Halting Problem of One Binary Horn Clause is Undecidable.*

Philippe Devienne, Patrick Lebègue, Jean–Christophe Routier

Laboratoire d'Informatique Fondamentale de Lille – CNRS U.A. 369
Université des Sciences et Technologies de Lille
59655 Villeneuve d'Ascq Cedex, FRANCE
tel : [33] 20.43.47.18 – fax : [33] 20.43.65.66
{devienne, lebegue, routier} @ lifl.fr

Abstract. This paper proposes a codification of the halting problem of any Turing machine in the form of only one right–linear binary Horn clause as follows :

$$p(t) \leftarrow p(tt) \ .$$

where t (resp. tt) is any (resp. linear) term. Recursivity is well–known to be a crucial and fundamental concept in programming theory. This result proves that in Horn clause languages there is no hope to control it without additional hypotheses even for the simplest recursive schemes.
Some direct consequences are presented here. For instance, there exists an explicitly constructible right–linear binary Horn clause for which no decision algorithm, given a goal, always decides in a finite number of steps whether or not the resolution using this clause is finite. The halting problem of derivations w.r.t. one binary Horn clause had been shown decidable if the goal is ground [SS88] or if the goal is linear [Dev88, Dev90, DLD90]. The undecidability in the non–linear case is an unexpected extension.
The proof of the main result is based on the unpredictable iterations of periodically linear functions defined by J.H. Conway within number theory. Let us note that these new undecidability results are proved w.r.t. any type of resolution (bottom–up or top–down, depth–first or breadth–first, unification with or without occur–check).

1 Introduction

For imperative languages, C. Böhm and G. Jacopini [BJ66] proved that all programming can be done with at most one while loop. A corollary was that the control structures *goto* and *while* have the same expressive power. For term rewriting system (using pattern–matching), Max Dauchet [Dau92] proved that it is possible with only one left–linear rewriting rule to simulate any Turing machine. In comparison to Horn clauses languages, the rewriting of the (supposed ground) goal w.r.t. one rule is non–deterministic because the rule is applied to any sub–term of the goal or not only to the whole goal.

* This work has been partially supported by GRECO de Programmation of CNRS.

Within number theory, other numerous examples can be found, in particular the Hilbert's tenth problem (1900) which was solved by Matijasevits in 1970 and there exists an explicitly constructable universal diophantine equation with degree 4. Another remarkable codification was proposed by J.H. Conway in 1972 by using only periodically linear functions from $\mathbb{N}$ to $\mathbb{N}$. The iterations of these integer functions are generalizations of the famous "$3x+1$" conjecture that we will recall later.

Closer to our study domain, for Horn clauses without function symbols as Datalog languages, a similar codification has been obtained using quasi–iterative programs (that is, each clause contains at most one occurrence of a recursive predicate) [GM87]. For Horn Clause languages, it has been established that all programming can be done with one recursive clause and three facts [PDL91]. The proof, like that of [BJ66], is simple and direct, that is, by an immediate translation of any Horn clause program directly into such a Horn clause program verifying the above form.

The right–linear binary clauses, $p(t) \leftarrow p(tt)$, may induce infinite computation that W. Bibel, S. Hölldobler and J. Würtz [BHW92] call *"cycle unification "*. Within dynamic analysis, some works were done for controlling recursivity [ABK89]. In Section 2, we introduce binary Horn clauses and their resolution, then a codification of the famous "$3x+1$" conjecture is given. In Section 3, a generalization of this conjecture is presented and based on the works of J.H. Conway. In Section 4, we show how the unpredictable iterations of [Con72] can be simulated by binary clauses and we use it to prove the undecidability of halting problem of binary clauses.

2 Binary Clauses

Let F be a set of function symbols (which contains at least one constant and one symbol whose arity is greater than 1) and Var be an infinite countable set of variables, we denote $M(F, Var)$ the set of terms built from F and Var.

Definition 1. The binary (recursive) Horn clauses have the following form :

$$p(t^1, ..., t^n) \leftarrow p(tt^1, ..., tt^n) \ .$$

where t^i and tt^i are any terms of $M(F, Var)$.

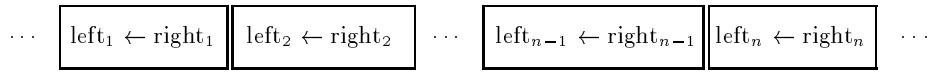
A binary clause is said to be right–linear (resp. left–linear) if all variable occurs at most once in the body part (resp. the head part).

