PAWS: A Tool for the Analysis of Weighted Systems[*]

Barbara König
Universität Duisburg-Essen
barbara_koenig@uni-due.de

Sebastian Küpper
Universität Duisburg-Essen
sebastian.kuepper@uni-due.de

Christina Mika
Universität Duisburg-Essen
christine.mika@uni-due.de

PAWS is a tool to analyse the behaviour of weighted automata and conditional transition systems. At its core PAWS is based on a generic implementation of algorithms for checking language equivalence in weighted automata and bisimulation in conditional transition systems. This architecture allows for the use of arbitrary user-defined semirings. New semirings can be generated during run-time and the user can rely on numerous automatisation techniques to create new semiring structures for PAWS' algorithms. Basic semirings such as distributive complete lattices and fields of fractions can be defined by specifying few parameters, more exotic semirings can be generated from other semirings or defined from scratch using a built-in semiring generator. In the most general case, users can define new semirings by programming (in C#) the base operations of the semiring and a procedure to solve linear equations and use their newly generated semiring in the analysis tools that PAWS offers.

1 Introduction

In recent times, modelling techniques have shifted from a purely acceptance-based reasoning to one that takes various notions of weight and quantities into consideration. The theory of weighted automata gives rise to a flexible framework, where acceptance behaviour can be quantified in numerous ways. Probabilities are commonly used to express the likelihood of making a given transition, whereas the tropical semiring is often used to denote the cost of making a transition. Due to the great variety of semirings the theory of weighted automata can be applied to a multitude of different fields of interest, such as natural language processing, biology or performance modelling (see e.g. [12]). Typical questions regarding weighted automata concern their language (or traces), for instance language equivalence. Due to the generality of the notion of weighted automata, they are a versatile means for modelling purposes, though decidability results are not as strong as for non-deterministic automata. In particular, it is well-known that language equivalence is an undecidable problem for weighted automata over the tropical semiring [16]. This, however, is not a damning result for language equivalence, since it is also well-known that language equivalence is decidable for many other semirings such as the two-valued Boolean algebra (for which weighted automata are just nondeterministic finite automata) or fields (where decision procedures run in polynomial time [14, 7]). PAWS[1] is a tool that offers algorithms to decide language equivalence and the threshold or universality problem for weighted automata.

Based on our previous work in [15] and [5], the tool PAWS implements two different approaches to language equivalence checks for weighted automata. One approach employs a backwards search assuming at first that all states are language equivalent and exploring words from the end to the beginning to determine non-equivalent pairs of states, similar to partition refinement. Different from usual partition refinement algorithms, this algorithm does not necessarily terminate at the moment when the partitions cannot be further refined in one step, because refinements can happen at a later point of time. The termination condition has to be chosen differently here, and as expected, the algorithm does not necessarily

[*]Research partially supported by DFG project BEMEGA.

[1]The tool can be downloaded from www.ti.inf.uni-due.de/research/tools/paws/

Submitted to:
QAPL 2017

terminate for all semirings (see the undecidability result by Krob [16]). However, for many semirings it does and in either case it is always a suitable semi-decision procedure for the absence of language equivalence. This approach is closely related to the line of work started by Schützenberger, [20] and later generalised in [3], using the notion of conjugacy.

Additionally, PAWS offers a second approach to decide language equivalence [5], which stands in the tradition of Bonchi's and Pous' seminal work on equivalence checks for NFAs using up-to techniques [6]. Here, a language equivalence relation on vectors is built, starting from the initial pair of vectors suspected to be language equivalent. As the algorithm progresses, it builds a language equivalence relation similar to a bisimulation relation and stops at the moment when a suitable relation proving language equivalence is found, or a witness to the contrary appears. The algorithm works up-to congruence and therefore prunes the relation on-the-fly, dropping redundant vector pairs. The flexibility of this optimised variant of the algorithm is reduced when compared to the partition refinement algorithm, but it can still be used for a variety of user-generated semirings, such as rings and l-monoids, and can lead to an exponential speed up in some cases. Based on a similar approach, PAWS also offers a decision procedure for the universality problem for weighted automata over the tropical semiring of the natural numbers, which is potentially exponentially faster than a naive approach due to Kupferman et al. [2]. The universality problem checks whether from a given initial vector, all words have a weight smaller than or equal to a given threshold.

Finally, PAWS also considers conditional transition systems, which, rather than adding weights to transitions, extend traditional transition systems by means of conditions, or product versions to enable flexible modelling of software product lines, while taking possible upgrades between different products of a software product line into account [8]. For these kinds of systems, bisimulation rather than language equivalence is considered, because the user experience for different products is in the focus. The bisimulation check can be performed on any finite distributive lattice defined via its set of join irreducible elements. Alternatively, a BDD-based implementation of (certain) lattices is offered and allows for a significantly faster bisimulation check as presented in [4]. Again, the bisimulation check is parametrised over the lattice used and can accept any lattice, either defined directly via its irreducible elements, or using the BDD-based approach, as input.

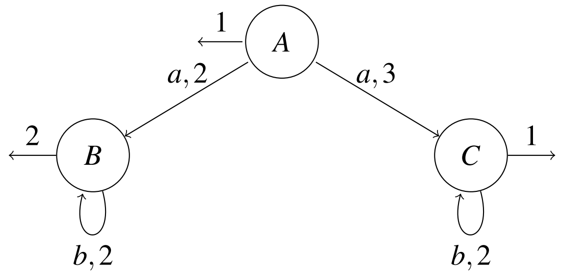
A key feature of PAWS is its extensibility. The algorithms are parametrised over the semiring and it is therefore possible to use the algorithms PAWS offers, not only for the semirings that come pre-implemented, but also for newly generated semirings. For that purpose, PAWS offers ways of adding new semirings and executing algorithms for these semirings. All algorithms are implemented generically and can be used for various semirings or l-monoids, provided all necessary operations such as addition, multiplication, and solving linear equations are specified. Therefore, PAWS is equipped with a semiring generator that allows to generate new semirings that are not pre-implemented and to define weighted automata or conditional transition systems over these. Specifically, we have built several layers of automatisation, so that whenever one, e.g., defines a complete distributive lattice, it suffices to give a partial order, from which the lattice of downward-closed sets is generated [11] and all operations are provided automatically. Building semirings from other semirings using crossproducts is almost completely automatised and modulo rings are automatised using Hensel liftings [10]. In addition, it is possible to add arbitrary semirings by providing code for the operations mentioned before.

2 Preliminaries: System Types and Decision Problems

Here we give a short overview over the systems that PAWS can analyse, i.e. weighted automata and conditional transition systems. For that purpose we require the notions of semirings and distributive lattices.

