How to Tackle Integer Weighted Automata Positivity

Y. Boichut[1], P.-C. Héam[2,3] and O. Kouchnarenko[3]

LIFO/University of Orléans[1]
LSV INRIA/CNRS/ENS Cachan[2]
INRIA/CASSIS and LIFC/University of Franche-Comté [3]

Abstract. This paper is dedicated to candidate abstractions to capture relevant aspects of the integer weighted automata. The expected effect of applying these abstractions is studied to build the deterministic reachability graphs allowing us to semi-decide the positivity problem on these automata. Moreover, the papers reports on the implementations and experimental results, and discusses other encodings.

1 Introduction

Weighted automata is a formalism widely used in computer science for applications in images compression [21,22], speech-to-text processing [28,9] or discrete event systems [14]. These large application areas make them intensively studied from the theoretical point of view [25,31,19,24,11,23]. The expressive power of these automata is high enough so that many natural questions are not decidable. Among them the problem to know whether for a given max/+-automaton $\mathcal{A}$, every word has a positive cost, called the positivity problem, was shown to be undecidable [25]. This problem is of special interest because systems/components comparisons modelled by max/+-automata can be based on or reduced to it.

The question we are interested in is whether the automatic verification of certain properties taking costs into account is possible on max/+-automata. As the semantics of max/+-automata model is described by an infinite structure, there is a need of finite abstractions of this semantics to perform analysis fully automatically. Here the problem of handling costs becomes apparent. Obviously, this kind of finite abstractions does not exist for max/+-automata, at least not for the cost-based verification problem investigated. Given a max/+-automaton, our research focuses on methods for semi-deciding whether in the infinite structure *there are a word and a reachable configuration containing some final state reachable from an initial state of the* max/+*-automaton, with cost* -1 *at most.*

After introducing preliminary notions and recalling useful results on max/+-automata (Section 2), we briefly explain how the positivity problem can be encoded into a reachability problem (Sect. 3). Next we explain how to tackle this reachability problem using two semi-decision approaches. The first one (developed in Sect. 4) is based on a configuration space exploration using a pruning property to reduce the search. The second one (exposed in Sect. 5) uses a rewriting encoding of the problem and applies approximation techniques developed in the rewriting theoretical framework. We report on experiments with the two semi-algorithms that were implemented (Sect. 6), in particular when bounding the depth of search. Section 7 contains a discussion on possible ways to tackle remaining unsolved instances and gives some perspectives before concluding in Sect. 8. Omitted proofs are provided in Appendix.

Well-structured transition systems, or WSTSs, are a general family of transition systems where general decidability results exist [12,1]. It turns out that it is possible to give to many classes of models a structure of WSTSs [13]. We want to emphasise the fact that it is not the case for max/+-automata. Consequently, thanks to the expressivity results in [5], the determinisation reachability graphs corresponding to max/+-automata do not give rise to systems sitting inside some level of the symbolic transition systems (STS) hierarchy in [20].

In a verification context, weighted (priced) systems have been studied in many recent works (see e.g.,[**?**,**?**,**?**,**?**]). The central underlying problem of these works is to compute the optimal weight of a path to reach a given configuration (from an initial configuration); the difficulties are due to timed constraints (for locations and/or transitions). In this paper, the main difficulty lies in the quantification *for all words u*.

2 Preliminaries

In this paper, Σ denotes a finite alphabet, i.e. a finite set of symbols whose elements are called *letters*. We assume that the reader is familiar with basic language theory notions as word, language, etc. In the paper, the words *weight* and *cost* are indistinctly used.

We denote by $\overline{\mathbb{Z}}$ the set $\mathbb{Z} \cup \{-\infty\}$. Addition and max-function are classically extended to $\overline{\mathbb{Z}}$ by: for every $x \in \overline{\mathbb{Z}}$, $-\infty + x = x + -\infty = -\infty$ and $\max(x, -\infty) = \max(-\infty, x) = x$.

Definition 1. *A* max/+-automaton $\mathcal{A}$ *over Σ is a quintuplet $\mathcal{A} = (Q, \Sigma, E, I, F)$ where Q is the finite set of states, $E \subseteq Q \times \Sigma \times \mathbb{Z} \times Q$ is the set of transitions, $I \subseteq Q$ is the set of initial states, and $F \subseteq Q$ is the set of final states. Moreover, $\mathcal{A}$ satisfies the following condition: if (p, a, c, q) and (p, a, d, q) are in E, then $c = d$.*

Figure 1 gives two examples of max/+-automata. Initial states are represented with an input arrow, and final states with a double circle.

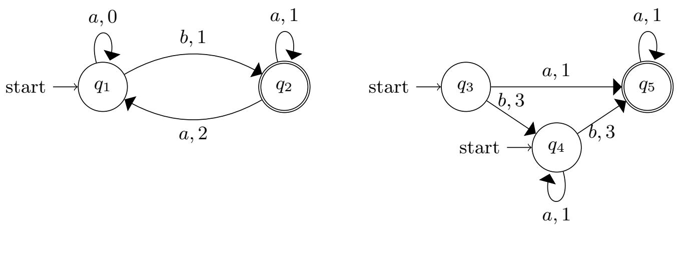

Fig. 1. max/+-automata $\mathcal{A}_{exe1}$ and $\mathcal{A}_{exe2}$

A *path* π of a max/+-automaton $\mathcal{A}$ is a finite sequence $\pi = (p_0, a_0, c_0, q_0), (p_1, a_1, c_1, q_1), \ldots, (p_n, a_n, c_n, q_n)$ of transitions of $\mathcal{A}$ such that for every $0 \leq i < n$, $q_i = p_{i+1}$. If we add the conditions: p_0 is an initial state, q_n is a final state, then we call π a *successful path.* The *label* $lab(\pi)$ of the path π is the word $a_0a_1\dots a_n$, and the *cost* of the path π is the sum of the c_i's: $\mathrm{cost}_{\mathcal{A}}(\pi) = \sum_{i=0}^{n} c_i$. The *cost* of a word u, denoted $\mathcal{A}(u)$, is the maximum of all costs of successful paths of label u: $\mathcal{A}(u) = \max\{\mathrm{cost}_{\mathcal{A}}(\pi) \mid lab(\pi) = u\}$.

