reflection on results

Although the naive algorithm theoretically has no guarantee to terminate, it is usually quite faster in practice than the RES approach when limited by the precision of float representation of the programming language. This especially holds for larger systems due to the worst case exponential running time of the RES approach. However, with the improvements listed in the above section we have brought the running time closer or sometimes even below the running time of the naive algorithm.
The locality improvement has shown a great reduction in running time because it reduces the size of the RES, but the power of this improvement depends on the model checking problem and does not always help. The improvement that uses a dependency graph for substitutions has given a great reduction in running time independent of the given problem. The improvement that orders the equations may not have shown great reductions in running time, but this is because the order of states of most models was already in an order that leads

to rather efficient solving of the RES. Using this improvement results in an efficient order independently of how the model is defined.

There is of course still room for more improvement. The ordering method used is not optimal and as shown for the bounded retransmission protocol sometimes even worse than a simple breadth first search order. Another possible ordering method that may be more beneficial would be for instance to (dynamically) order the equations on the number of dependencies they have (per block).
A part of the algorithm that we have not paid that much attention to is the simplification of the real formulas, while performance profiling indicates that this amounts for at least half of the running time, often at least 80%. Especially finding and removing terms from $\bigsqcap$ and $\bigsqcup$ formulas that are definitely worse than another term takes much time.
Now the algorithm tries to simplify the complete right-hand side of an equation after the creation of the RES, each solving step and each substitution step. The simplification time could be reduced by only considering subformulas of this right-hand side that have actually changed after the solving and substitution steps. The simplification step can also be strengthened. For instance, removing definitely worse terms from $\bigsqcap$ and $\bigsqcup$ formulas does not use the fact that we only work with values within the interval $[0, 1]$.

When looking at the models and formulas instead of the algorithms, there could have been more variation in them. The models are mostly only grid or tree like with final (desirable) states, with slight exception of the bounded retransmission protocol. Also, almost all formulas describe reachability properties. However, it seems to be rather difficult to come up with different types of models and different types of formulas that result in interesting and meaningful $pL\mu^{\odot}_{\oplus}$ model checking problems. Especially when one looks into infinite behaviour, the result of the model checking problem is often equal to 0 or 1, which can usually be easily seen from the model itself.
One type of formula that should be investigated more is the $pL\mu^{\odot}_{\oplus}$ formula with alternating fixpoint operators. The bounded retransmission protocol use case contains one such formula and it already shows quite different results compared to the other formulas. The running time of the *SolveRES* algorithm is quite better than the naive algorithm and when using the ordering improvement the running time actually increases. However, since we only tested one formula with alternating fixpoint operators, this behaviour may not be very representative.

8.4 Experimental addition: Mean value

The model checking problems that one can specify with $pL\mu^{\odot}_{\oplus}$ always returns a probability. This probability can be seen as the *mean probability* that the behaviour expressed by a $pL\mu^{\odot}_{\oplus}$-formula happens in a system. However, when analyzing probabilistic systems one can also be interested in the *mean value* of some attribute, which can be outside of the scope of the interval $[0, 1]$. In this section we will shortly propose how to tackle this by making a slight alteration to the logic of $pL\mu^{\odot}_{\oplus}$. Since this idea is somewhat out of the scope of this paper, we will leave the proof or other ideas for this matter as future research.