- A *semiring* is a tuple $\mathbb{S} = (S, +, \cdot, 0, 1)$ where $(S, +, 0)$ is a commutative monoid, $(S, \cdot, 1)$ is a monoid, 0 *annihilates* $\cdot$ (i.e., $0 \cdot s_1 = 0 = s_1 \cdot 0$) and $\cdot$ *distributes over* $+$ (i.e., $(s_1 + s_2) \cdot s_3 = s_1 \cdot s_3 + s_2 \cdot s_3$ and $s_3 \cdot (s_1 + s_2) = s_3 \cdot s_1 + s_3 \cdot s_2$, for all $s_1, s_2, s_3 \in S$).
- A *complete distributive lattice* is a partially ordered set $(L, \sqsubseteq)$ where for all subsets $L' \subseteq L$ of L the infimum $\bigsqcap L'$ and the supremum $\bigsqcup L'$ w.r.t. the order $\sqsubseteq$ exists and infimum distributes over finite suprema: $(\ell_1 \sqcup \ell_2) \sqcap \ell_3 = (\ell_1 \sqcap \ell_3) \sqcup (\ell_2 \sqcap \ell_3)$ for all $\ell_1, \ell_2, \ell_3 \in L$. Together with $\top = \bigsqcup L$ and $\bot = \bigsqcap L$, a complete distributive lattice forms a semiring $\mathbb{L} = (L, \sqcup, \sqcap, \bot, \top)$.
- An l-monoid is a lattice $(L, \sqcup, \sqcap, \bot, \top)$ together with a monoid $(L, \cdot, 1)$ such that $\cdot$ distributes over $\sqcup$. If $\bot$ annihilates $\cdot$, i.e. $\bot \cdot \ell = \bot$ for all $\ell \in L$, then the l-monoid can be regarded as a semiring $(L, \sqcup, \cdot, \bot, 1)$.

A weighted automaton (WA) can be understood as a non-deterministic automaton that additionally carries weights from a given semiring $\mathbb{S}$ on each transition, as well as a termination weight for each state. Rather than accepting or rejecting a word w, a state x in a weighted automaton associates it with a value from $\mathbb{S}$. This value can be obtained as follows: Take the sum of the weight of all w-labelled paths p starting in x. The weight of a path p is the product of all transition weights along p, including the termination weight. Consider for instance the following simple weighted automaton over the field $\mathbb{R}$:



The weight of the word ab in state A can be computed as follows: there are two paths to consider, (A, a, B, b, B) and (A, a, C, b, C). The first path has the weight $2 \cdot 2 \cdot 2 = 8$ and the second path has the weight $3 \cdot 2 \cdot 1 = 6$, so overall, the weight of the word ab in A is $8 + 6 = 14$. For weighted automata, PAWS offers an algorithm to decide which pairs of states are language equivalent, i.e. assign the same weight to all words. In general, this problem is undecidable, but for many specific semirings (e.g. the reals) it is decidable.

PAWS offers two algorithms to decide language equivalence. The more generally applicable algorithm, called *Language Equ.(Complete)* in the tool, can be applied to any semiring and yields, if it terminates, a complete characterisation of language equivalence in the automaton. This algorithm is a generalised partition refinement algorithm, based on the final chain construction in coalgebra. The idea is to enumerate the weights that each state assigns to all words, starting from the empty word ε. Whenever a word is found whose corresponding weight vector is a linear combination of previously explored words, the vector can be dropped and no words extending this word need to be explored. As soon as all branches terminate, the algorithm terminates. Then, the language equivalent states are exactly those which assign the same weight to all words that were explored. It is worth noting that, in contrast to traditional partition refinement algorithms, this algorithm may require additional steps after the final partition of the state set is already established. This is to be expected though, because of the undecidability of language equivalence in general.

For complete and completely distributive lattices, and more generally, completely distributive l-monoids, an optimisation up to congruence is available (*Language Equ.(Up-To)*). This algorithm checks whether two initial vectors are language equivalent by building a language equivalence relation on vectors over the semiring and pruning the relation by congruence closure, i.e. pairs of vectors that are already in the congruence closure of previously found pairs of vectors, are discarded. When compared to *Language Equ.(Complete)*, additional optimisation occurs in the form of pruning additionally vector pairs that are redundant because of symmetry or transitivity. Moreover, the algorithm may perform much better than *Language Equ.(Complete)* because only a pair of vectors is being compared, instead of all pairs of states, so in particular, a counterexample to language equivalence may quickly be identified.

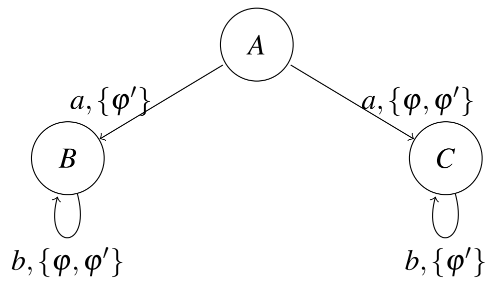
One semiring, where language equivalence is undecidable is the tropical semiring over the natural numbers $(\mathbb{N}_0 \cup \{\infty\}, \mathsf{min}, +, \infty, 0)$, as shown by Krob [16]. However, the universality problem is decidable in this case. The universality problem asks, for any given number $n \in \mathbb{N}_0$, whether the weight of all words from a given initial vector is bounded by n.

All algorithms for these problems require a method for solving linear equations over the semiring.

PAWS also analyses a second type of systems: conditional transition systems (CTS) [4, 1]. CTS are defined over a finite partial order of conditions. Each transition is assigned to a downwards-closed set of conditions under which the transition may be taken. Before execution, one condition is fixed and all transitions that carry the respective condition remain active and all other transitions remain inactive. Afterwards, the CTS evolves just like a traditional labelled transition system. At any point though, a change of conditions can occur by going down in the order. If the condition is changed, so do the active transitions and additional transitions may become available. Note that due to the requirement that all transitions carry a downwards-closed set of conditions, only additional transitions can appear, no transition can be deactivated by performing a change in conditions. Conditions can intuitively be considered as different versions of a program, where a smaller condition signifies an improved product to which a user may upgrade at any point. Using this interpretation, CTS can be used to model and analyse such classes of systems in particular for their behaviour when upgrades occur.

Also note the strong relation of partial orders to complete distributive lattices via the Birkhoff duality [11]: Every finite partial order gives rise to a lattice by considering all downwards-closed sets of elements ordered by inclusion. And vice versa, every finite distributive lattice can be represented in this way. We will use this duality to obtain compact representations of CTS.

Now consider the following CTS defined over the set of conditions $\{\varphi, \varphi'\}$, where $\varphi' \leq \varphi$.



Furthermore assume we are starting in state A under the initial condition φ. Then only the transitions from B to B and from A to C are available. If we choose to make a step from A to C via the a-labelled edge, we cannot do any further steps, unless we first perform an upgrade to φ', which allows us to use the b-loop in state C. Alternatively, starting in A, we could also perform an upgrade right away and gain the option to take an a transition to B instead of transitioning to C.