Example 1. For instance, for the max/+-automaton $\mathcal{A}_{exe1}$ in Fig. 1, the word *baaab* labels the successful paths $(q_1, b, 1, q_2), (q_2, a, 2, q_1), (q_1, a, 0, q_1), (q_1, a, 0, q_1), (q_1, b, 1, q_2)$, $(q_1, b, 1, q_2), (q_2, a, 1, q_2), (q_2, a, 2, q_1), (q_1, a, 0, q_1), (q_1, b, 1, q_2)$ and $(q_1, b, 1, q_2), (q_2, a, 1, q_2), (q_2, a, 1, q_2), (q_2, a, 2, q_1)(q_1, b, 1, q_2)$. Therefore $\mathcal{A}_{exe1}(baaab) = 6$.

Notice that since u is finite, there are finitely many successful paths of label u. A max/+-automaton is finitely ambiguous if there exists an integer k such that every word accepted by the automaton is the label of k successful paths, at most. In Fig. 1, $\mathcal{A}_{exe2}$ is finitely ambiguous, whereas $\mathcal{A}_{exe1}$ is not: the word ba^nb is accepted by $n-1$ different successful paths. We end this section by recalling some useful results on decision procedures for finite (integer weighted) automata exploited in this paper.

Theorem 1. *Given a* max/+-*automaton $\mathcal{A}$, it is undecidable to test whether for every $u \in L(\mathcal{A})$, $\mathcal{A}(u) \geq 0$ [25], and polynomial time decidable whether for every $u \in L(\mathcal{A})$, $\mathcal{A}(u) \geq 0$ if $\mathcal{A}$ is finitely ambiguous [19,31].*

3 Reachability Encoding

Given an max/+-automaton $\mathcal{A}$, while it is undecidable to test whether for every $u \in L(\mathcal{A})$, $\mathcal{A}(u) \geq 0$ [25], we define a determinisation-based abstraction of the model, leading to graphs for which reachability can be semi-decided. More precisely, in this section, the operational semantics of a max/+-automaton $\mathcal{A}$ over Σ is given as a determinisation reachability graph where for a given word in Σ^*, the corresponding configuration contains the information on maximal costs for reaching states of $\mathcal{A}$.

Let $\mathcal{A} = (Q, \Sigma, E, I, F)$ be a max/+-automaton. The *determinisation graph* $\mathcal{G}(\mathcal{A}) = (V, \delta, s_0, K)$ of $\mathcal{A}$ is defined by

- $V = \overline{\mathbb{Z}}^Q$, $s_0 \in V$ and $s_0(p) = 0$ if $p \in I$, and $s_0(p) = -\infty$, otherwise;
- $\delta \subset (V \times \Sigma) \times V$ is the function defined $\delta(s, a) = s'$ iff $s'(p) = \max\{s(q) + c \mid (q, a, c, p) \in E\}$, with the convention that $\max \emptyset = -\infty$;
- $K = \{s \in V \mid \exists q \in F,\ s(q) \neq -\infty$ and $\forall p \in F,\ s(p) < 0\} \subseteq V$.

Example 2. Let us consider for instance the automaton $\mathcal{A}_{exe3}$ depicted in Fig. 2.

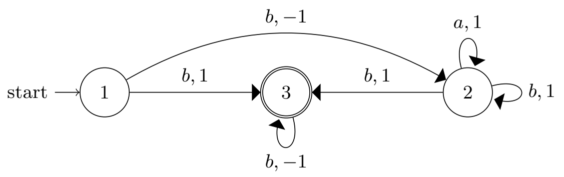

Fig. 2. max/+-automata $\mathcal{A}_{exe3}$

An element s of $\overline{\mathbb{Z}}^{\{1,2,3\}}$ is denoted (x,y,z) if $s(1)=x$, $s(2)=y$ and $s(3)=z$. $\mathcal{G}(A_{exe3}) = (\overline{\mathbb{Z}}^{\{1,2,3\}}, \delta_{exe3}, (0,-\infty,-\infty), K_{exe3})$ with $K_{exe3} = \{(x,y,z) \mid z < 0 \text{ and } z \neq -\infty\}$. A part of δ_{exe3} is depicted in Fig. 3 (at this stage, we are not concerned with dashed arrows).

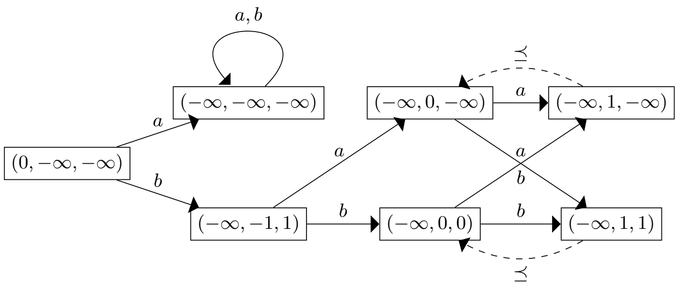

Fig. 3. A part of $\mathcal{G}(\mathcal{A}_{exe3})$

The automaton $\mathcal{A}$ is said to be *non-positive* if in $\mathcal{G}(\mathcal{A})$ there exists a path from s_0 to an element of K.

