Max-plus Automata for the Worst-Case Complexity Analysis

Work in progress

Laure Daviaud

DICE 2016

Abstract

This work intends to link the computation of the complexity of a program with results in automata theory about the description of max-plus automata. Max-plus automata are special cases of weighted automata, which use operations + and max on integers, with a semantics that is particularly suitable for the worst-case complexity analysis.

The first piece of work using this approach was done in [CDZ14]. In that paper, the worst case complexity is studied in a program abstraction: the size-change abstraction. This abstraction is essentially a means of taking into account the decrease of the variables of a program.

The objective now is to give another abstraction extending the size-change abstraction and/or directly abstract a program in a max-plus automaton in order to: (1) study the compositionality of this method, (2) take into account both the decrease and the increase of the variables for the complexity analysis.

1 Context and related works

Max-plus automata are quantitative extensions of non deterministic finite automata and belong to the wider family of weighted automata, as introduced by Schützenberger [Sch61]. They compute functions that map words to integers according to a semantics using operations max and +. They are in particular used in optimisation and verification and for the description of discrete event systems [Gau95, GM99, LM].

More formally, a *max-plus automaton* over the alphabet A is a tuple (Q, T, I, F) where Q is a finite set of states, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states and $T \subseteq Q \times A \times \mathbb{Z} \times Q$ is a finite set of transitions. The weight of a run is the sum of the weights of the transitions in the run and the *function computed by the automaton* maps each word $w \in A^*$ to the maximum of the weights of the accepting runs (runs going from an initial to a final state) labelled by w or to $-\infty$ if there is no accepting run labelled by w.

The **size-change abstraction** (SCA) is a program abstraction for automated termination analysis (e.g. [MV06]). It is employed for the termination analysis of functional [LJBA01, MV06, JK09], logical [Vid07] and imperative [AK03, CGB$^+$11] programs, term rewriting systems [CZ10], and is implemented in the industrial-strength systems ACL2 [MV06] and Isabelle [Kra07]. SCA consists in abstracting a program in a finite state system whose transitions are labelled by a representation of the decrease of the variables of the program.

More formally, we fix a set of variables χ. An *instance of the size-change abstraction* is given by a set of states Q and a set of transitions $T \subseteq Q^2 \times 2^{\chi^2 \times \{>,\geqslant\}}$. The semantics of such a system is defined by means of the *valuations* $v : \chi \to \mathbb{N}$. A pair of valuations (v_1, v_2) is a *model* for $t = (p, q, \sigma) \in T$, if for all $(x, y, \blacktriangleright) \in \sigma$, $v_1(x) \blacktriangleright v_2(y)$. A *trace* is a sequence $v_1 \xrightarrow{t_1} v_2 \xrightarrow{t_2} \cdots$ such that for all i, $t_i = (p_i, q_i, \sigma_i)$ for some $p_i, q_i \in Q$ with $q_i = p_{i+1}$ and $\sigma_i \in 2^{\chi^2 \times \{>,\geqslant\}}$ such that (v_i, v_{i+1}) is a model of t_i. The *length of a trace* is the number of transitions that the trace uses. An SCA instance is said to be *terminating*, if it does not have any trace of infinite length. It is decidable in PSPACE whether an SCA instance is terminating [LJBA01].

In what follows, we will restrict valuations to $\chi \to [0, n]$ to guarantee that the length of a trace is bounded for terminating SCA. Indeed, for terminating SCA, no pairs of a state q_i and a valuation $v_i \in \chi \to [0..n]$ can appear twice in a trace (otherwise we would have a cycle, which could be pumped to an infinite trace); thus the length of traces is bounded by $|Q|(n+1)^k$ for SCA with k variables.

With respect to complexity, we are interested in studying the length of the longest trace in an SCA instance. In [CDZ14], max-plus automata are used to give an equivalent of the asymptotic worst-case complexity in this abstraction.

More precisely, given a max-plus automaton $\mathcal{A}$, one can consider the function g that maps each integer n to the length of the longest word having values less than n by $\mathcal{A}$. The following theorem gives an asymptotic equivalent of the function g.

Theorem 1 ([CDZ14])**.** *Given a max-plus automaton computing a function $f : A^* \to \mathbb{N} \cup \{-\infty\}$, only using non negative weights, there is $\alpha \in \{+\infty\} \cup (\mathbb{Q} \cap [1, +\infty))$ such that*

$$g(n) = \Theta(n^\alpha)$$

where $g(n) = \sup\{|w| \ : \ f(w) \leqslant n\}$, with the convention that $n^{+\infty} = +\infty$. Moreover, there is an algorithm with input a max-plus automaton that computes such an α.

By an adequate transformation from SCA to max-plus automata, one can apply this theorem to get the following result which gives an equivalent of the length of the longest trace in an SCA instance.