The idea to tackle this is by adding a so-called label atom, which we will de-

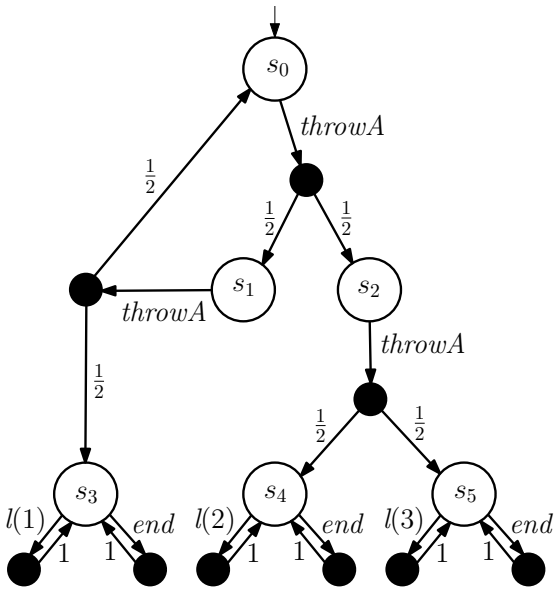


Figure 22: A PLTS that models a three sided die using a two sided die. The action *label* is shortened to *l*.

note with L. Given a PLTS $\langle S, s_0, A, T\rangle$, the semantics of this label atom is as follows:

$$[\![L]\!]_e(s) = \begin{cases} v & \text{if } (s, label(v), s) \in T \\ 0 & \text{else} \end{cases}$$

where $v \in \mathbb{R}^+$ is a positive real number. Note that this requires that each state only has at most one self loop with the action *label*.
Officially adding such an operator would formally force a revision of the underlying definitions and proofs, since this brings the value of a formula outside of the lattice $\langle [0,1], \leq\rangle$. However, as we only allow positive values for labels, we can first divide all label values with the maximum label value M beforehand such that all labels fall within $[0,1]$. After the value of the model checking problem with altered labels has been computed, we multiply the result with M to get the desired mean value. This way the label atom could simply be seen as the atom λ, except that it is dependent on the state.
Note that the operators $\cdot$, $\odot$, $\ominus$ and $\oplus$ do not have an intuitive meaning in this context, so we will not consider them. The intuitive meaning of the operators $\wedge$ and $\vee$ are now "pick the lowest value" and "pick the highest value" respectively.
We will show results of adding this operator with two examples. First we will use the model as shown in figure 22, which models a three sided die using a two sided die. Note that we used such a model before, but this model has more (self-loop) actions. The final states of this model are labeled with the value of the modeled three sided die. With the formula $\mu X.(\langle throwA\rangle X \vee \langle end\rangle L)$ we can get the mean value of the three sided die. Checking this formula results in 2, which is expected since $\frac{1}{3}\cdot 1 + \frac{1}{3}\cdot 2 + \frac{1}{3}\cdot 3 = 2$.
Secondly we will use the yahtzee example. With the formula

$$\mu X.(\langle throw\rangle X \vee \langle write\rangle X \vee \langle hold\rangle X \vee \langle endOfGame\rangle L)$$

we can get the mean value of the yahtzee game when the player chooses optimally (to maximize his/her score). The timings and resulting values are shown in the table below.

	Naive	*CreateRES-O*	*SolveRES-O*	Result
$\mathcal{M}_0$	0.10	0.033	0.67	8.97
$\mathcal{M}_1$	1.1	35	85	12.3

Since the maximum score for the yahtzee game is 18, it makes sense that the mean final score of the yahtzee game is close to 9. As expected the mean final score of the yahtzee game increases when allowing the player to hold and rethrow once, since the player has better capabilities to maximize his/her score. Although not shown here, solving a $pL\mu$-formula with label atoms has the same running time as solving the same $pL\mu$-formula but with lambda atoms instead of label atoms for both the naive and *SolveRES* algorithm (and variants), as could be expected from the semantics of the label atom.

9 Conclusion

We have given the logic $pL\mu^{\odot}_{\oplus}$ defined in [Mio12]. With this logic one can check quantitative properties that answer questions of the form "what is the probability that ...?" on Probabilistic Labeled Transition Systems, a type of transition system that allows both non-deterministic and probabilistic behaviour.
We have created some intuition (combinations of) atoms and operators in this logic, with the aim to be able to create formulas that express meaningful properties. One main issue in this is dependence. Since it is generally hard to extract the dependence between two formulas ϕ_1 and ϕ_2 given a model, it is not easy to know what operator to use to express the probability that both ϕ_1 and ϕ_2 happen ($\wedge$, $\cdot$ and $\ominus$) or at least one of the two happens ($\vee$, $\odot$ and $\oplus$). However, when preceded by a fixpoint operator, the operators $\wedge$ and $\vee$ seem to be the better choices.