Proposition 1. *Let $\mathcal{A} = (Q, \Sigma, E, I, F)$ be a* max/+*-automaton. There exists $u \in L(\mathcal{A})$ such that $\mathcal{A}(u) < 0$ if and only if $\mathcal{A}$ is non-positive.*

4 State Space Exploration

We are interested in semi-deciding whether a max/+-automaton $\mathcal{A}$ is non-positive. Clearly, this is a matter of walking – by classical algorithms like depth-first search, random-walk, etc. – the determinisation graph $\mathcal{G}(\mathcal{A})$ defined above, until either a configuration in K is reached or there is an argument to prove such a configuration can no longer be found. Unfortunately, the determinisation reachability graph is generally infinite, and it is not easy to determine when it is safe to stop. Consequently, these algorithms may not terminate and can only conclude that $\mathcal{A}$ is non-positive but, when $\mathcal{G}(\mathcal{A})$ has infinitely many reachable configurations, they cannot conclude that $\mathcal{A}$ is not non-positive.

While reachability seems to a be a good tool to find configurations in K, for practical problems the determinisation graph usually has far too many configurations to calculate. To alleviate this problem, we exploit a *pruning configuration* approach. For that there is a need to introduce the relation $\preceq$ over configurations of a determinisation graph $\mathcal{G}(\mathcal{A})$ of an max/+-automaton $\mathcal{A}$. We define this relation by: $s_1 \preceq s_2$ iff for every state p in $\mathcal{A}$, $s_1(p) = -\infty$ iff $s_2(p) = -\infty$ and $s_1(p) \leq s_2(p)$ otherwise. The pruning is based on the following property.

Proposition 2. *Let $\mathcal{A} = (Q, \Sigma, E, I, F)$ be a* max/+*-automaton and $\mathcal{G}(A) = (V, \delta, s_0, K)$ its determinisation graph. Let $s_1, s_2 \in \overline{\mathbb{Z}}^Q$ such that $s_1 \preceq s_2$. Then if a configuration s_2' in K is reachable in $\mathcal{G}(\mathcal{A})$ from s_2, then there also is a configuration s_1' in K reachable from s_1.*

Proposition 2 can be proved by a direct induction using the following lemma.

Lemma 1. *Let $\mathcal{A} = (Q, \Sigma, E, I, F)$ be a* max/+*-automaton and $\mathcal{G}(A) = (V, \delta, s_0, K)$ its determinisation graph. Let $s_1, s_2 \in \overline{\mathbb{Z}}^Q$ such that $s_1 \preceq s_2$. Then for every letter $a \in \Sigma$, $\delta(s_1, a) \preceq \delta(s_2, a)$.*

While bounding the depth, Proposition 2 leads to the search based algorithm depicted in Fig. 4. In this algorithm, δ and K are related to the determinisation graph of $\mathcal{A}$. Notice too that a **Return** instruction ends the execution of the algorithm. Integer k is the bound of the number of computed configurations of the determinisation graph of $\mathcal{A}$. Set C is the set of computed accessible configurations. Set L encodes configurations to explore. Line 08, the function **Get** takes an element of L: the way this function is implemented may lead to different search approaches (depth first search, breadth first search, etc.). Next the graph is classically computed but only for configurations s such that there is no $s' \in C$ such that $s' \preceq s$ (notice that $\preceq$ is reflexive). The procedure ends at Line 06 if there is no more configuration to visit: K is not reachable. The algorithm then returns 1, indicating that for all $u \in \Sigma^+$, $\mathcal{A}(u) \geq 0$. The procedure ends at Line 10 if a configuration of K is reachable. Then the algorithm returns -1 indicating there exists u such that $\mathcal{A}(u) < 0$. At Line 18, the algorithm returns 0, indicating that it cannot conclude whether $\mathcal{A}$ is non-positive or not.

```
Algorithm Name: Explore
Input: $\mathcal{A}$, $k \in \mathbb{N}$
Local Variables: $L, C$ finite sets,
Begin
01.    Compute $C := \emptyset$
02.    Compute $s_0$
03.    Compute $L := \{s_0\}$
04.      While $(k \geq 0)$
05.        If $C \cap K = \emptyset$ and $L = \emptyset$
06.          Return 1
07.        EndIf
08.        Get $s \in L$
09 .       If $C \cap K \neq \emptyset$
10.          Return $-1$
11.        EndIf
12.        If not exists $s' \in C$ such that $s' \preceq s$
13.          $C := C \cup \{s\}$
14.          $L := L \cup \{\delta(s, a) \mid a \in \Sigma\}$
15.        EndIf
16.        $k := k - 1$
17.      EndWhile
18.    Return 0
End
```

Fig. 4. Exploration algorithm

For instance, let consider the max/+-automaton depicted in Fig. 2. The exploration algorithm computes the graph depicted in Fig. 3 where dashed arrows represent the $\preceq$ relation. On this example, the algorithm stops after a few steps and returns 1.

5 Rewriting Techniques Approach

Rewriting techniques are also well-suited for performing reachability analysis. In particular, reachability analysis allows verifying safety properties on critical systems: Java programs [?,?], cryptographic protocols [?] or Java Bytecode programs [?].

For the use of such techniques, rewriting semantics are defined for a given reachability problem, and the reachability analysis is performed from a rewriting point of view. Section 5.1 describes the rewriting model we use for determinisation graphs, and Section 5.2 explains how to show that an max/+-automaton is positive.

