Termination and Non Size Increasingness of assembly programs

Jean-Yves Marion & Jean-Yves Moyen

1 Introduction

In this work, we study a general framework for modelising assembly-like programs and study their properties. This is originally inspired by Lee, Jones and Ben-Amram's Size Change Principle [LJBA01] but turn out to be also able to detect memory usage properties similar to Hofmann's Non Size Increasingness [Hof99] and perform transformations similar to Wadler's deforestation [Wad90]. Similar technics exist, using Petri nets [Moy03, MM03] or matrices [NW].

2 Petri nets

Programs are written in a small assembly-like language. We have counters storing positivce unary numbers. Those must be thinked about as the size of actual data (*e.g.* the length of lists). We can increment or decrement counters values, and perform conditionnal (if a counter's value is 0) and unconditionnal jumps to fixed labels of the program.

The following program computes in y the addition of x and y:

$$\begin{array}{lcl} 0 & : & \texttt{if } x = 0 \texttt{ jmp } 4; \\ 1 & : & x--\,;\ 2\ :\ y++; \\ 3 & : & \texttt{jmp } 0\,;\ 4\ :\ \texttt{end}; \end{array}$$

The program is then modelised by a Petri net. The control part (in red) consists of one place for each label and one transition for each instruction (two for conditionnal jumps). The memory part (in blue) consist of one memory place, with links to increment and decrement transitions and the size-change part (in green) consists of one place for each variable with links to the corresponding increment/decrement transitions. The Petri net for addition is shown on the left part of Figure 1.

Now, it is easy to show that for each execution of the program we can put sufficiently enough tokens in the varaibles and memory places, plus on in 0, and have a firing sequence of the petri net simulating the execution of the program. This leads to a sufficient condition for uniform termination of the program.

Theorem 1. *If, whatever the initial marking, the Petri net has no infinite firing sequence, the the program always terminates.*

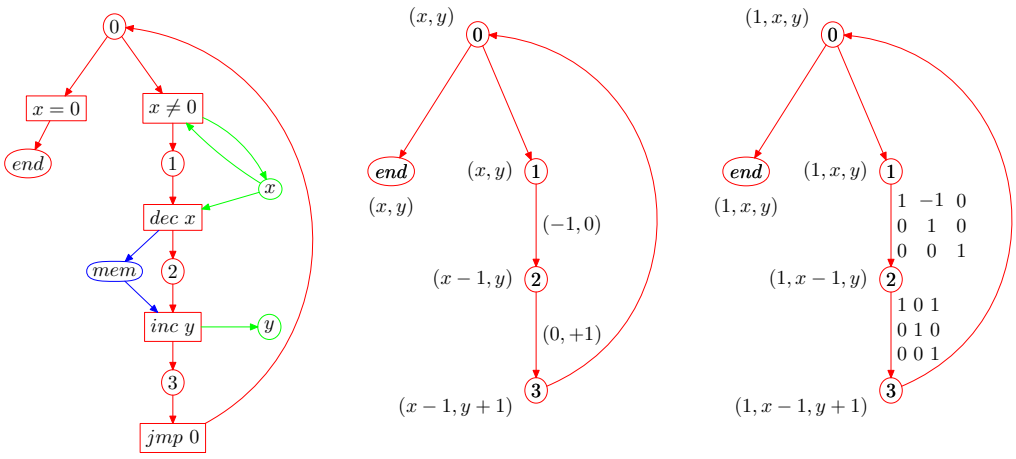

Figure 1: Petri net, VASS and MMSS for the addition.

A Petri net with transition matrix Γ uniformly terminates if and only if the equation $\Gamma X \geq 0$ has no positive solution. See chapter 9.2 of [Moy03] for full details.

In order to deal with memory management, we forget all the variables part. We may then consider the net as a directed weigthed graph. Vertices are labels, edges are instructions and weight are memory management.

Theorem 2. *If the memory graph has no cycle of negative weight, then the program is Non Size Increasing.*

3 Matrices Multiplication Systems

The Petri net modelisation is neat but no good enough. Indeed, two main problems subsist. First of all, we need to insure that only firing sequences with a single token in the control places (the instruction pointer) are taken into account. In [Moy03], chapter 9.3 this condition is enforced with extra equations. We suggest here a better way to deal with this problem.