To solve a $pL\mu^{\odot}_{\oplus}$ model checking problem, we have given an approximation algorithm that directly applies the semantics. Also we have defined a $\text{RES}^{\odot}_{\oplus}$, a type of fixpoint equation system over the lattice $\langle [0,1], \leq \rangle$ inspired by the work of Mader [Mad97], and a translation from a $pL\mu^{\odot}_{\oplus}$ model checking problem to a $\text{RES}^{\odot}_{\oplus}$. Also, we have given an algorithm *SolveRES* based on Gauss elimination that can be used to solve a $\text{RES}^{\odot}_{\oplus}$, which consists of an equation solving step and a substitution step. However, we have only been able to give a method for the equation solving step when applied to a RES, which is strict substructure of the $\text{RES}^{\odot}_{\oplus}$ that does not allow the operators $\cdot$, $\odot$, $\ominus$ and $\oplus$.

Fortunately, using the corresponding sub-logic $pL\mu$ we have been able to specify a number of meaningful use cases. On these use cases we have compared *SolveRES* with a naive algorithm that recurses over the $pL\mu$-formula and uses fixpoint iteration to solve fixpoint operators. Although the latter algorithm has an infinite worst case running time, it is usually faster than the *SolveRES* algorithm in practice due to *SolveRES*'s exponential worst case running time complexity.
To reduce the running time of the *SolveRES* algorithm we have given a number of improvements to the algorithm. One can compute the local RES which only considers equations that are needed for the solution. Using a dependency graph, one can immediately extract what equations are eligible for substitution. Since the ordering of the equations is quite important for the running time, we have also given a method to order the equations in a beneficial way. However, as these improvements do not change the worst case running time complexity, the approximation algorithm is still often faster than the improved *SolveRES* algorithm.

9.1 Future research

As discussed in section 6.1, a probability space over paths in a PLTS with a scheduler to remove non-deterministic behaviour cannot be used to create events that correspond to $pL\mu^{\odot}_{\oplus}$-formulas or formulas of a sub-logic. Whether other probability spaces exist for the PLTS for this end however is unknown.
What sets the μ-calculus approach apart from other verification logics like CTL is its expressiveness. Most $pL\mu^{\odot}_{\oplus}$-formulas that we have considered however are

reachability properties, which can also be expressed in CTL-like logics. Formulas that cannot be expressed by such logics, such as formulas with multiple (alternating) fixpoints operators, are not much represented in this thesis. This is because it is quite difficult to come up with such formulas that can be checked on a concrete model such that both the formula and the solution are meaningful.

Mio has shown in his PhD thesis [Mio12] that one can represent a $pL\mu^{\odot}_{\oplus}$-model checking problem as a $2\frac{1}{2}$-player meta-parity game. We on the other hand have shown that one can represent a $pL\mu^{\odot}_{\oplus}$-model checking problem as a $\text{RES}^{\odot}_{\oplus}$. It could be interesting whether and if so how one can translate from a $\text{RES}^{\odot}_{\oplus}$ to a $2\frac{1}{2}$-player meta-parity game or vice versa such that the solutions correspond, similar as to one can translate a Boolean Equation System to a 2-player parity game [Kei06] (in the context of an $L\mu$ model checking problem).