For example, "append$([X \mid L], LL, [X \mid LLL]) \leftarrow$ append(L, LL, LLL)." is a right–linear binary clause.

It is well known that during the resolution, before applying any clause, the formal variables of the clause have been renamed to fresh variables which do not appear anywhere else. The simplest way to do it is to put an additional index on all formal variables, which corresponds, for instance, to the number of the inference.

i^{th} inference : $\mathrm{append}([X_i \mid L_i], LL_i, [X_i \mid LLL_i]) \leftarrow \mathrm{append}(L_i, LL_i, LLL_i)$.

The sequence of inferences using the clause, left $\leftarrow$ right, can be drawn in the form of a series of dominoes :



Like in the domino series, the i^{th} domino can be followed by an $\mathrm{i}+1^{th}$ one, if terms $left_{i+1}$ and $right_i$ can be unifiable and this constraint is compatible with those of the other iterations. Hence, applying n times this binary clause is equivalent to solve the following system :

$$\{ \mathrm{left}_{i+1} = \mathrm{right}_i \mid \forall i \in [1, n-1] \} \ .$$

For example applying n times "append" clause is equivalent to solve the following system :

$$\{\mathrm{append}([X_{i+1} \mid L_{i+1}], LL_{i+1}, [X_{i+1} \mid LLL_{i+1}]) = \mathrm{append}(L_i, LL_i, LLL_i) \mid \forall i \in [1, n-1]\},$$

that is in a solved form :

$$\forall i \in [1, n-1] \begin{cases} L_i = [X_{i+1} \mid L_{i+1}] \\ LL_i = LL_{i+1} \\ LLL_i = [X_{i+1} \mid LLL_{i+1}] \end{cases} .$$

If good intuition is possible about simple binary clauses such as the above one, the non–linearity of the terms, the existence of some variables on one side of the clause, and the permutation of variables during inference generally make intuitive comprehension of behaviour impossible.

The following example shows how difficult the problem of proving termination can be. The exact origin of the Collatz conjecture – also called "Syracuse conjecture" or "$3x+1$ problem" [Lag85] is not clearly known. It had circulated by word of mouth among the mathematical community for many years. This problem is credited to Lothar Collatz at the University of Hamburg. This conjecture asserts that the following program, given any integer n, always terminates.

```
While n > 1 Do
    If n is even
        Then n ← n/2
        Else n ← 3n + 1
    EndIf
EndWhile
```

Nabuo Yoneda at the University of Tokyo has checked it for all $n < 2^{40}$. The behaviour of the Collatz's series seems to be structureless and "random". For instance, from $(2^{500} - 1)$, integers greater than $(2^{500} \times 10^{88})$ are reached.

The Collatz's program can be translated into equivalence relations on $Var \times \mathbb{N}$:

$$\forall k \in \mathbb{N} \ \underline{\text{If}}\ k \text{ is even } \underline{\text{Then}}\ X_k = X_{\frac{k}{2}} \quad \underline{\text{Else}}\ X_k = X_{3k+1} \ .$$

Let f be the function such that $\forall\ i > 0,\ f(2i) = i$ and $f(2i-1) = 6i-2$. Since there does not exist some $k \in \mathbb{N}$ such that $f^k(1) = n\ (\forall\ n > 4)$, we may assert that we may extend the previous relation to the following system of equations :

$$\begin{cases} X_i = X_{2i} \\ X_{2i-1} = X_{3(2i-1)+1} \end{cases}$$

The following binary clause and goal generate such equations :

$$\begin{cases} p([X \mid U], [Y, X \mid V], [\_, \_, \_, Y, \_, \_ \mid W]) \leftarrow p(U, V, W). \\ \leftarrow p(Z, Z, Z). \end{cases}$$

From the general goal $\leftarrow p(L, LL, LLL)$, through the inferences the solved systems of equations increases as :

$$\begin{array}{lll} 1. & L = [X_1 | U_1] & LL = [Y_1, X_1 | V_1] \\ 2. & L = [X_1, X_2 | U_2] & LL = [Y_1, X_1, Y_2, X_2 | V_2] \\ \vdots & \vdots & \vdots \\ n. & L = [X_1, X_2, \cdots, X_n | U_n] & LL = [Y_1, X_1, Y_2, X_2 \cdots, Y_n, X_n | V_n] \end{array}$$

$$\begin{array}{ll} 1. & LLL = [\_, \_, \_, Y_1, \_, \_ | W_1] \\ 2. & LLL = [\_, \_, \_, Y_1, \_, \_, \_, \_, Y_2, \_, \_ | W_2] \\ \vdots & \vdots \\ n. & LLL = [\_, \_, \_, Y_1, \_, \_ \cdots, \_, \_, \_, Y_n, \_, \_ | W_n] \end{array}$$

Then, from the goal $\leftarrow p(Z, Z, Z)$, we force the equalities :

1. $L = LL \ \Rightarrow \ X_{2i-1} = Y_i$ and $X_{2i} = X_i$
2. $L = LLL \ \Rightarrow \ X_{6i-2} = Y_i$.