Theorem 2 ([CDZ14])**.** *Let $\mathcal{S}$ be a terminating SCA instance. The length of the longest trace of $\mathcal{S}$ is of order $\Theta(n^\alpha)$ if the variables are restricted to take values in $[0, n]$, where $\alpha \geqslant 1$ is a rational number. Moreover, there is an algorithm that given a terminating SCA instance computes such a α.*

Example 1. *Figure 1 represents a program, its abstraction size-change and its transformation into a max-plus automaton that allows one to apply Theorem 1 for the study of the worst-case complexity.*

The different instructions in the program are abstracted into transitions that are labelled by the decrease of the variables. The different possible executions of the program correspond to non deterministic choices in the SCA instance.

The max-plus automaton, given in the rightmost part, computes a function on $\{t_1, t_2\}^*$ which maps a word w to the integer $\max(|w|_{t_1}, |w|_{t_2})$ where $|w|_x$ is the number of occurrences of the letter x in w.

This max-plus automaton allows one to study the length of the longest trace of the SCA instance in the middle. The idea behind the construction is to represent the variables of the

program in the states of the max-plus automaton. For each transition t in the SCA instance containing $z \geqslant s'$ or $z > s'$, there is a transition in the max-plus automaton from z to s, labelled by t. The weight of the transition is 1 if the inequality is strict and 0 otherwise. By such a transformation, the maximal number of strict inequalities used in a given trace in the SCA instance corresponds to the weight of the corresponding word in the max-plus automaton. Thus, an asymptotic equivalent to the function g defined in Theorem 1 in the max-plus automaton gives an asymptotic equivalent to the length of the longest trace in the SCA instance given that variables are bounded by some integer.

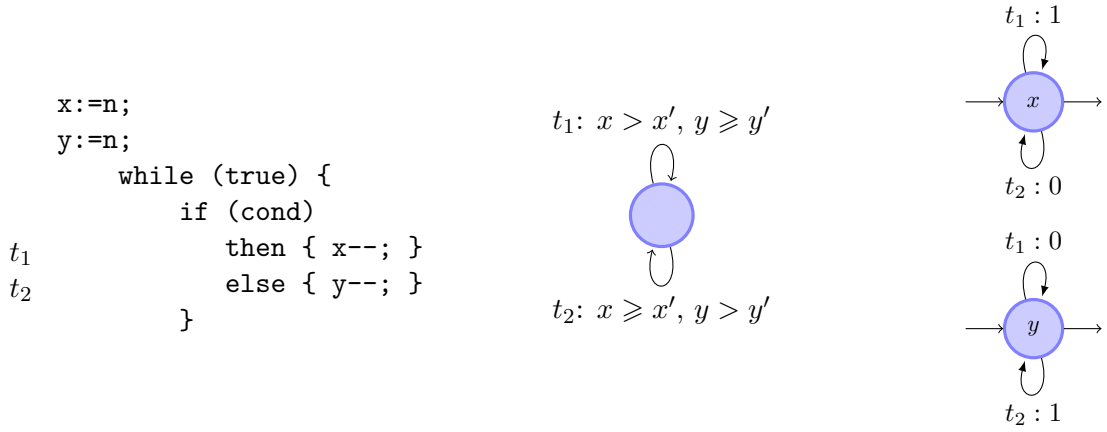

Figure 1: Program, its abstraction size-change and the corresponding max-plus automaton.

2 Compositionality and generalisation

Now, the aim of this work is to deal with the link between worst-case complexity and max-plus automata in depth.

The first objective is to study the compositionality of this kind of methods. Suppose thus that a program calls some function `f` annotated by some information. With respect to SCA, we consider two cases: (1) `f` is annotated by information on the decrease of the variables of the shape $x \geqslant y'$ or $x > y'$. Thus, `f` can be directly abstracted as a transition of the SCA instance. Nevertheless, in this case, the execution time of `f` is not taken into account. (2) `f` is annotated by its complete abstraction in an SCA instance. It is then sufficient to connect the SCA instance representing `f` to the SCA instance representing the rest of the program. This case is exact with respect to SCA model.

The main issue is then to take into account the execution time of `f` without annotating it with the complete information contained in its size-change abstraction. We therefore introduce a new abstraction, using transitions that are labelled by asymptotic information on the decrease of the variables plus an asymptotic equivalent of the execution time (in the flavour of the "α" given in Theorem 1). The information on the decrease of the variables can be more or less precise (of the shape $x \geqslant y'$, $x > y'$ or $x \geqslant y' + n^\alpha$) depending on the model we consider. By using an adaptation of Theorem 1, these different models allow one to give over-approximations of the worst-case complexity of a program, by compositionality.