The algorithm *SolveRES* is practically only capable of solving a RES, which is a strict substructure of a $\text{RES}^{\odot}_{\oplus}$. We have discussed that the equation solving method for RES's cannot be applied when the other operators in $\text{RES}^{\odot}_{\oplus}$'s are added, but that does not mean it is impossible to solve such equations. It could also be investigated whether other algorithms than Gauss elimination that are used to solve BES's could be adapted to solve $\text{RES}^{\odot}_{\oplus}$'s.
We have given a number of methods to improve the running time of the *SolveRES* algorithm, but more improvements are possible. As mentioned in the reflection, especially the simplification of real formulas can be improved upon. Also, other improvements could be explored, such as ordering the equations in the order of least dependencies first or removing the need of a normal form.
When we would translate a RES to a $2\frac{1}{2}$-player meta-parity game, the resulting game is also a (less general) $2\frac{1}{2}$-player parity game which does not contain any branching nodes. Very little number of algorithms are known to solve such $2\frac{1}{2}$-player parity games [CH06, HSTZ17]. It could be interesting to see how the *SolveRES* algorithm competes with these algorithms.

Formulas in the logic of $pL\mu^{\odot}_{\oplus}$ always returns a probability when checked on a model. However, in the context of probabilistic systems one can also be interested in the mean value of some property of the system. An idea to tackle this was proposed in section 8.4, which allowed us to calculate the mean final score of a yahtzee game. Another type of mean value that might be interesting is the mean number of transitions needed to satisfy some property.

References

[AN01] André Arnold and Damian Niwinski. *Rudiments of Calculus*, volume 146. Elsevier, 2001.

[Ant13] Gary Antonick. Ant on a grid. `https://wordplay.blogs.nytimes.com/2013/08/12/ants-2/`, 2013. [Online; accessed 6-9-2017].

[ASB95] Adnan Aziz, Vigyan Singhal, and Felice Balarin. It usually works: The temporal logic of stochastic systems. In *CAV*, volume 939 of *Lecture Notes in Computer Science*, pages 155–165. Springer, 1995.

[AW06] Suzana Andova and Tim A. C. Willemse. Branching bisimulation for probabilistic systems: Characteristics and decidability. *Theor. Comput. Sci.*, 356(3):325–355, 2006.

[BBH+84] Hans Bekic, Dines Bjørner, Wolfgang Henhapl, Cliff B. Jones, and Peter Lucas. *On the Formal Definition of a PL/I Subset (Selected parts)*, volume 177 of *Lecture Notes in Computer Science*. Springer, 1984.

[BC98] Christel Baier and E Clarke. The algebraic mu-calculus and mtbdds. In *Proc. 5th Workshop on Logic, Language, Information and Computation (WOLLIC'98)*, pages 27–38. Citeseer, 1998.

[Bir40] Garrett Birkhoff. *Lattice theory*, volume 25. American Mathematical Soc., 1940.

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[CGK+13] Sjoerd Cranen, Jan Friso Groote, Jeroen J. A. Keiren, Frank P. M. Stappers, Erik P. de Vink, Wieger Wesselink, and Tim A. C. Willemse. An overview of the mcrl2 toolset and its recent advances. 7795:199–213, 2013.

[CH06] Krishnendu Chatterjee and Thomas A. Henzinger. Strategy improvement and randomized subexponential algorithms for stochastic parity games. In *STACS*, volume 3884 of *Lecture Notes in Computer Science*, pages 512–523. Springer, 2006.

[dA03] Luca de Alfaro. Quantitative verification and control via the mu-calculus. In *CONCUR*, volume 2761 of *Lecture Notes in Computer Science*, pages 102–126. Springer, 2003.

[dAM01] Luca de Alfaro and Rupak Majumdar. Quantitative solution of omega-regular games. In *STOC*, pages 675–683. ACM, 2001.

[FBH83] Ruma Falk and Maya Bar-Hillel. Probabilistic dependence between events. *The Two-Year College Mathematics Journal*, 14(3):240–247, 1983.

[Fel68] William Feller. *An introduction to probability theory and its applications: volume I*, volume 3. John Wiley & Sons New York, 1968.