For CTS we are interested in conditional bisimulation. Two states are conditionally bisimilar, if there exists a conditional bisimulation relating the states. A conditional bisimulation is a family of traditional bisimulations R_φ, one for each condition φ, on the respective underlying transition systems. For two conditions $\varphi' \leq \varphi$ it must hold that $R_{\varphi'} \supseteq R_\varphi$, which intuitively means that if two states are bisimilar under φ, they must also be bisimilar under every smaller condition φ'. Furthermore, the standard transfer property for bisimulations must be satisfied.

In [4] we have shown how to model a small adaptive routing protocol as CTS.

Bisimulation for CTS is computed with a modified partition refinement algorithm akin to the partition refinement algorithm to compute bisimilarity for labelled transition systems. This algorithm uses the duality between finite partially ordered sets and finite distributive lattices to compute and propagate in parallel the partition refinement for all conditions at once. The BDD based-implementation uses the same basic algorithm, but represents lattice elements via BDDs, which can often lead to a very compact representation, which in turn makes the algorithm viable for significantly larger sets of conditions.

Summarizing, the problems PAWS solves and the corresponding algorithms and semirings are displayed in the following table:

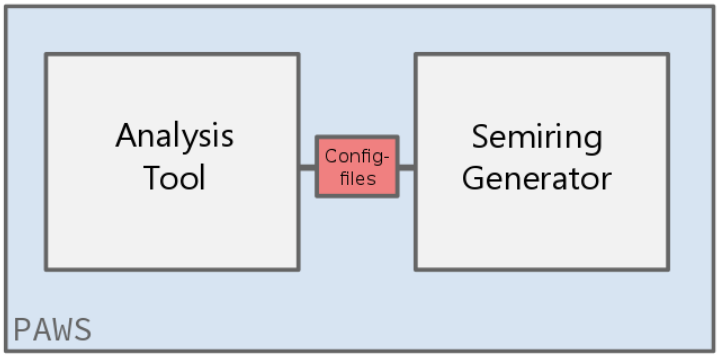
Problem	Algorithm	Semiring	Model
Language equivalence (all pairs)	*Language Equ.(Complete)*	any semiring	WA
Language equivalence (initial vectors)	*Language Equ.(Up-To)*	l-monoids, lattices	WA
Universality Problem	*Universality*	tropical Semiring	WA
Conditional bisimilarity	*CTS Bisimilarity*	finite lattice	CTS

3 Design and Usage

In this section, we give an overview of some design decisions and the usage of the tool. First, we mention the basic structure of the tool and then discuss some of the problems and math-related features of the tool. Furthermore we explain how to work with PAWS.

3.1 Design

PAWS is a Windows tool offering a complete graphical interface, developed in Microsoft's Visual Studio using C#. The program is divided into two autonomous components:



Both components are designed according to the MVC (Model View Controller) pattern. In the sequel, we will discern these two program parts, as their interaction is rather limited, allowing them to be considered as separate concerns.

3.1.1 The Semiring Generator

The development of the semiring generator started in a master's thesis [18]. It supports five different generation processes, which, for clarity, are equipped with five separate input forms. The semiring generator supports three fully automated cases:

- Direct products
- Fields of fractions
- Field extensions for $\mathbb{Q}$

Furthermore, there are two options to generate code based on user implementations:

- l-monoids
- Arbitrary semirings

Note that generation of finite lattices and modulo rings is less involved and is done directly within the analysis component.

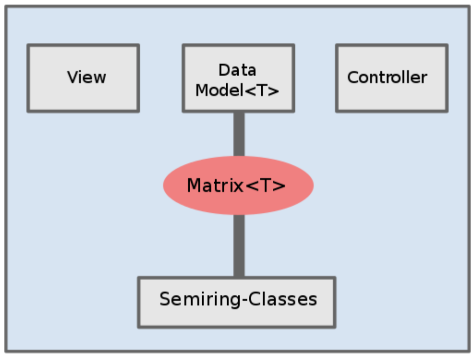
For the generation of source code we use *CodeDOM* of the *.NET* Framework, which enables code generation based on object graphs. The five processes are implemented via one superclass and three derived classes. The superclass contains all methods for creating if-statements, for-loops or useful combinations of these, based on predefined patterns. Except for the direct product generator, every class uses the constructor generating methods of the superclass. Most of the differences between the classes are reflected in the methods for generating the binary operators. Concerning methods for solving linear equations we are offering two templates: a straightforward implementation of the Gaussian algorithm and one method for l-monoids, based on the residuum operator [9].

With code generation, one always has to face the question of how to give the program, specifically the analysis component of PAWS, access to the newly generated classes. We decided to use the *Microsoft.Build.Execution* namespace for updating the analysis tool. This decision avoids creating multiple DLL files, which would be the case with a pure reflection-based solution to the problem. However, reflection is used to solve another difficulty. Due to the combination of different semirings or data types as elements of a new semiring, a dynamic approach is required to enable the automated generation of constructors with a string parameter, specifying the semiring value.

PAWS also manages the names of the semiring classes in individual text files. First, this prevents a user from overwriting an already fixed class name. Another advantage is that by using *System.Activator*, an instance can be created dynamically during runtime without knowing the class name at the level of program design. Hence, both components use consistent config files, which is ensured by updating the corresponding config file if one of the program parts creates or deletes a semiring. But when deleting a semiring that has already been used to create and store an automaton or transition system, conflicts may occur with the serialised objects and thus with the user's storage files. Therefore, deletion of semirings must be dealt with separately. A semiring can only be deleted or modified if it has not been used in the previous session within an automaton.

3.1.2 The Analysis Tool

As already mentioned, for both program parts MVC is used to implement the user interface. The main focus of PAWS is to give the user the tools to define an automaton in order to be able to subsequently analyse it with the supported algorithms. In order to implement this as dynamically as possible, we have



opted for a generic implementation. Therefore, the class $Matrix\langle T\rangle$ (where T is the generic type of the semiring), which implements automata in a matrix representation, is the core of the tool's architecture. The main argument for the generic approach is that the algorithms for the analysis of an automaton are based on the basic operators of a semiring. A further method is needed to solve systems of linear equations, which is also defined for each semiring within PAWS. It is therefore recommended to use a generic class that includes all the methods of analysis, which in turn dynamically call the corresponding operators or methods of the currently used semiring. This dynamic approach is thus combined with reflection.

Consequently, the management of automata created by the user requires a generic implementation as well. The model for the matrices is therefore also generic and thus it makes sense to manage several semiring models by the controller.

Because of the generic approach, automatically checking the correctness of the user input proves to be problematic when the user has generated his or her own semiring and has implemented a string constructor to read semiring elements from input strings without input verification. In this case PAWS can not check whether the input is well-formed, this has to be taken care of in the user-implemented method. Such semirings can not be further used to generate other semirings.

A further design decision is that one can create finite semirings directly within the analysis component of PAWS. The reason why we chose to not move this option to the generator is that for finite lattices and modulo rings over the integers no new classes have to be generated nor is there any need for new source code at all. In this case, it is sufficient to configure template classes for the corresponding semirings via static variables that contain the required information about the semiring. The operators are designed to behave according to the configuration of the class. In such cases, the analysis program must also access the configuration files in order to make the extensions known to the semiring generator.