5.1 Rewriting Model for Determinisation Graphs

Focusing on the abstraction chosen in this paper, we specify the determinisation graph $\mathcal{G}(\mathcal{A})$ of a given automaton $\mathcal{A}$ as follows: its states are represented by terms and its transition relation is then compiled into rewrite rules. Integers are manipulated in their peano representations, i.e., using the constructors s (for successor), p (for predecessor) and 0. For example, 1 is represented by the term $s(0)$ and -2 by $p(p(0))$.

Thus, a configuration of a determinisation graph $\mathcal{G}(\mathcal{A})$ is specified by a term of the form $run(w_1, \ldots, w_n)$ where n is the number of states of $\mathcal{A}$, w_i is either a peano integer or $-\infty$. Considering this representation, the initial configuration $(0, -\infty, -\infty)$ of the determinisation graph in Fig. 3 is specified by the term $run(0, -\infty, -\infty)$.

The transition relation of a determinisation graph $\mathcal{G}(\mathcal{A})$ is then specified by a term rewriting system (TRS), i.e., a set of rewrite rules. The algorithm for generating such a TRS is simple. For a given max/+−automaton $\mathcal{A} = (\mathcal{Q}, \Sigma, E, I, F)$, we generate a set of rules per symbol of Σ by anticipating every possible scenario.

For instance, concerning $\mathcal{A}_{exe3}$ of Fig. 2 and the letter b, b can be read from the states 1, 2 and 3. So, a configuration of the determinisation graph when b is reading is a term of the form $run(t_1, t_2, t_3)$ where x_i's are variables, $t_1 \in \{-\infty, s(x_1), p(x_1)\}$, $t_2 \in \{-\infty, s(x_2), p(x_2)\}$ and $t_3 \in \{-\infty, s(x_3), p(x_3)\}$. For each of these terms, according to the transition relation of $\mathcal{G}(\mathcal{A}_{exe3})$, a successor term can be defined.

Example 3. For example, let $run(s(x_1), -\infty, p(x_3))$ be one of the forms mentioned right above. According to the $\mathcal{G}(\mathcal{A}_{exe3})$ transition relation, the following successor term can be set: $run(-\infty, +(s(x_1), p(0)), max(+(s(x_1), s(0)), +(p(x_3), p(0))))$. Consequently, one can define the rewrite rule

$$run(s(x_1), -\infty, p(x_3)) \rightarrow run(-\infty, +(s(x_1), p(0)), max(+(s(x_1), s(0)), +(p(x_3), p(0)))).$$

Doing so for each letter of Σ and for each form of terms, the whole transition relation can be defined as a TRS $\mathcal{R}$. In addition to these rules, those concerning the function max and the addition $+$ between two peano integers complete the set of rewrite rules. These classical additional rules are given in Appendix, Section 9.3.

5.2 Reachability Analysis

The rewriting model is now defined. Since we face systems whose number of states is potentially infinite, a complete and exact rewriting analysis is in general impossible. A well-suited approach as proposed in [16] is to compute an over-approximation of the reachable terms by rewriting – with a given set of rewrite rules $\mathcal{R}$ – from an initial set of terms E.

Initially, terms and subterms of terms in E are split into equivalence classes. For example, one can use tree automata to define equivalence classes where classes are actually the states of these automata. We refer the interested reader to [10,17] for more detail on tree automata and theoretical results on this topic. The technique in [16] enhances and creates new equivalence classes of terms and subterms by rewriting. If a term t is in an equivalence class C and t' is reachable by rewriting from t, then t' is added into the equivalence class C. Moreover, new equivalence classes may be added if there are subterms of t' which are not in existing equivalence classes. One proceeds in this way for all equivalence classes defined.

Approximations are done by manipulating equivalence classes of terms. In [15], Genet uses equations for merging equivalence classes. Let $c = c'$ be an equation where c and c' are two patterns, i.e., two terms that may contain variables. Let also C and C' be two equivalence classes of terms built with the technique described in [16]. If there exists a solution of c in C (resp. C') and a solution of c' in C' (resp. C), then the two equivalence classes are merged.

Example 4. For example, let consider the equation $s(x) = s(s(x))$ and the equivalence classes C_0, C_1, C_2, C_3 and C_4 such that $C_i = \{s^{(i)}(0)\}$. Since $s(0)$ is in C_1 and $s(s(0))$ is in C_2, using the equation we obtain that $s(0) = s(s(0))$. Consequently, C_1 and C_2 are merged into $C_{1,2}$. The same process can be applied for C_3 and C_4. Thus, the merging of C_3 and C_4 results in the equivalence class $C_{3,4}$. Once again, $s(s(0))$ and $s(s(s(0)))$ are respectively in $C_{1,2}$ and $C_{3,4}$. Using the given equation, the process results in a single equivalence class denoted C_{1to4}. Finally, using the given equation over the five equivalence classes gives rise to only two equivalence classes: C_0 and C_{1to4}.

As soon as the set of equivalence classes is stable by equation, rewriting is performed anew, and so on. The computation stops when all equivalence classes are closed by rewriting, i.e., when a fix-point set of terms is computed. Thus, the final set of terms is an over-approximation of the set of reachable terms.

For performing a reachability analysis, we can check on the fix-point set of terms if a pattern has a solution. If no solution exists then we can conclude that no term matching such a pattern is reachable from an initial set of terms E by rewriting with the given TRS $\mathcal{R}$.

Example 5. For example, in Fig. 2, the state 3 is the final state of $\mathcal{A}_{exe3}$. From the rewriting model, if we obtain a fix-point set of terms E', we have to check whether the patter $run(x, y, p(z))$ has a solution. In the negative case, we can conclude that no path in the determinisation graph has a negative cost. And, consequently, we also conclude that for every $u \in \mathcal{L}(\mathcal{A}_{exe3})$, $\mathcal{A}_{exe3}(u) \geq 0$. Whereas in the positive case, no conclusion can be raised. Indeed, the solution of the pattern may come from a side-effect of the approximation.