[GM14] Jan Friso Groote and Mohammad Reza Mousavi. *Modeling and Analysis of Communicating Systems*. MIT Press, 2014.

[HK97] Michael Huth and Marta Z. Kwiatkowska. Quantitative analysis and model checking. In *LICS*, pages 111–122. IEEE Computer Society, 1997.

[HKW11] Yi-Ling Hwong, Vincent J. J. Kusters, and Tim A. C. Willemse. Analysing the control software of the compact muon solenoid experiment at the large hadron collider. In *FSEN*, volume 7141 of *Lecture Notes in Computer Science*, pages 174–189. Springer, 2011.

[HSTZ17] Ernst Moritz Hahn, Sven Schewe, Andrea Turrini, and Lijun Zhang. Synthesising strategy improvement and recursive algorithms for solving 2.5 player parity games. In *VMCAI*, volume 10145 of *Lecture Notes in Computer Science*, pages 266–287. Springer, 2017.

[I+81] VI Istrc et al. *Fixed point theory: an introduction*, volume 7. Springer, 1981.

[Kei06] Misa Keinänen. Techniques for solving boolean equation systems. 2006.

[KKZ05] Joost-Pieter Katoen, Maneesh Khattri, and Ivan S. Zapreev. A markov reward model checker. In *QEST*, pages 243–244. IEEE Computer Society, 2005.

[KNP02] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM: probabilistic symbolic model checker. In *Computer Performance Evaluation / TOOLS*, volume 2324 of *Lecture Notes in Computer Science*, pages 200–204. Springer, 2002.

[Kol50] Andreĭ Nikolaevich Kolmogorov. Foundations of the theory of probability. 1950.

[Koz83] Dexter Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.

[KT28] Bronisław Knaster and Alfred Tarski. Un théorème sur les fonctions d'ensembles. *Ann. Soc. Polon. Math*, 1928.

[LM05] Alberto Lluch-Lafuente and Ugo Montanari. Quantitative mu-calculus and CTL defined over constraint semirings. *Theor. Comput. Sci.*, 346(1):135–160, 2005.

[LS89] Kim Guldstrand Larsen and Arne Skou. Bisimulation through probabilistic testing. In *POPL*, pages 344–352. ACM Press, 1989.

[Mad97] Angelika Mader. *Verification of modal properties using Boolean equation systems*. PhD thesis, Technical University Munich, 1997.

[Mio12] Matteo Mio. *Game semantics for probabilistic modal μ-calculi*. PhD thesis, The University of Edinburgh, 2012.

[MM97] Carroll Morgan and Annabelle McIver. A probabilistic temporal calculus based on expectations. In *Proc. Formal Methods Pacific*, volume 97. Citeseer, 1997.

[MM06] Annabelle McIver and Carroll Morgan. A novel stochastic game via the quantitative *mu*-calculus. *Electr. Notes Theor. Comput. Sci.*, 153(2):195–212, 2006.

[MM07] Annabelle McIver and Carroll Morgan. Results on the quantitative μ-calculus qmμ. *ACM Trans. Comput. Log.*, 8(1):3, 2007.

[MMS96] Carroll Morgan, Annabelle McIver, and Karen Seidel. Probabilistic predicate transformers. *ACM Trans. Program. Lang. Syst.*, 18(3):325–353, 1996.

[NCI99] Murali Narasimha, Rance Cleaveland, and S. Purushothaman Iyer. Probabilistic temporal logics via the modal mu-calculus. In *FoSSaCS*, volume 1578 of *Lecture Notes in Computer Science*, pages 288–305. Springer, 1999.

[Seg95] Roberto Segala. *Modeling and verification of randomized distributed real-time systems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1995.

[Sto02] Mariëlle Stoelinga. An introduction to probabilistic automata. *Bulletin of the EATCS*, 78:176–198, 2002.