Example 2. *Consider the program in Figure 2. The instruction t'_1 calls a function* `f`*. Suppose that this function is annotated by its execution time n^α (meaning that if the variables used in* `f` *are restricted to take values in $[0, n]$ the execution time of* `f` *is equivalent to n^α) and by the asymptotic amount by which the variable y is incremented with respect to the variable x ($y \geqslant x + n^\beta$). The instructions of the program are abstracted by taking into account these two pieces of information. In the automata representation, every transition is labelled by two elements: the first one represents the weight of the transition or equivalently the asymptotic decrease of the variable; the second one represents the asymptotic execution time needed to perform this transition. States x and y represent the variables x and y before the evaluation of the condition* `if` *(or equivalently after performing all the instructions in the condition). States x_1 and y_1 represent the variables x and y after performing the first instruction t_1* `x--;` *in the case where* `cond` *is evaluated to* `true`*.*

```
      x:=n;
      y:=n;
          while (true) {
              if (cond)
t1               then { x--;
t'1                     y:=f(x); }
t2            else { y--; }
              }
```

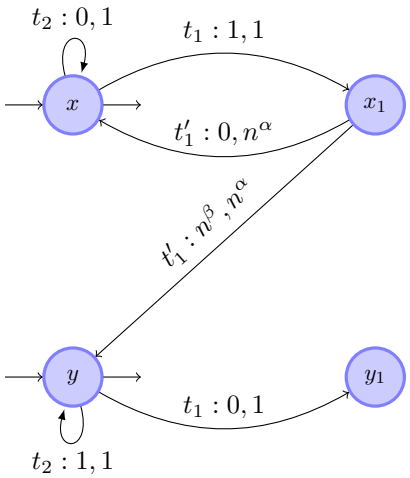

Figure 2: Compositionality.

Finally, we would like to consider both the increase and decrease of the variables. Then, the problem becomes that of giving an asymptotic equivalent of the length of the longest trace in the program when variables are restricted to be in $[-n, n]$. Nevertheless, the construction of the max-plus automaton in this case requires the use of both negative and positive weights and Theorem 1 is no longer true in this case.

Theorem 1 can be generalised and adapted in restricted cases of max-plus automata to remain true even with the use of negative weights. However, generally speaking, the question of describing the asymptotic behaviour of a max-plus automaton is undecidable. Nevertheless, restrictions on the program under study yield the computability of the worst-case complexity in a specific abstraction taking into account both decrease and increase of the variables.

References

[AK03] H. Anderson and S-C. Khoo. Affine-based size-change termination. In Atsushi Ohori, editor, *APLAS*, volume 2895 of *Lecture Notes in Computer Science*, pages 122–140. Springer, 2003.

[CDZ14] T. Colcombet, L. Daviaud, and F. Zuleger. Size-change abstraction and max-plus automata. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 2014.

[CGB+11] M. Codish, I. Gonopolskiy, A-M. Ben-Amram, C. Fuhs, and J. Giesl. Sat-based termination analysis using monotonicity constraints over the integers. *TPLP*, 11(4-5):503–520, 2011.

[CZ10] M. Codish and M. Zazon-Ivry. Pairwise cardinality networks. In *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*, pages 154–172, 2010.

[Gau95] S. Gaubert. Performance evaluation of $(\max, +)$ automata. *IEEE Trans. Automat. Control*, 40(12):2014–2025, 1995.

[GM99] S. Gaubert and J. Mairesse. Modeling and analysis of timed Petri nets using heaps of pieces. *IEEE Trans. Automat. Control*, 44(4):683–697, 1999.

[JK09] N. D. Jones and L. Kristiansen. A flow calculus of *mwp*-bounds for complexity analysis. *ACM Trans. Comput. Log.*, 10(4), 2009.

[Kra07] A. Krauss. Certified size-change termination. In Frank Pfenning, editor, *Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings*, volume 4603 of *Lecture Notes in Computer Science*, pages 460–475. Springer, 2007.

[LJBA01] C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. In Chris Hankin and Dave Schmidt, editors, *POPL*, pages 81–92. ACM, 2001.

[LM] S. Lombardy and J. Mairesse. Max-plus automaton. *Handbook of Automata*, European Mathematical Society, to be published.

[MV06] P. Manolios and D. Vroon. Termination analysis with calling context graphs. In Thomas Ball and Robert B. Jones, editors, *CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 401–414. Springer, 2006.

[Sch61] M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.

[Vid07] G. Vidal. Quasi-terminating logic programs for ensuring the termination of partial evaluation. In G. Ramalingam and Eelco Visser, editors, *Proceedings of the 2007 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation, 2007, Nice, France, January 15-16, 2007*, pages 51–60. ACM, 2007.