With a goal of the form :

$$\leftarrow p([\underbrace{a, \_, \cdots, \bar{a}}_{n} \mid L], [\underbrace{a, \_, \cdots, \bar{a}}_{n} \mid L], [\underbrace{a, \_, \cdots, \bar{a}}_{n} \mid L]) \ .$$

we force $X_1 = a$ and $X_n = \bar{a}$. Therefore, the resolution is finite iff a unification fails because of $X_n \neq X_1$, that is, if the "$3x+1$" program is finite from the input n. In other words, the "$3x+1$" conjecture is equivalent to prove that, given any goal $p(L, L, L)$ where L is a list of the form $[a, \_, \cdots, \bar{a} \mid \_]$, the resolution is finite.

3 A Generalization of the "3x+1" Problem

J.H. Conway [Con72] considers the class of periodically piecewise linear functions $g : \mathbb{N} \to \mathbb{N}$ having the structure :

$$\forall\, 0 \leq k \leq d-1,\ \text{if } n \pmod d = k\ ,\ \ g(n) = a_k n\ \ .$$

where $a_0, \cdots, a_{d-1}$ are rational numbers such that $g(n) \in \mathbb{N}$. These are exactly the functions $g : \mathbb{N} \to \mathbb{N}$ such that $\frac{g(n)}{n}$ is periodic. Conway studies the behaviour of the iterates $g^k(n)$ and he states the following theorem :

Theorem 2. (Conway). *If f is any partial recursive function, there is a function g such that :*

1. *$\frac{g(n)}{n}$ is periodic (mod d) for some d and takes rational values.*
2. *$g^{(k)}(2^n) = 2^{f(n)}$ for the minimal $k \geq 1$ such that $g^{(k)}(2^n)$ is a power of 2.*

Principle of proof. Conway's proof uses Minsky machines[Min67], which have the same computational power as Turing machines. He shows that to every Minsky machine, it is possible to associate such a function g which simulates the behaviour of this machine. In fact, he explains how to construct this function g from the Minsky machines.

Since it is undecidable whether or not a given partial recursive function is everywhere defined and identically zero, we obtain the corollary :

Corollary 3. (Conway). *There is no algorithm, which, given a function g with $\frac{g(n)}{n}$ periodic, and given a number n, determines whether or not there is k with $g^k(n) = 1$.*

We are now going to establish some variant of this corollary. To every partial recursive function f, we associate a function f' such that :

$$\begin{cases} f'(0) = 0 \\ f'(x) = f(x-1), \quad \text{if } x-1 \text{ is in the domain of } f \end{cases}$$

It is clear that f' is identically zero iff f is. Since 0 is in the domain of the function f', if we consider the function g', defined as previously, associated to f', there is some $p \in \mathbb{N}^*$ such that :

$$g'^p(2^0) = g'^p(1) = 2^{f'(0)} = 1\ \ .$$

Therefore, we can naturally extend the Corollary 3 :

Corollary 4. *There is no algorithm, which, given a function g with $\frac{g(n)}{n}$ periodic such that $\exists\ p \in \mathbb{N}^*,\ g^p(1) = 1$, and given a number n, determines whether or not there is k with $g^k(n) = 1$.*

4 Partial Recursive Functions and Binary Clauses

In this section, we codify the Conway's generalization from which we deduce the undecidability of the halting problem for one right–linear binary Horn clause.

4.1 Codification of the Conway's Unpredictable Iterations

Proposition 5. *For every periodically piecewise linear function g, there exist a right–linear binary clause $p(t) \leftarrow p(tt)$, a variable X and a goal $\leftarrow p(\gamma)$ such that :*

$$(\{\gamma = t_1\} \cup \{tt_i = t_{i+1} \mid \forall i > 0\})\uparrow_{\{X\}} \equiv \{X_n = X_{g(n)} \mid \forall n > 0\} \ .$$

($\mathcal{S}\uparrow_{\{X\}}$ is the projection onto the variables X_i of the equations expressed in $\mathcal{S}$.)