As an additional feature, we have also integrated GraphViz[2] into the tool, as it allows visualization of weighted automata, if desired by the user.

3.2 Usage

We will discuss the usage of the tool separately for the two individual components of PAWS, hence this section contains subsections giving details about the following two components.

- The *semiring generator* to build and provide the required semirings over which automata can be defined. This generator is used to generate semirings that cannot be obtained in a fully automated way and supports some form of automatic generation.

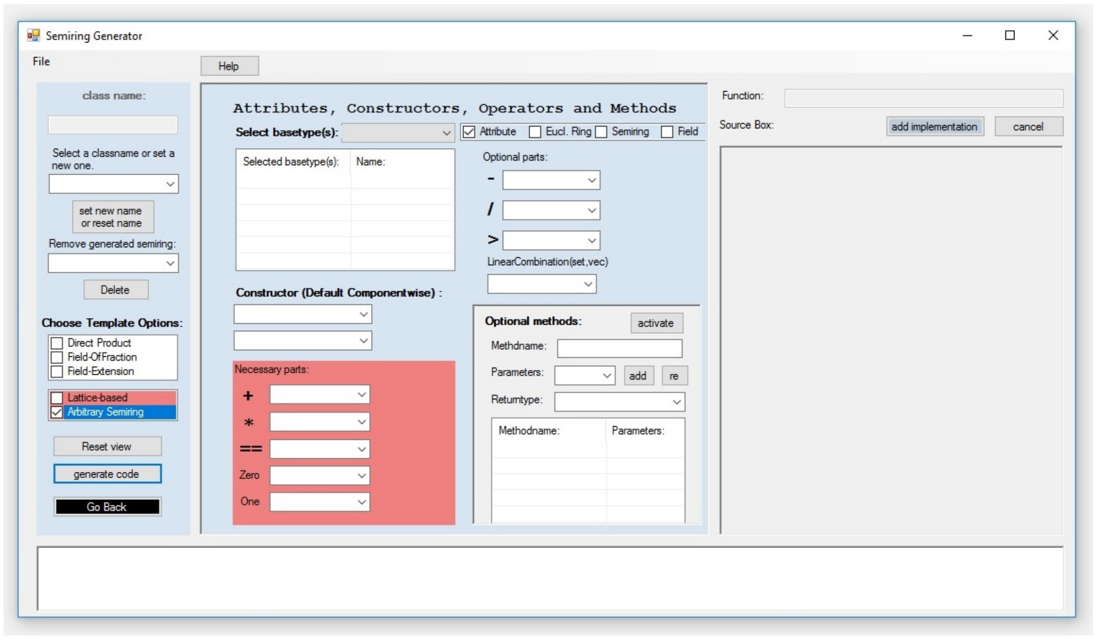
[2] `www.graphviz.org`

- The *analysis tool* that allows the user to choose a previously generated semiring, one of the semirings that come built-in with the PAWS or to build a lattice and then to define automata in a matrix representation over those lattices. Matrices are then interpreted as weighted automata or conditional transition systems (CTS) and can be used to compute language equivalence for weighted automata with two different approaches, decide the threshold problem for weighted automata over the tropical semiring of natural numbers or to compute the greatest bisimilarity of a CTS.

3.2.1 The Semiring Generator

The semiring generator is used to generate the semirings under consideration. In order for a semiring to be usable within the context of PAWS, the structure needs to define the following components:

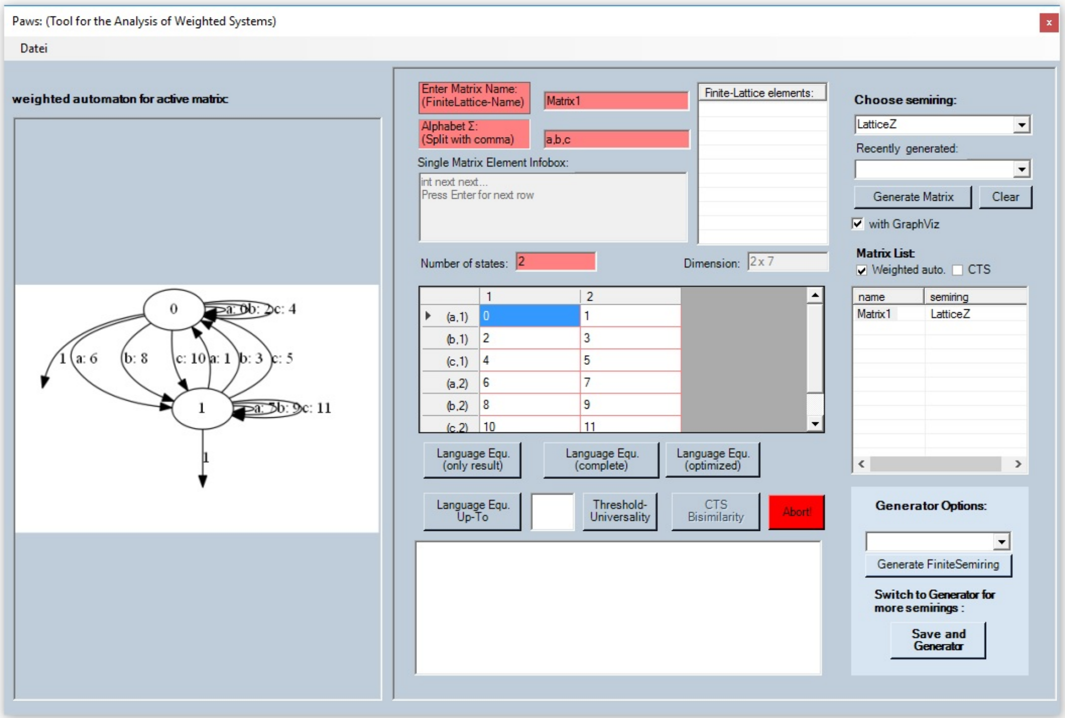
- A universe which contains all elements of the semiring. All predefined datatypes from C#, as well as combinations of them can serve as universes.
- An addition operator $+$ of the semiring.
- A multiplication operator $*$ of the semiring.
- One() and Zero() methods, which return the units of addition and multiplication.
- A method for solving linear equations over the semiring.



While the first four components are necessary to define a semiring in a mathematical context either way, the procedure to solve linear equations is an additional requirement – one that PAWS aims at reducing as much as possible – but in order to provide the greatest amount of flexibility possible, the tool offers the option to define a procedure to solve linear equations from scratch.

3.2.2 The Analysis Tool

The analysis component is designed to offer generic algorithms applicable to numerous predefined or user-defined semirings. Some of the algorithms can however only be used with specific (types) of semirings. The most general algorithm is the language equivalence check, for which all semirings are eligible. Conditional transition systems are only defined over lattices, therefore, the bisimulation check is limited to lattice structures. However, the user still has the choice between two different ways of dealing with lattices: representing elements of the lattice as downwards-closed sets of irreducibles via the Birkhoff duality [11], applicable to all finite distributive lattices, or representing them using binary decision diagrams (BDDs). The BDD variant is more restrictive and mainly designed for the application to CTS. Here, the irreducibles are required to be full conjunctions of features from a base set of features, ordered by the presence of distinct upgrade features. Lastly, the threshold check can only be performed over a single semiring, the tropical semiring over natural numbers.