[Tar72] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[Var85] Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *FOCS*, pages 327–338. IEEE Computer Society, 1985.

[vBGH+17] Rutger van Beusekom, Jan Friso Groote, Paul F. Hoogendijk, Robert Howe, Wieger Wesselink, Rob Wieringa, and Tim A. C. Willemse. Formalising the dezyne modelling language in mcrl2. In *FMICS-AVoCS*, volume 10471 of *Lecture Notes in Computer Science*, pages 217–233. Springer, 2017.

[Zie04] Wieslaw Zielonka. Perfect-information stochastic parity games. 2987:499–513, 2004.

A mCRL2 models

A.1 Ant on a grid

```
act dead,live,step;

proc X(x,y:Pos)=
     (x==1 || x==8) -> dead.X(x,y) <>
     (y==1 || y==8) -> live.X(x,y) <>
     (
       step.dist b1,b2:Bool[1/4].
       (( b1 &&  b2) -> X(x+1,y)+
        ( b1 && !b2) -> X(max(1,x-1),y)+
        (!b1 &&  b2) -> X(x,y+1)+
        (!b1 && !b2) -> X(x,max(1,y-1))
     ) );

init X(5,3);
```

A.2 Airplane seats

```
map  N:Pos;
eqn  N=100;

act  last_passenger_has_his_own_seat:Bool;
     enter_plane:Bool#Bool;
     enter;

proc Plane(everybody_has_his_own_seat:Bool, number_of_empty_seats:Int)=
       (number_of_empty_seats==0)
         -> last_passenger_has_his_own_seat(everybody_has_his_own_seat).delta
         <> (enter.dist b0:Bool[if(everybody_has_his_own_seat,if(b0,1,0),
             if(b0,1-1/number_of_empty_seats,1/number_of_empty_seats))].
           b0 -> enter_plane(true,false)
               .Plane(everybody_has_his_own_seat,number_of_empty_seats-1)
              <> dist b1:Bool[if(b1,1/number_of_empty_seats,1-1/number_of_empty_seats)].
                enter_plane(false,b1).
                Plane(if(number_of_empty_seats==1,everybody_has_his_own_seat,b1),
                  number_of_empty_seats-1));

init enter.dist b:Bool[if(b,1/N,(N-1)/N)].Plane(b,N-1);
```

A.3 Yahtzee

No holding dice

```
act
  throw;
  write: Pos # Nat;
  label: Nat;
  endOfGame;

proc
  P(filled: List(Pos), value: Nat) =
    (#filled < 3) -> (
      throw.dist d1, d2, d3: Pos[if(d1 < 4 && d2 < 4 && d3 < 4, 1/27, 0)]
      .sum c: Pos.(c < 4 && !(c in filled)) ->
        sum sc: Nat.(sc == if(d1 == c, c, 0) + if(d2 == c, c, 0) + if(d3 == c, c, 0)) ->
          write(c, sc).P(c |> filled, value + sc))
    <>
      (label(value).P(filled, value) + endOfGame.P(filled, value));

init P([], 0);
```

Holding and rethrowing dice once

```
act
  throw;
  write: Pos # Nat;
  hold: Bool # Bool # Bool;
  label: Nat;
  endOfGame;

proc
  P(filled: List(Pos), value: Nat) =
    (#filled < 3) -> (
      throw.dist d1, d2, d3: Pos[if(d1 < 4 && d2 < 4 && d3 < 4, 1/27, 0)]
      .sum b1, b2, b3: Bool.hold(b1, b2, b3)
      .dist d11, d22, d33: Pos[if(b1, if(d1 == d11, 1, 0), if(d11 < 4, 1/3, 0))
                               *if(b2, if(d2 == d22, 1, 0), if(d22 < 4, 1/3, 0))
                               *if(b3, if(d3 == d33, 1, 0), if(d33 < 4, 1/3, 0))]
      .sum c: Pos.(c < 4 && !(c in filled)) ->
        sum sc: Nat.(sc == if(d11 == c, c, 0) + if(d22 == c, c, 0)
                                              + if(d33 == c, c, 0)) ->
          write(c, sc).P(c |> filled, value + sc))
    <>
      (label(value).P(filled, value) + endOfGame.P(filled, value));

init P([], 0);
```