Section 6 reports on the implementation and experimental results for the proposed rewriting model. Notice that the rewriting-based encoding and analysis are used when the exploration algorithm in Fig. 4 returns 0. Obviously, other rewriting models and other rewriting approximations can be defined.

6 Experiments

In order to evaluate our approaches, we randomly generate non-deterministic finite max/+-automata using the following method: given a set of states $\{1, \ldots, n\}$, for each letter a and each i, j, there is a fixed probability $p_{\text{transition}}$ to have a transition of the form (i, a, c, j). If such a transition exists, its weight is uniformly picked up between $-c_{\max}$ and $c_{\max}$. Moreover, 1 is the unique initial state, n is always a final state, and there is a fixed probability p_{final} for each other state to be final. If a generated automaton accepts the empty language, it is rejected. We have done several tests with different values of $c_{\max}$, $p_{\text{transition}}$ and p_{final}. Table 1 reports on results obtained with $c_{\max} = 3$, $p_{\text{transition}} = 0.3$ and $p_{\text{final}} = 0.1$. For each value of n from 2 to 20, we randomly generate 1000 automata. Line n is the number of states of the automata. We first run the Explore algorithm developed in Sec. 4 with $k = 10n$. Line *pos.* (resp. *neg.*) reports on the proportion of inputs when the algorithm returns 1 (resp. -1). Line *??* indicates the number of automata (out of 1000 automata generated for each n) for which the algorithm returns 0. Line *depth* reports on the average number of computed reachable configurations in the Explore algorithm (when it returns 1 or -1).

n	2	3	4	5	6	7	8	9	10	12	14	16	18	20
pos.	0.34	0.23	0.15	0.1	0.1	0.12	0.13	0.16	0.21	0.27	0.33	0.40	0.42	0.6
neg.	0.65	0.74	0.82	0.86	0.87	0.85	0.83	0.81	0.77	0.71	0.66	0.59	0.57	0.54
depth	2.45	3.83	4.66	5.92	6.68	6.88	7.14	7.10	7.40	7.35	7.46	7.64	7.47	7.37
??	4	21	25	39	27	28	27	22	21	23	8	8	5	4
TRS	36	43	58	79	125	296	554	1068	2094	8242	32822	131130	524350	$\approx 2^{21}$
inc.	0	4	6	10	6	T	T	T	T	T	T	T	T	T

Table 1. Experimental results

When the first algorithm gives the inconclusive results, we apply the second rewriting approximation approach to them. Experiments have been led for $n = 2, 3, 4$ and 5 using equations allowing to split integers into 13 equivalence classes: < -5, $= -5$, $= -4$, $= -3$, $= -2$, $= -1$, $= 0$, $= 1$, $= 2$, $= 3$, $= 4$, $= 5$ and > 5. For example, the equivalence class < -5 is defined by the equation $p(p(p(p(p(p(x)))))) = p(p(p(p(p(p(p(x)))))))$.

Table 1 reports at line *inc.* on the number of automata that are not shown to be positive using the rewriting approximation technique among inconclusive analyses from the pruning approach. The result T points out that the implementation of the rewriting approach fails to answer because of a stack overflow. This table also gives details (line *TRS*) about the average number of rewrite rules generated for the rewriting specifications.

7 Discussions and Perspectives

Let consider the max/+-automaton $\mathcal{A}_{exe4}$ depicted in Fig. 5. Notice that $\mathcal{A}_{exe4}$ is not non-positive.

For this automaton, the exploration will never end since $\mathcal{G}(\mathcal{A}_{exe4})$ has infinitely many configurations of the form $(-n, 2n)$, which are pairwise incomparable by $\preceq$. For more difficult reasons, similar to those given in [7], the approximation technique can not conclude either.

We discuss and propose several ways to handle remaining intractable cases.

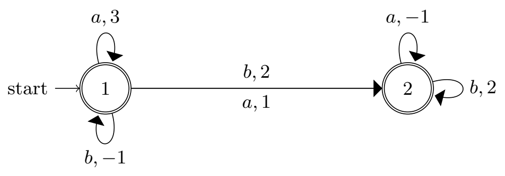

Fig. 5. max/+-automaton $\mathcal{A}_{exe3}$

7.1 Counter Systems Encoding

Presburger logic is the first order logic over $(\mathbb{Z}, +, =)$. A *n-counter-system $\mathcal{C}$* is a tuple (Q, T, P) where Q is a finite set of *states*, P is a finite set of Presburger formulas with $2n$ free variables, and T is a finite set of elements of the form (p, φ, q) where $\varphi \in P$. For every $\varphi(x_1, \ldots, x_n, y_1, \ldots, y_n) \in P$ we define the relation $\rightarrow_\varphi$ on $\mathbb{Z}^n \times \mathbb{Z}^n$ by: $(a_1, \ldots, a_n) \rightarrow_\varphi (b_1, \ldots, b_n)$ iff $\varphi(a_1, \ldots, a_n, b_1, \ldots, b_n)$ is true. Finally, given the set $S_0 \subseteq \mathbb{Z}^n$, the set $\mathrm{Post}^*_{\mathcal{C}}(S_0)$ (resp. $\mathrm{Pre}^*_{\mathcal{C}}(S_0)$) is the set of $s \in \mathbb{Z}^n$ such that there exist $w = w_1 \ldots w_k \in P^*$ ($w_i \in P$), $s_0 \in S_0$ and $s_1, \ldots, s_k \in \mathbb{Z}^n$, where $s_k = s$ and for every i, $s_i \rightarrow_{w_{i+1}} s_{i+1}$ (resp. $s_{i+1} \rightarrow_{w_{i+1}} s_i$).