Enter Matrix Name: (FiniteLattice-Name) Matrix1

Alphabet Σ: (Split with comma) a,b,c

Single Matrix Element Infobox: int next next... Press Enter for next row

Number of states: 2

Dimension: 2 x 7

Choose semiring: LatticeZ

Recently generated:

[✓] with GraphViz

Matrix List: [✓] Weighted auto. [ ] CTS

	1	2
(a,1)	0	1
(b,1)	2	3
(c,1)	4	5
(a,2)	6	7
(b,2)	8	9
(c,2)	10	11

name	semiring
Matrix1	LatticeZ

The general workflow of the analysis tool is as follows:

▷ Choose a semiring

▷ Generate a matrix over this semiring, representing a weighted automaton or a conditional transition system

Alternatively: Choose the matrix from a list of matrices that have been generated previously

▷ Start the algorithm and provide – if necessary – additional input

Additional input comes in two forms: initial vectors and the threshold to be checked against in case of the threshold algorithm. Depending on the semiring of choice, questions regarding language equivalence might not be decidable, leading to non-termination of the corresponding procedure in PAWS. In order to deal with this problem and to allow abortion of an overlong computation, the actual computation is delegated to a separate thread that can at any time be aborted. In that case all intermediate results are discarded.

Note that only the two language equivalence-based algorithms can run into non-termination issues. For the CTS bisimulation check, as well as the threshold problem on the tropical semiring of natural numbers, termination is always guaranteed. However, the runtime of CTS bisimulation check can be doubly exponential in the number of features under consideration – because the lattice is the set of all possible configurations, which in turn are all possible conjunctions over the features. Using the BDD-based implementation of lattices – which is particularly suited to the needs of CTS modelling – this explosion is mitigated in many cases, but it can not be ruled out completely. On the other hand, the BDD-based modelling only allows for special lattices to be modelled, i.e. those that arise as the lattices constructed from a set of features and upgrade features, whereas the variant called *FiniteLattice* allows for arbitrary (finite, distributive) lattices to be represented. In this case, lattices are represented via the partial order of irreducible elements, using Birkhoff's representation theorem [11].

3.3 Evaluation

Here we describe several case studies and list runtime results. Before we discuss the results, let us first consider the theoretical complexity results, listed in Table 1, where we consider various semirings (in the left column) and various problems (in the top row). In the case of the reals the considered order for the threshold (or universality) problem is $\geq$.

	equivalence	inclusion	universality
reals	P [21]	undecidable	undecidable ($\geq$) [19]
tropical semiring	undecidable [16]	undecidable	PSPACE-cmpl. [2]
distr. lattices	PSPACE-cmpl.	PSPACE-cmpl. [17]	PSPACE-cmpl.

Table 1: Overview of the corresponding complexity classes for considered weighted automata problems.

The main focus of PAWS is on weighted automata and language equivalence, hence we will conduct our experiments for this problem. For extensive experiments concerning the universality problem see [5]. Experiments on conditional transition systems were conducted in [4].

We will consider the decidable cases, namely the reals, as well as another (finite) ring, and a distributive lattice. More concretely, we will evaluate the algorithm *Language Equ.(Complete)* on three different semirings: the reals $(\mathbb{R}, +, \cdot, 0, 1)$, the finite ring $(\mathbb{Z}_{100}, +_{100}, \cdot_{100}, 0, 1)$ (where addition and multiplication is modulo 100), and *LatticeZ*, the lattice $(\mathbb{Z}, \min, \max, \infty, -\infty)$ restricted to the interval $[-1000, 1000]$. Note that *Language Equ.(Complete)* determines for all pairs of states whether they are equivalent.

PAWS was executed on an Intel Core 2 Quad CPU Q9550 at 2.83 GHz with 4 GB RAM, running Windows 10. Our test cases were constructed as follows: we considered different numbers of states ($|X|$) and also varied the probability of the existence of a transition between a pair of states (p_{Tr}). Usually we set $p_{Tr} = 0.5$, but we will also compare to other values. Weights are set randomly and the default alphabet size was 2.

For each semiring and each chosen combination of parameters we randomly generated 1000 weighted automata and observed the runtime behaviour. We considered various percentiles, for instance the 50%

and 90% percentiles: the 50% percentile is the median and the 90% percentile means that 90% of the runs were faster and 10% slower than the time given in the respective field. Analogously for the other percentiles.

As expected, the runtimes are heavily dependent on the semiring. This is due to two reasons: first, the algorithms *Language Equ.(Complete)* internally uses procedures for solving linear equations and the complexity of those procedures, which depend on the semiring, plays a major role in the runtime. Second, as explained in Section 2, the algorithm lists words of increasing length and associates them with weight vectors. A vector need not be considered if it is a linear combination of existing vectors. Hence, if the corresponding semimodule has few generators – as it is for instance the case for a vector space of limited dimension over a field – the search space can be cut off quickly.

We first consider the two rings: the reals and $\mathbb{Z}_{100}$ (see Table 2). For the reals, the Gaussian algorithm has a polynomial runtime $\mathscr{O}(|X|^3)$ in the number of states. Due to the fact that the search can be cut off after at most $|X|$ vectors, there are only minor variations in the runtime over the various percentiles. Furthermore the alphabet size and the probability p_{Tr} have little impact on the runtime. Instead the times are mainly dependent on the number of states.

$(\|X\|, p_{Tr})$	Semiring	50%	90%	95%
(10,0.5)	$\mathbb{R}$	28	29	30
	$\mathbb{Z}_{100}$	42	73	2307
(15,0.5)	$\mathbb{R}$	98	100	101
	$\mathbb{Z}_{100}$	252	263	267
(20,0.5)	$\mathbb{R}$	359	361	362
	$\mathbb{Z}_{100}$	518	523	528
(25,0.5)	$\mathbb{R}$	712	717	719
	$\mathbb{Z}_{100}$	1034	1058	1081
(30,0.5)	$\mathbb{R}$	1327	1333	1336
	$\mathbb{Z}_{100}$	1983	3509	3548
(35,0.5)	$\mathbb{R}$	3648	3660	3664
	$\mathbb{Z}_{100}$	5242	5290	5304
(40,0.5)	$\mathbb{R}$	5183	5197	5202
	$\mathbb{Z}_{100}$	7654	7924	7960
(45,0.5)	$\mathbb{R}$	7400	7419	7425
	$\mathbb{Z}_{100}$	11492	11538	11551
(50,0.5)	$\mathbb{R}$	10471	10509	10720
	$\mathbb{Z}_{100}$	15673	15720	15738

Table 2: Runtime results in milliseconds on randomly generated automata over two different semirings

There are different effects for the ring $\mathbb{Z}_{100}$, which also uses the Gauss method, but the runtime is particularly influenced by the subsequent application of the Hensel lifting, which depends on the one hand on the prime factor decomposition $q = \prod_{i=0}^{k} p_i^{e_i}$, where k is the number of prime factors (here: $q = 100$), and second on the number of zero lines that occur in computation of the row echelon form. It is not previously known which combination of the free variables leads to a solution [10, 18]. The cases in which all those possibilities have to be enumerated are very rare, however, these cases do occur and they may cause a timeout after 5 hours. In order to see this effect, consider the 99.9% percentiles in Table 3. An additional, very interesting, observation, which we have made while testing the implementation, is that the worst case is less frequent in practice with an increasing number of states (see Table 3). A possible explanation for this could be the decreasing probability for the occurrence of zero lines with an increase in the number of states.