A.4 Bounded retransmission protocol

```
map N:Pos;
    MAX:Pos;

eqn N=4;
    MAX=2;

act new_file;
    fail_transmission;
    success_frame;
    send_aF;  read_aF;  c_aF;
    read_aB;  send_aB;  c_aB;
    read_aA;  send_aA;  c_aA;
    read_aG;  send_aG;  c_aG;
    read_TO_Ack; send_TO_Ack;  c_TO_Ack;
    read_TO_Msg; send_TO_Msg;  c_TO_Msg;
    send_success_file; read_success_file;  c_success_file;
    send_sync; read_sync;  c_sync;

proc sender(s:Int, srep:Int, nrtr:Int, i:Int)=
        %idle
        (s==0)->new_file.sender(1,0,nrtr,1)+
        % next frame
        (s==1)->send_aF.sender(2,srep,nrtr,i)+
        % wait ack
        (s==2)->read_aB.sender(4,srep,nrtr,i)+
        (s==2)->read_TO_Msg.sender(3,srep,nrtr,i)+
        (s==2)->read_TO_Ack.sender(3,srep,nrtr,i)+
        % retransmit
        (s==3 && nrtr<MAX)->send_aF.sender(2,srep,nrtr+1,i)+
        (s==3 && nrtr==MAX && i<N)->fail_transmission.sender(5,1,nrtr,i)+
        (s==3 && nrtr==MAX && i==N)->fail_transmission.sender(5,2,nrtr,i)+
        % success
        (s==4 && i<N)->success_frame.sender(1,srep,0,i+1)+
        (s==4 && i==N)->send_success_file.sender(5,3,0,i)+
        % resync
        (s==5)->send_sync.sender(0,srep,0,i)
    ;

    receiver(r:Int)=
        % new file
        (r==0)->read_aG.receiver(1)+
        (r==1)->send_aA.receiver(0)+
        read_success_file.receiver(0)+
        read_sync.receiver(0)
    ;
```

```
    channelK(k:Int)=
        % idle
        (k==0)->read_aF.( dist b:Bool[9/10].
                          b-> channelK(1)
                          <>  channelK(2)
                         )+
        % sending
        (k==1)->send_aG.channelK(0)+
        % lost
        (k==2)->send_TO_Msg.channelK(0)
    ;

    channelL(l:Int)=
        % idle
        (l==0)->read_aA.( dist b:Bool[1/20].
                          b-> channelL(2)
                          <>  channelL(1)
                         )+
        % sending
        (l==1)->send_aB.channelL(0)+
        % lost
        (l==2)->send_TO_Ack.channelL(0)
    ;

init
    hide({c_sync, c_aB, c_aA, c_aG, c_TO_Ack, c_TO_Msg, new_file},
        allow({new_file, fail_transmission, success_frame, c_aF,
           c_aB, c_aA, c_aG, c_TO_Ack, c_TO_Msg, c_success_file,
           c_sync},
    comm({read_aF | send_aF -> c_aF,
          read_aB | send_aB -> c_aB,
          read_aA | send_aA -> c_aA,
          read_aG | send_aG -> c_aG,
          read_TO_Ack | send_TO_Ack -> c_TO_Ack,
          read_TO_Msg | send_TO_Msg -> c_TO_Msg,
          send_success_file | read_success_file -> c_success_file,
         send_sync | read_sync -> c_sync},

        sender(0,0,0,0)||
        channelK(0)||
        channelL(0)||
        receiver(0)
        )));
```