It is known [18] that subsets of $\mathbb{Z}^n$ that are definable by a Presburger formula with n free variables are exactly regular subsets of $(\mathbb{Z}^n, +)$. This nice property, associated with nice connections to Petri nets, has lean to many works to compute sets of the form $\mathrm{Post}^*_{\mathcal{C}}(S_0)$ or $\mathrm{Pre}^*_{\mathcal{C}}(S_0)$ (see [26] for a recent work with references), supported by tools as FAST [4], LASH [8] or TReX [2].

We now illustrate how to encode our problem into this model. Let $\mathcal{A} = (Q, \Sigma, E, I, F)$ be a max/+-automaton. Without loss of generality we may assume that $Q = \{1, \ldots, n\}$. We consider the function ψ from $\overline{\mathbb{Z}}^Q$ into $\mathbb{Z}^{2n}$ defined as follows: for every $s \in \overline{\mathbb{Z}}^Q$, $\psi(s)$ is the vector $(s_1, \ldots, s_{2n})$ where for every $1 \leq i \leq n$, $s_i = s(i)$ and $s_{n+i} = 0$ if $s_i \in \mathbb{Z}$, and $s_i = 0$ and $s_{n+i} = 1$ otherwise. For instance if $Q = \{1, 2, 3\}$ and $s(1) = -1$, $s(2) = 3$ and $s(3) = -\infty$, then $\psi(s) = (-1, 3, 0, 0, 0, 1)$. Notice first that the max-function is Presburger definable: $z = \max(x, y)$ iff x, y, z satisfy the formula

$$\varphi_{\max}(z, x, y) := ((z = x \vee z = y) \wedge ((x \leq y) \Rightarrow z = y))$$

Writing exact formulas encoding a generic $\mathcal{A}$ is quite long. Since we do not use this approach and since our goal is just to show how to use it, we provide the encoding for the automaton $\mathcal{A}_{exe3}$. One has $\delta_{exe3}(s, b) = s'$ iff $\varphi_b(\psi(s), \psi(s'))$ is satisfied, where φ_b is depicted in Fig. 6.

In this context, the non-positivity problem is reduced either to $\mathrm{Pre}^*_{\mathcal{C}}(\{\psi(s_0))\} \cap \psi(K) = \emptyset$? or, equivalently, to $\psi(s_0) \in \mathrm{Post}^*_{\mathcal{C}}(\{\psi(K))\}$? where $\mathcal{C}$ is the counter system encoding $\mathcal{A}$. One can also easily verify that $\psi(K)$ is Presburger definable.

7.2 Using max/+ Theory

Another way to improve the approach consists in using theoretical results on max/+-automata. For instance, a recent work [23] points out new subclasses of max/+-automata for which the positivity problem is decidable. However, the proposed constructive proof is far from being effective, and an algorithmic research has still to be done.

$$
\begin{aligned}
&\varphi_b(x_1,x_2,x_3,x_4,x_5,x_6,y_1,y_2,y_3,y_4,y_5,y_6) :=\\
&\quad y_1 = 0 \wedge y_4 = 1\\
&\quad \wedge ((x_4 = 0 \wedge x_5 = 0 \wedge x_6 = 0) \Rightarrow (y_2 = 0 \wedge y_3 = 0 \wedge y_5 = 1 \wedge y_6 = 1))\\
&\quad \wedge ((x_4 = 0 \wedge x_5 = 0 \wedge x_6 = 1) \Rightarrow (y_2 = 0 \wedge y_3 = x_3 - 1 \wedge y_5 = 0 \wedge y_6 = 1))\\
&\quad \wedge ((x_4 = 0 \wedge x_5 = 1 \wedge x_6 = 0) \Rightarrow (y_2 = x_2 + 1 \wedge y_3 = x_2 + 1 \wedge y_5 = 0 \wedge y_6 = 0))\\
&\quad \wedge ((x_4 = 0 \wedge x_5 = 1 \wedge x_6 = 1) \Rightarrow (y_2 = x_2 + 1 \wedge \varphi_{\max}(y_3, x_2 + 1, x_3 - 1) \wedge y_5 = 0 \wedge y_6 = 0))\\
&\quad \wedge ((x_4 = 1 \wedge x_5 = 0 \wedge x_6 = 0) \Rightarrow (y_2 = x_1 - 1 \wedge y_3 = x_1 + 1 \wedge y_5 = 0 \wedge y_6 = 0))\\
&\quad \wedge ((x_4 = 1 \wedge x_5 = 0 \wedge x_6 = 1) \Rightarrow (y_2 = x_1 - 1 \wedge \varphi_{\max}(y_3, x_1 + 1, x_3 - 1) \wedge y_5 = 0 \wedge y_6 = 0))\\
&\quad \wedge((x_4 = 1 \wedge x_5 = 1 \wedge x_6 = 0) \Rightarrow (\varphi_{\max}(y_2, x_1 - 1, x_2 + 1)\\
&\qquad\qquad \wedge \varphi_{\max}(y_3, x_1 + 1, x_2 + 1) \wedge y_5 = 0 \wedge y_6 = 0))\\
&\quad \wedge((x_4 = 1 \wedge x_5 = 1 \wedge x_6 = 1) \Rightarrow \varphi_{\max}(y_2, x_1 - 1, x_2 + 1)\\
&\qquad\qquad \wedge \exists z\ (\varphi_{\max}(y_3, x_1 + 1, z) \wedge (\varphi_{\max}(z, x_2 + 1, x_3 - 1)) \wedge y_5 = 0 \wedge y_6 = 0))
\end{aligned}
$$

Fig. 6. Presburger formula

Another very interesting direction may be to use results of [27]: for a one-letter alphabet, many problems becomes decidable. In particular, such results can be used during the exploration of a determinisation graph. When visiting a configuration s, for each letter a, one can test with one step whether there exists $n \geq 0$ such that $\delta(s, a^n) \cap K \neq \emptyset$. It may deeply reduce the exploration for non-positive max/+-automata. Moreover, we think this approach can be used to perform a symbolic exploration of the determinisation graph: rather than visiting each accessible configuration, we would work on infinite sets of configurations similarly to the counter system encoding presented below.