The algorithm is much less efficient in the case of the lattice of integers *LatticeZ* (see Table 4), although the procedure for solving a single solving equation system, based on the residuum operation [9], has linear runtime. However, this does not help, since we can not guarantee to find a small set of generators in the semimodule and hence there is no limit on the number of vectors which are enumerated. Already for $|X| = 15$ states it is infeasible to obtain results. The situation becomes worse when the alphabet size increases, since the number of vectors will also grow. This is different for the case of the reals and $\mathbb{Z}_{100}$, since there the states limit the number of columns.

$(\|X\|, p_{Tr})$	Semiring	50%	85%	90%	95%	99%	99.9%
(10,0.5)	$\mathbb{Z}_{100}$	42	68	73	2307	7041	time-out
(15,0.5)	$\mathbb{Z}_{100}$	252	258	263	267	897	1787758
(20,0.5)	$\mathbb{Z}_{100}$	518	521	523	528	532	time-out
(25,0.5)	$\mathbb{Z}_{100}$	1034	1044	1058	1081	1821	time-out
(30,0.5)	$\mathbb{Z}_{100}$	1983	3493	3509	3548	3745	3881
(35,0.5)	$\mathbb{Z}_{100}$	5242	5283	5290	5304	5422	5501

Table 3: Worst case runtime occurrence of *Language Equ.(Complete)* over the finite semiring $\mathbb{Z}_{100}$.

$(\|X\|, p_{Tr})$	Semiring	50%	85%	90%	95%	99%
(5,0.5)	*LatticeZ*	41	242	347	758	3186
(10,0.5)	*LatticeZ*	18808	287920	564525	1975251	16321319

Table 4: Runtimes in milliseconds of *Language Equ.(Complete)* on randomly generated automata over *LatticeZ*.

Note, that the performance of algorithm *Language Equ.(Complete)* is bad in the case of *LatticeZ*, the on-the-fly up-to technique *Language Equ.(Up-To)* performs much better. At first sight, such a comparison seems unfair, since *Language Equ.(Up-To)* tests language equivalence only for a single pair of states, whereas the other method considers all pairs of states at once. However, the advantage is significant enough that the up-to technique can be expected to be more performant even when it is applied to all pairs of states, one after the other. In fact, in the case of randomly generated automata with ten states, there are 90 pairs of states to consider and the running time we have recorded for *Language Equ.(Up-To)* is lower than 38 milliseconds in 99.3% of all cases. Thus, even if we take the value of the 99.3% percentile for all pairs of states, we end up with $90 \cdot 38$ milliseconds, which is significantly lower than the median runtime for *Language Equ.(Complete)*.

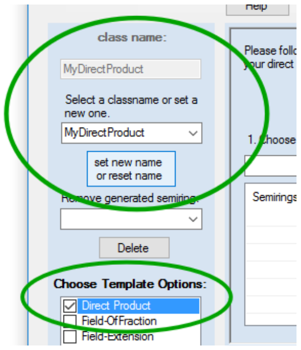
For randomly generated automata, it is rare to find two language-equivalent states, in fact, only very few steps are typically required to find a word that separates randomly generated automata. While *Language Equ.(Up-To)* may stop its computation after finding a single word separating the initial pair of states, it is not sufficient for termination of *Language Equ.(Complete)* to find witnesses of absence of language equivalence. The intermediate results of *Language Equ.(Complete)* also need to span the same semimodule for termination of the algorithm, which may take numerous extra steps after all non-equivalent states have been separated.

4 GUI Overview of PAWS

In this section, we demonstrate the handling of the various features and options that PAWS offers. First, we will present the semiring generator. In this case, we briefly explain a fully automated generation and the even more complex case, in which, apart from the constructor, the user has to specify the implementations on his or her own. We then illustrate the various possibilities to use the analysis component of PAWS in a short overview.

4.1 The Semiring Generator

First, we consider a fully automated generation demonstrated by the direct product input mask (Figure 1 and 2). Then, in Figure 3 the console informs the user that the generated source code is compilable. In case the user specifies some code on his or her own, the console will display suitable compilation error messages. Figures 4 and 5 depict the use of the input mask for user-dependent implementations.



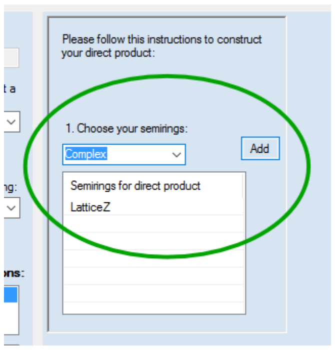
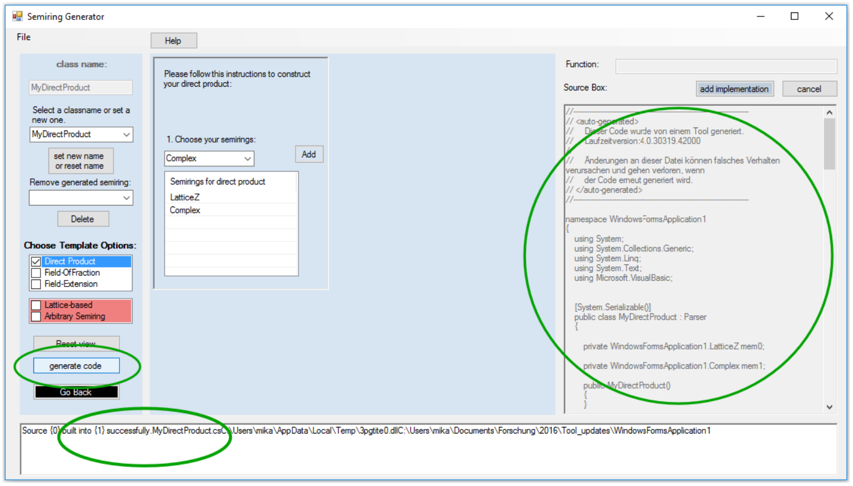

Please follow this instructions to construct your direct product:
1. Choose your semirings:
Complex
Add
Semirings for direct product
LatticeZ

Figure 1: Lower ellipse: Choose an input mask. Top ellipse: Specify the name of the new semiring class.