Lemma 6. *For every natural integers a, a', b, b', there exist two variables X and Y, a right–linear binary clause $p(t) \leftarrow p(tt)$ and a goal $\leftarrow p(\gamma)$ such that :*

$$(\{\gamma = t_1\} \cup \{tt_i = t_{i+1} \mid \forall i > 0\})\uparrow_{\{X,Y\}} \equiv \{X_{ai+b} = Y_{a'i+b'} \mid i > 0\} \ .$$

Proof. The following program :

$$\begin{cases} p([\underbrace{\_, \cdots, \_, Z}_{b}, \overbrace{\_, \cdots, \_}^{a}|L], [X|LL]) \leftarrow p(L, LL). \\ \leftarrow p(L, L). \end{cases}$$

because of the equality of the two arguments in the goal generates : $Z_i = X_{ai+b}$.

By composition of two programs like this one, we obtain :

$$\begin{cases} p([\underbrace{\_, \cdots, Z}_{b}, \overbrace{\_, \cdots}^{a}|L1], [X|L2], [\underbrace{\_, \cdots, Z}_{b'}, \overbrace{\_, \cdots}^{a'}|L3], [Y|L4]) \\ \qquad\qquad \leftarrow p(L1, L2, L3, L4). \\ \leftarrow p(L, L, LL, LL). \end{cases}$$

It involves the equalities :

$$X_{ai+b} = Z_i \qquad \text{and} \qquad Y_{a'i+b'} = Z_i \ .$$

By adding a function symbol it is quite easy to transform a any–arity predicate in a unary predicate. □

Proof of proposition. Let g be a periodically piecewise linear function characterized by $d, a_0, \cdots, a_{d-1}$. g can be decomposed into a finite number of equivalence relations in the form $(X_{ai+b} = Y_{a'i+b'})_{i>0}$. All the right–linear binary clauses and goals which characterized these relations (see Lemma 6) can be merged in one right–linear binary clause and one goal by merging the arguments of these right–linear clauses. □

4.2 The Undecidability Theorems

Notation 7. *We denote* $\forall\, k \in \mathbb{N},\ g^{-k}(n) = \{m \in \mathbb{N} \mid g^k(m) = n\}$.

In order to simplify the proofs, $g^{-k}(n)$ *will represent any integer, if it exists, of this set.*

Theorem 8. *There is no algorithm that, when given a right–linear binary Horn clause and given a goal, always decides in a finite number of steps whether or not the resolution (with or without occur–check) stops.*

Proof. Let us choose a function g such that $\frac{g(n)}{n}$ is periodic and such that there exists $p \in \mathbb{N}^*,\ g^p(1) = 1$ and consider the following systems of equations :

$$\forall\, i \in \mathbb{N}^*,\ X_i = X_{g(i)}\ ;\ X_1 = \bar{a}\ ;\ X_n = a\ . \tag{1}$$

It involves that $\forall\, k \in \mathbb{Z},\ X_{g^k(n)} = a$. Therefore, if – and only if – there exists some $k \in \mathbb{Z}$ such that $g^k(n) = 1$, there is a contradiction between :

$$X_1 = \bar{a} \quad \text{and} \quad X_{g^k(n)} = a\ .$$

Furthermore, since there exists $p \in \mathbb{N}^*$ such that $g^p(1) = 1$ and $k \in \mathbb{Z}$ such that $g^k(n) = 1$ we can claim that there exists $k' \in \mathbb{N}^*$ such that $g^{k'}(n) = 1$. Then Corollary 4 asserts that there is no algorithm that, when given $n \in \mathbb{N}$, decides in a finite number of steps whether or not the above systems of equations (1) produces a contradiction.

According to Proposition 5 and its lemma, we are able to construct a right–linear binary clause $p(t) \leftarrow p(tt)$ such that for some variable X and a goal $\leftarrow p(\gamma)$:

$$(\{\gamma = t_1\} \cup \{tt_i = t_{i+1} \mid \forall i > 0\})\!\uparrow_{\{X\}} \equiv \{X_n = X_{g(n)} \mid \forall n > 0\}\ .$$

In the same way as in the Collatz codification, we can easily add the equations : $X_n = a$ and $X_1 = \bar{a}$ in the goal. Then we construct the following systems of equations :

$$X_i = X_{g(i)}\ ,\ X_1 = \bar{a}\ ,\ X_n = a\ .$$

Hence, we can conclude that there is no algorithm that decides in a finite number of steps whether or not the resolution stops. It is easy to verify that the occur–check will not play any role. □