8 Conclusion

We proposed to exploit abstractions and approximations to semi-decide the positivity problem over max/+-automata whose determinisation reachability graphs are infinite state systems. The positivity problem is then reduced to a reachability problem on these graphs. We developed two semi-decision procedures and explained how to conclude more often and how to do it efficiently.

The first kind of determinisation-based reachability graphs abstractions together with pruning technique gives rise to a semi-decision procedure. The experimental results on thousands of automatically generated max/+-automata show that when bounding the depth of search in the determinisation graphs, the algorithm seems to be efficient enough.

The second kind of abstractions is based on the reachability analysis through rewriting approximations as well as tree automata. The rewriting-based reachability encoding has been applied to the inconclusive cases previously obtained with the exploration algorithm.

Rewriting approximation techniques were already implemented in [3]. In the future we plan to integrate integer weighted automata based algorithms into this tool in order to treat practical applications. Obviously, other rewriting models and other rewriting approximations can be defined. Moreover, one can propose an abstraction refinement for rewriting approximations guided by the property to be verified, as in [6].

Finally we plan to experiment our techniques with other random generators of non-deterministic finite automata, for instance using the one developped in [30].

References

1. P. A. Abdulla, K. Cerans, B. Jonsson, and T. Yih-Kuen. General decidability theorems for infinite-state systems. In *Proc. 11th IEEE Symp. Logic in Computer Science*, 1996.
2. A. Annichini, A. Bouajjani, and M. Sighireanu. Trex: A tool for reachability analysis of complex systems. In G. Berry, H. Comon, and A. Finkel, editors, *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 368–372. Springer, 2001.
3. E. Balland, Y. Boichut, T. Genet, and P.-E. Moreau. Towards an efficient implementation of tree automata completion. In *AMAST*, pages 67–82, 2008.
4. S. Bardin, A. Finkel, J. Leroux, and Laure Petrucci. Fast: acceleration from theory to practice. *STTT*, 10(5):401–424, 2008.
5. N. Bertrand and Ph. Schnoebelen. A short visit to the sts hierarchy. *Electr. Notes Theor. Comput. Sci.*, 154(3):59–69, 2006.
6. Y. Boichut, R. Courbis, P.-C. Héam, and O. Kouchnarenko. Finer is better: Abstraction refinement for rewriting approximations. In *Rewriting Techniques and Application, RTA'08*, volume 5117 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2008.
7. Y. Boichut and P.-C. Héam. A theoretical limit for safety verification techniques with regular fix-point computations. *Inf. Process. Lett.*, 108(1):1–2, 2008.
8. B. Boigelot and P. Wolper. Representing arithmetic constraints with finite automata: An overview. In P. J. Stuckey, editor, *ICLP*, volume 2401 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2002.
9. A.L. Buchsbaum, R. Giancarlo, and J. Westbrook. An approximate determinization algorithm for weighted finite-state automata. *Algorithmica*, 30(4):503–526, 2001.
10. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. `http://www.grappa.univ-lille3.fr/tata/`, 2002.
11. M. Droste and P. Gastin. Weighted automata and weighted logics. *Theor. Comput. Sci.*, 380(1-2):69–86, 2007.
12. A. Finkel. Reduction and covering of infinite reachability trees. *Information and Computation*, 89(2):144–179, 1990.
13. A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.
14. S. Gaubert. Performance Evaluation of (max,+) Automata. *IEEE Trans. on Automatic Control*, 40(12), 1995.
15. Th. Genet. Timbuk 3.0 : Equationnal approximations. Available at http://http://www.irisa.fr/lande/genet/timbuk/.
16. Th. Genet. Decidable approximations of sets of descendants and sets of normal forms. In *proceedings of RTA*, volume 1379 of *LNCS*. Springer-Verlag, 1998.
17. R. Gilleron and S. Tison. Regular tree languages and rewrite systems. *Fundamenta Informaticae*, 24:157–175, 1995.
18. S. Ginsburg and E.H. Spanier. Bounded regular sets. *Proceedings of the American Mathematical Society*, 7:1043–1049, 1966.