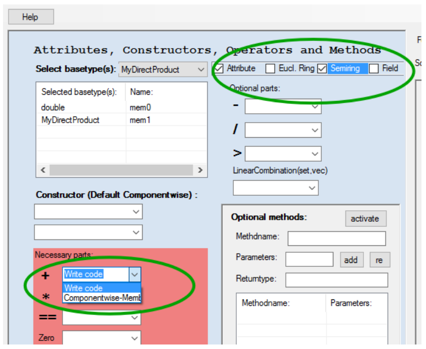
Figure 2: Specify a direct product, add semirings as member fields to the class.



```
//------------------------------------------------------------------------------
// <auto-generated>
//     Dieser Code wurde von einem Tool generiert.
//     Laufzeitversion:4.0.30319.42000
//
//     Änderungen an dieser Datei können falsches Verhalten
verursachen und gehen verloren, wenn
//     der Code erneut generiert wird.
// </auto-generated>
//------------------------------------------------------------------------------

namespace WindowsFormsApplication1
{
   using System;
   using System.Collections.Generic;
   using System.Linq;
   using System.Text;
   using Microsoft.VisualBasic;


   [System.Serializable()]
   public class MyDirectProduct : Parser
   {

      private WindowsFormsApplication1.LatticeZ mem0;

      private WindowsFormsApplication1.Complex mem1;

      public MyDirectProduct()
      {
      }
```

```
Source {0} built into {1} successfully.MyDirectProduct.csC:\Users\mika\AppData\Local\Temp\3pgtite0.dllC:\Users\mika\Documents\Forschung\2016\Tool_updates\WindowsFormsApplication1
```

Figure 3: Pressing the circled button will generate the code visible in the right-hand text-box. The console at the bottom informs the user, whether the source code was compiled and successfully added to the analysis component of PAWS.


Help
Attributes, Constructors, Operators and Methods
Optional parts:
-
/
>
LinearCombination(set,vec)
Constructor (Default Componentwise) :
Necessary parts:
+
*
==
Zero
Optional methods:
activate
Methdname:
Parameters:
add
re
Returntype:

Select basetype(s): MyDirectProduct

☑ Attribute ☐ Eucl. Ring ☑ Semiring ☐ Field

Selected basetype(s):	Name:
double	mem0
MyDirectProduct	mem1

+ Write code

Write code
Componentwise-Memb

Methodname:	Parameters:

Figure 4: Top ellipse: Specify the type of the class. Bottom ellipse: Select *write code* to implement the addition (+).

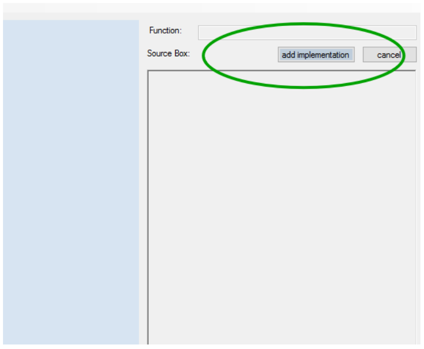
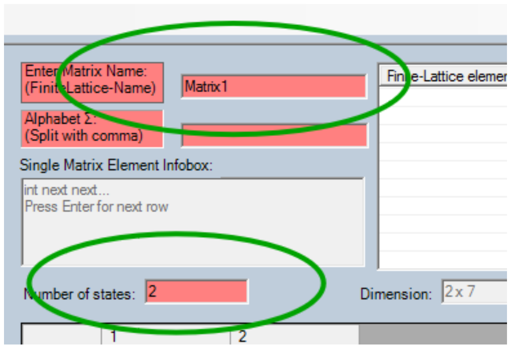


Figure 5: The text box for entering source code will be enabled after selecting *write code* as shown in Figure 4.

4.2 The Analysis Tool

In this section the use of the PAWS analysis component is presented in a brief overview. The illustrations serve to explain our intuition behind the design and the use of PAWS.

In Figures 6 and 7 the first steps for creating a weighted automaton are illustrated. After generating a matrix (Figures 8 and 9), the user can choose one of the available algorithms and wait until the result of the computation is displayed inside the text area (Figure 10).



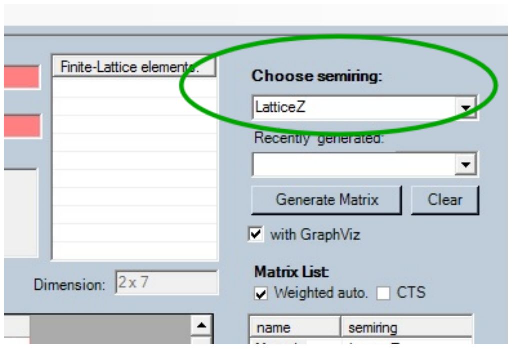
Enter Matrix Name: (FiniteLattice-Name) Matrix1

Alphabet Σ: (Split with comma)

int next next...
Press Enter for next row

Number of states: 2

Dimension: 2 x 7



Choose semiring: LatticeZ

☑ with GraphViz

☑ Weighted auto. ☐ CTS

Dimension: 2 x 7

Figure 6: Top ellipse: First determine the name. Bottom ellipse: Type in the number of states.

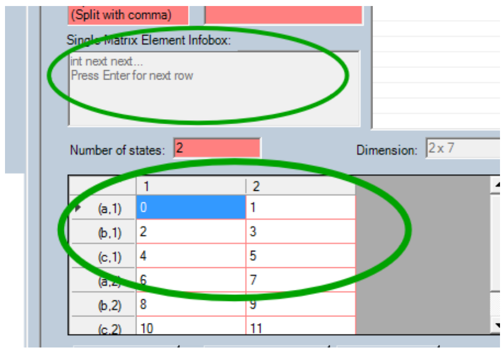
Figure 7: Choose the semiring.

With the PAWS analysis component, besides automata, also finite semirings can be generated and stored for further semiring generation as well as for the analysis.

5 Conclusion, Future Work and Related Work

We have seen that PAWS is a flexible tool to analyse the behaviour of weighted automata and conditional transition systems. The generic approach allows for adding new semirings with a varying degree of support by the tool itself.

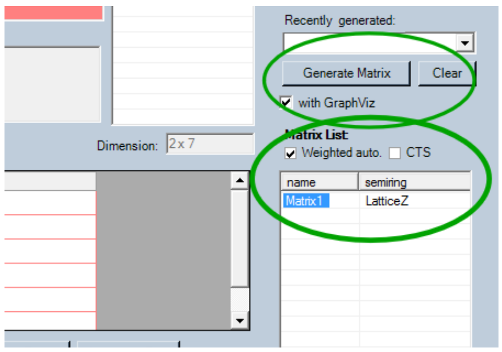
Concerning related approaches, we are not aware of analysis tools for language equivalence and the threshold problem for weighted automata.