As an immediate consequence, we can state the following corollary :

Corollary 9. *There is no algorithm that when given a program in the following form :*

$$\begin{cases} p(f) \leftarrow . \\ p(t) \leftarrow p(tt). \end{cases}$$

where f, t, tt *are terms and a goal, "*$\leftarrow p(g).$*", decides in a finite number of steps whether or not there exists a finite number of answer–substitutions.*

Proof. If we consider the program built with :

- the binary Horn clause and the goal defined in the previous proof,
- a fact, "$p(X) \leftarrow .$", where X is a variable.

Through a reasoning similar to the previous one, we can conclude that this problem is undecidable. Indeed, if n is such that there exists $k \in \mathbb{N}$ such that $g^k(n) = 1$ then the program will have a finite number of solutions else an infinite number. □

But it is possible to establish a more stronger form of the previous theorem and corollary :

Theorem 10. *There exists an explicitly constructible, right–linear binary Horn clause for which there is no Turing machine that, when given a goal, always decides in a finite number of steps whether or not the resolution (with or without occur–check) stops.*

Proof. Let us consider one of the partial recursive function ϕ associated to the following program :

```
input f, n      ;where f is any partial recursive function.
compute f(n)
write 0
```

In fact this program answers 0 if and only if n is in the domain of f (i.e. iff f halts with input n). It is possible to characterized the function f by its Gödel number [Rog87]. (In the following, f represents equally the function or the associated Gödel number. The context should clear the ambiguity if any.) Consequently, it is possible to characterized the input of ϕ by the number $2^f 3^n$ and then to consider that ϕ has just one argument. It is obvious that we may impose $\phi(0) = 0$ without change.

In summary, we have a function ϕ such that $\phi(0) = 0$ and such that for every input of the form $2^f 3^n$, ϕ gives 0 iff n is in the domain of f. Then $2^f 3^n$ is in the domain of ϕ iff n is in the domain of f.

Let g be the Conway function associated to ϕ (from the Minsky machine which codes ϕ, we are able to construct g). According to the Conway's theorem, $2^f 3^n$ is in the domain of ϕ iff there exists some $k \in \mathbb{N}^*$ such that $g^k(2^{2^f 3^n}) = 2^{\phi(2^f 3^n)} = 2^0 = 1$. Since it is undecidable to know whether or not n is in the domain of f, it is undecidable to know whether or not $2^f 3^n$ is in the domain of ϕ (i.e. if ϕ halts with the input $2^f 3^n$). Therefore, it is undecidable to know whether or not, for a given n, there exists $k \in \mathbb{N}$ such that $g^k(n) = 1$. Let us note that since $\phi(0) = 0$, we have some $p \in \mathbb{N}^*$ such that $g^p(1) = 1$.

Therefore, as in the proof of Theorem 8, it is possible to construct a right–linear binary clause $p(t) \leftarrow p(tt)$ such that for some variable X and a goal $\leftarrow p(\gamma)$:

$$(\{\gamma = t_1\} \cup \{tt_i = t_{i+1} \mid \forall i > 0\})\uparrow_{\{X\}} \equiv \{X_n = X_{g(n)} \mid \forall n > 0\} \ .$$

Then by adding the equalities $X_n = a$ and $X_1 = \bar{a}$ in the goal, we construct the following systems of equations :

$$X_i = X_{g(i)} \ , \ X_1 = \bar{a} \ , \ X_n = a \ \ .$$

Hence, we can conclude that for this particular right–linear Horn clause, there is no algorithm that decides in a finite number of steps whether or not the resolution stops. It is easy to verify that the occur–check will not play any role. □

Corollary 11. *There is a particular explicitly constructible program in the following form :*

$$\begin{cases} p(f) \leftarrow . \\ p(t) \leftarrow p(tt). \end{cases}$$

where f, t, tt are terms, such that it is undecidable to know whether or not, given a goal, "$\leftarrow p(g).$", there exists a finite number of answer–substitutions.