19. K. Hashiguchi, K. Ishiguro, and S. Jimbo. Decidability of the Equivalence Problem for Finitely Ambiguous Finance Automata. *IJAC*, 12(3), 2002.
20. Th. A. Henzinger, R. Majumdar, and J.F. Raskin. A classification of symbolic transition systems. *ACM Trans. Comput. Log.*, 6(1):1–32, 2005.
21. K. Culik II and P.C. von Rosenberg. Generalized weighted finite automata based image compression. *J. UCS*, 5(4):227–242, 1999.
22. F. Katritzke, W. Merzenich, and M. Thomas. Enhancements of partitioning techniques for image compression using weighted finite automata. *Theoretical Computer Science*, 313(1):133–144, 2004.
23. D. Kirsten and S. Lombardy. Deciding unambiguity and sequentiality of polynomially ambiguous min-plus automata. In *STACS*, pages 589–600, 2009.
24. I. Klimann, S. Lombardy, J. Mairesse, and Ch. Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theoretical Computer Science*, 327(3):349–373, 2004.
25. D. Krob. The Equality Problem for Rational Series with Multiplicities in the Tropical Semiring is Undecidable. *IJAC*, 4(3), 1994.
26. J. Leroux. Structural presburger digit vector automata. *Theoretical Computer Science*, 409(3):549–556, 2008.
27. S. Lombardy. Sequentialization and unambiguity of (max,+) rational series over one letter. In S. Gaubert and J.-J. Loiseau, editors, *Workshop on max-plus Algebras and Their Applications to Discrete-event Systems, Theoretical Computer Science, and Optimization*, Prague, 2001. IFAC, Elsevier Sciences.
28. M. Mohri, F. Pereira, and M. Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.
29. M. Mohri, F. Pereira, and M. Riley. Weighted automata in text and speech processing. volume abs/cs/0503077, 2005.
30. D. Tabakov and M.Y. Vardi. Experimental evaluation of classical automata constructions. In G. Sutcliffe and A. Voronkov, editors, *LPAR*, volume 3835 of *Lecture Notes in Computer Science*, pages 396–411. Springer, 2005.
31. A. Weber. Finite-valued Distance Automata. *Theoretical Computer Science*, 134, 1994.

9 Appendix

9.1 Proof of Proposition 1

Proposition 1 is a direct consequence of the following lemma. The reader familiar with max/+-automata may notice that this lemma is a direct consequence of matricial presentation of max/+-automata.

Lemma 2. *Let $u \in \Sigma^+$, $\mathcal{A} = (Q, \Sigma, E, I, F)$ be a max/+-automaton and $\mathcal{G}(\mathcal{A}) = (V, \delta, s_0, K)$ its determinisation graph. There is a path in $\mathcal{G}(\mathcal{A})$ from s_0 to s labelled by u if and only if for every $p \in Q$,*

$$s(p) = \max\{\mathrm{cost}_{\mathcal{A}}(\pi) \mid \pi \textit{ is a path in } \mathcal{A} \textit{ from an initial state to } p\}.$$

Proof. We will prove the lemma by induction on the length of u.

Assume that $u \in \Sigma$ and that $\delta(s_0, u) = s$. By definition of δ, for every state p, $s(p) = \max\{s_0(q) + c \mid (q, a, c, p) \in E\}$. Therefore and by definition of s_0, $s(p)$ is exactly the maximal value of all transition weights from an initial state to p, proving the lemma for u's in Σ.

Assume now that the lemma is true for all words of length $k \geq 1$. Let $u \in \Sigma^{k+1}$. There exists $v \in \Sigma^k$ and $a \in \Sigma$ such that $u = va$. Let $s_1 = \delta(s_0, v)$. Each path π in $\mathcal{A}$ from an initial state to p can be decomposed into $\pi = \pi_1, (q, a, c, p)$ where π_1 is labelled by v and $(q, a, c, p) \in E$. Since $\mathrm{cost}_{\mathcal{A}}(\pi) = \mathrm{cost}_{\mathcal{A}}(\pi_1) + c$, one has

$$\begin{aligned}
s(p) &= \max\{s_1(q) + c \mid (q, a, c, p) \in E\} \\
&= \max\{\max\{\mathrm{cost}(\pi_1) \mid \pi_1 \text{ from an initial state to } q\} + c \mid (q, a, c, p) \in E\} \\
&= \max\{\max\{\mathrm{cost}(\pi_1) + c \mid \pi_1 \text{ from an initial state to } q\} \mid (q, a, c, p) \in E\} \\
&= \max\{\mathrm{cost}(\pi_1) + c \mid \pi_1 \text{ from an initial state to } q \text{ and } (q, a, c, p) \in E\} \\
&= \max\{\mathrm{cost}_{\mathcal{A}}(\pi) \mid \pi \text{ is a path in } \mathcal{A} \text{ from an initial state to } p\}.
\end{aligned}$$

Consequently, the lemma is true for words of Σ^{k+1}, concluding the proof.

9.2 Proof of Lemma 1

Proof. Notice first that δ is a function defined on $V \times \Sigma$, thus $\delta(s_1, a)$ and $\delta(s_2, a)$ both exist.

Now $\delta(s_1, a)(p) = -\infty$ iff $\{q \mid s_1(q) \neq -\infty\} \cap \{q \mid \exists (q, a, c, p) \in E\} = \emptyset$. Since $s_1 \preceq s_2$, $\{q \mid s_1(q) \neq -\infty\} = \{q \mid s_2(q) \neq -\infty\}$. Therefore, $\delta(s_1, a)(p) = -\infty$ iff $\delta(s_2, a)(p) = -\infty$. Moreover, for every state p,

$$\begin{aligned}
\delta(s_1, a)(p) &= \max\{s_1(q) + c \mid (q, a, c, p) \in E\} \\
&\leq \max\{s_2(q) + c \mid (q, a, c, p) \in E\} \\
&= \delta(s_2, a)(p),
\end{aligned}$$

proving the lemma.

9.3 Rewriting Rules for max and +

$$\begin{array}{ll}
max(s(x), s(y)) \to s(max(x, y)) & +(s(x), p(y)) \to +(x, y) \\
max(p(x), s(y)) \to s(y) & +(p(x), s(y)) \to +(x, y) \\
max(s(y), p(x)) \to s(y) & +(s(x), s(y)) \to s(s(+(x, y)) \\
max(p(x), p(y)) \to p(max(x, y)) & +(p(x), p(y)) \to p(p(+(x, y)) \\
max(0, 0) \to 0 & +(0, x) \to x \\
max(s(x), 0) \to s(x) & +(x, 0) \to x \\
max(0, s(x)) \to s(x) & \\
max(p(x), 0) \to 0 & \\
max(0, p(y)) \to 0 &
\end{array}$$