Number of states: 2 Dimension: 2x7

	1	2
(a,1)	0	1
(b,1)	2	3
(c,1)	4	5
(a,2)	6	7
(b,2)	8	9
(c,2)	10	11

Figure 8: After a semiring has been selected, a description of the expected input is shown above the input area for the matrix.

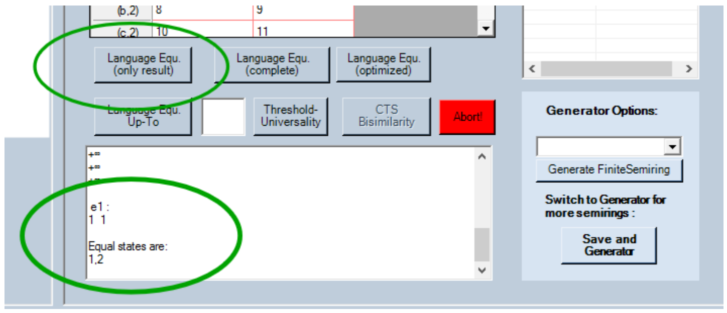


Dimension: 2x7

[✓ Weighted auto.] CTS

name	semiring
Matrix1	LatticeZ

Figure 9: By pressing the button *Generate Matrix*, the matrix is generated in the tool and listed according to its form (Weighted automaton or CTS).



```
+=
+=
e1 :
1 1
Equal states are:
1,2
```

Figure 10: After starting one of the supported algorithms for a transition system, the analysis result will be displayed in the bottom text area.

For the problem of generating semirings dynamically, there exists previous work for solving fixpoint equations over semirings by Esparza, Luttenberger and Schlund [13]. In [13] FPSOLVE is described, a C++ template based tool for solving fixpoint equations over semirings. That is, the tool has a different application scenario than ours. However, the tools share similarities since in FPSOLVE the user also has the possibility to generate new semirings. For this, only the addition, multiplication and Kleene star must be implemented. However, a string constructor must also be specified without automatic support and the main method must be adjusted with the corresponding command-line. PAWS is designed to enable the generation of new semirings for solving linear equations using a graphical user interface and does not change already existing code, which is not part of a semiring class.

In contrast to this, work has already been done on an analysis tool for featured transition systems – which are basically CTS without a notion of upgrades – to analyse software product lines wrt. simulation. In their work [8], Cordy et al. have implemented their model using BDDs as well, yielding a similar speed up as our own approach. The significant differences here lie in the notion of behaviour, since Cordy et al. have focused on simulation relations, whereas we are concerned with bisimulations. Furthermore we capture a notion of upgrade and thus support partial orders of products instead of just abritrary sets of products.

We intend to develop PAWS further in several ways. We are looking for new classes of semirings where solutions of linear equations can be effectively computed, in order to equip those semirings with an improved support from PAWS. Furthermore, we are interested in analysing more extensive case studies, where we will use PAWS to conduct all required analyses.

References

[1] Jiří Adámek, Filippo Bonchi, Mathias Hülsbusch, Barbara König, Stefan Milius & Alexandra Silva (2012): *A Coalgebraic Perspective on Minimization and Determinization*. In: Proc. of FOSSACS '12, Springer, pp. 58–73. LNCS/ARCoSS 7213.

[2] Shaull Almagor, Udi Boker & Orna Kupferman (2011): *What's Decidable about Weighted Automata?* In: Proc. of ATVA '11, Springer, pp. 482–491. LNCS 6996.

[3] Mariel-Pierre Béal, Slyvain Lombardy & Jacques Sakarovitch (2006): *Conjugacy and Equivalence of Weighted Automata and Functional Transducers*. In: Prof. of CSR '06, Springer, pp. 58–69. LNCS 3967.

[4] Harsh Beohar, Barbara König, Sebastian Küpper & Alexandra Silva (2016): *Conditional Transition Systems: A Model for Software Product Lines with Upgrades*. Unpublished.

[5] Filippo Bonchi, Barbara König & Sebastian Küpper (2017): *Up-To Techniques for Weighted Systems*. In: Proc. of TACAS '17, Part I, Springer, pp. 535–552. LNCS 10205.

[6] Filippo Bonchi & Damien Pous (2013): *Checking NFA equivalence with bisimulations up to congruence*. In: Proc. of POPL '13, ACM, pp. 457–468.

[7] Michele Boreale (2009): *Weighted bisimulation in linear algebraic form*. In: Proc. of CONCUR '09, Springer, pp. 163–177. LNCS 5710.

[8] Maxime Cordy, Andreas Classen, Gilles Perrouin, Pierre-Yves Schobbens, Patrick Heymans & Axel Legay (2012): *Simulation-based abstractions for software product-line model checking*. In: Prof. of ICSE '12, pp. 672–682.

[9] Raymond A. Cuninghame-Green (1979): *Minimax algebra*. Lecture Notes in Economics and Mathematical Systems, Springer-Verlag.

[10] Abhijit Das & C. E. Veni Madhavan (2009): *Public-Key Cryptography: Theory and Practice*. Pearson Education. pp. 295-296.

[11] Brian A. Davey & Hilary A. Priestley (2002): *Introduction to lattices and order*. Cambridge University Press.

[12] Manfred Droste, Werner Kuich & Heiko Vogler, editors (2009): *Handbook of Weighted Automata*. Springer.

[13] Javier Esparza, Michael Luttenberger & Maximilian Schlund (2014): *FPsolve: A Generic Solver for Fixpoint Equations over Semirings*. In: Proc. of CIAA '14, Springer, pp. 1–15. LNCS 8587.

[14] Stefan Kiefer, Andrzej S. Murawski, Joel Ouaknine, Björn Wachter & James Worrell (2011): *Language Equivalence for Probabilistic Automata*. In: Proc. of CAV '11, Springer, pp. 526–540. LNCS 6806.

[15] Barbara König & Sebastian Küpper (2016): *A generalized partition refinement algorithm, instantiated to language equivalence checking for weighted automata*. Soft Computing, pp. 1–18, doi:10.1007/s00500-016-2363-z. Available at `http://dx.doi.org/10.1007/s00500-016-2363-z`.

[16] Daniel Krob (1994): *The equality problem for rational series with multiplicities in the tropical semiring is undecidable*. International Journal of Algebra and Computation 4(3), pp. 405–425.

[17] Orna Kupferman & Yoad Lustig (2010): *Latticed Simulation Relations and Games*. International Journal of Foundations of Computer Science 21(02), pp. 167–189.

[18] Christine Mika (2015): *Ein generisches Werkzeug für Spracháquivalenz bei gewichteten Automaten*. Master's thesis, Universität Duisburg-Essen.

[19] A. Paz (1971): *Introduction to Probabilistic Automata*. Academic Press, New York.

[20] Marcel-Paul Schützenberger (1961): *On the Definition of a Family of Automata*. Information and Control 4(2–3), pp. 245–270.

[21] W. Tzeng (1992): *A polynomial-time algorithm for the equivalence of probabilistic automata*. SIAM Journal on Computing 21(2), pp. 216–227.