Instead of modelising the progrma with a Petri net, we will modelise it with a Vector Addition System with States (VASS). That is a directed graph whose edges are weighted by vectors. Then, when taking a path in the VASS, we will have a vector associated to each vertice and every time an edge is followed, its weight is added to the current vector. If this brings a negative component on the vector, then it is forbidden to take that edge. The VASS for addition is shown on the middle part of Figure 1.

Theorem 3. *Again, each execution of the program corresponds to a positive path in the VASS. And again, if the VASS has no such positive path, then the program always terminates.*

Uniform termination of VASS is decidable but a bit more tedious. First of all, we need to reduce it to the inexistance of a cycle of positive weight (in each component), then use the fact that the set of cycles in a graph is a regular language (think of the graph as an automata) and thus, by commutativity of the addition of vectors, the set of weight is a rationnal part of $\mathbb{Z}^n$ which can be expressed as a finite union of semi-linear parts. It is then possible to check if one of this part intersects $\mathbb{N}^n$, that is if there is a cycle of positive weight. all of this can be done in NPTIME.

But even with VASS, a problem subsists. Indeed, we can hardly handle permutation of variables that may happen, typically, when performing a call to a sub-function or upon return of such call. Keeping the idea of a graph with a vector associated to each vertice during paths, it is natural, to perform permutation of values in a vector, to replace vector addition by matrix multiplication. In order to still be able to add or substract a constant, we just need to add to each vector the constant 1 as a new component.

This result in the construction of a Matrices Multiplication System with States (MMSS). The MMSS for addition is shown on the right part of Figure 1.

Again, if the MMSS uniformly terminates, then the program uniformly terminates. We do not know currently if uniform termination of MMSS is decidable.

However, we may notice that if we restrict our matrices to be over a finite ring, then uniform termination is decidable. Among other things, we may see that Size Change Principle [LJBA01] can be seen as a MMSS over the three-element set $\{\downarrow, =, \uparrow\}$ ($\uparrow$ replace the absence of arrow in SCP). Indeed, each elementary size change graph will be a single matrice and the graph algorithm explained in the paper consist of finding an idempotent cycle in the MMSS.

Since SCP is PSPACE-complete, uniform termination of MMSS will probably be PSPACE-hard.

4 Conclusion

This modelisation of program allow to regroup several analysises in one. Indeed, memory management can easily be plugged in MMSS by adding a memory component to each vector. Moreover, if we consider that freeing memory has a weight α and not 1, then we will be able to detect programs which consume $\alpha|x|$ space for inputs of size $|x|$, that is characterise LINSPACE.

Moreover, the graph we build are similar to the trees build in deforestation [Wad90]. Thus, this method also allow to perform a program transformation similar to deforestation, as explained in [Moy03], chapter 9.4.

We still have to study uniform termination of MMSS. If it happen to be undecidable in the general case, then we can still wotlk in the finite case. The possibility to have more than three values should be sufficient to prove termination of more programs than SCP does.

References

[Hof99] Martin Hofmann. Linear types and Non-Size Increasing polynomial time computation. In *Proceedings of the Fourteenth IEEE Symposium on Logic in Computer Science (LICS'99)*, pages 464–473, 1999.

[LJBA01] Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The Size-Change Principle for Program Termination. In *Symposium on Principles of Programming Languages*, volume 28, pages 81–92. ACM press, January 2001.

[MM03] Jean-Yves Marion and Jean-Yves Moyen. Termination and ressource analysis of assembly programs by Petri Nets. Technical report, Loria, 2003.

[Moy03] Jean-Yves Moyen. *Analyse de la complexité et transformation de programmes*. Thèse d'université, Nancy 2, Dec 2003.

[NW] K.-H. Niggl and H. Wunderlich. Certifying polynomial time and linear/polynomial space for imperative programs. *SIAM Journal on Computing*. À apparaître.

[Wad90] Philip Wadler. Deforestation: transforming programs to eliminate trees. *Theoretical Computer Science*, 73(2):231–248, 1990.