Proof. The proof is similar to the one of Corollary 9. □

4.3 Some Immediate Consequences

The next result follows directly from the previous corollary :

Corollary 12. *There exists a particular explicitly constructible program, built from a right–linear binary Horn clause, a goal and a fact, for which there is no Turing machine which decides in a finite number of steps whether or not this program is bounded*[2]*.*

Proof. Let us consider the program used in the proof of Corollary 11, since it is undecidable to know whether or not this program has a finite number of solutions (answer–substitutions), the boundedness of this program is undecidable. □

Through a minor modification of the proof of Theorem 10, it is possible to establish the following theorem :

Theorem 13. *There exists an explicitly constructible right–linear binary Horn clause for which it is undecidable to know whether or not, when given a goal, the occur–check will be necessary during the resolution.*

Proof. In the proof of Theorem 10, if we replace the equalities :

$$X_1 = \bar{a} \text{ and } X_n = a$$

by the equalities :

$$X_1 = h(Y, s(Y)) \text{ and } X_n = h(Z, Z) \ \ .$$

It is undecidable to know whether or not the program will stop because of the equalities $Z = Y$ and $Z = s(Y)$, that is because of the occur–check. □

[2] A program is said to be bounded if there exists an equivalent program written only with a finite number of unary clauses.

Remark. Any right–linear binary Horn clause can be simulated by a right–linear binary Horn clause $p(t) \leftarrow p(tt)$ such that *Var(t) = Var(tt)*, that is without local variable.

Proof. By adding a trash–argument to the predicate ([PDL91]) :

$$p(X,Y) \leftarrow p(X,Z) \;\equiv\; p(X,Y,trash([Z \mid L],LL)) \leftarrow p(X,Z,trash(L,[Y \mid LL])) \;.$$

□

5 Conclusion

In this paper we have proved the undecidability of the halting problem for programs with one binary recursive Horn clause and one goal. As an immediate consequence, the undecidability of the existence of a finite or infinite number of solutions for programs built with one binary and two unary clauses, has been proved. The next sensible question is to consider the problem (called cycle unification problem in [BHW92]) of the existence of at least one solution for this class of programs. We recently proved that *cycle unification is undecidable* [DLR92].

The proof ot this last result is based on the same principle using the codification of the Conway functions which are associated to Minsky machines. Roughly speaking, we synchronise two processes in one binary recursive Horn clause,

- the one put a mark at the $(2^n)^{th}$ element of a list, in less that 2^n iterations steps, iff n is in the domain of Minsky machine $\mathcal{M}$,
- the other put a distinct mark at the $(2^n)^{th}$ element of another list at the $(2^n)^{th}$ iterations step

The fact is chosen such that at the p^{th} iteration :

- if $p \neq 2^n$ there is no solution,
- if $p = 2^n$ we unify the $(2^n)^{th}$ elements of the two lists.

Consequently, we will have solutions iff there exists n such that n is not in the domain of $\mathcal{M}$. Hence, this program will have solutions iff $\mathcal{M}$ is not total, this is of course undecidable.

Acknowledgements : We would like to thank Prof. Max Dauchet and Prof. Jean–Paul Delahaye for their illuminating discussion and their helpful collaboration, Eric Wegrzynowski and Benham Bani–Eqbal for their attentive and clever readinds. We would like as well to thank anonymous referees for their helpful comments.

References

[ABK89] Apt K.R., Bol R.N., Klop J.W. "On the safe termination of PROLOG programs." *ICLP'89, Lisbon, pp. 353–368.* 1989.

[BHW92] Bibel W., Hölldobler S., Würtz J. "Cycle Unification." *CADE pp. 94–108.* June 1992.

[BJ66] Böhm C., Jacopini G. "Flow diagrams, Turing machines and languages with only two formation rules." *Communications of the Association for Computing Machinery, Vol.9, pp. 366–371.* 1966.

[Con72] Conway J.H. "Unpredictable Iterations." *Proc. 1972 Number Theory Conference. University of Colorado, pp 49–52.* 1972.

[Dau92] Dauchet M. "Simulation of Turing Machines by a regular rewrite rule." *Journal of Theoretical Computer Science. n°103. pp. 409–420* 1992.

[Dev88] Devienne P. "Weighted graphs – tool for studying the halting problem and time complexity in term rewriting systems and logic programming (extended abstract)." *Fifth Generation Computer Systems 88, Tokyo, Japan.* 1988.

[Dev90] Devienne P. "Weighted graphs – tool for studying the halting problem and time complexity in term rewriting systems and logic programming." *Journal of Theoretical Computer Science, n°75, pp. 157–215.* 1990.

[DLD90] Devienne P., Lebègue P., Dauchet M. "Weighted Systems of Equations." *Informatika 91, Grenoble, Special issue of TCS.* 1991.

[DLR92] Devienne P., Lebègue P., Routier J.C. "Cycle Unification is Undecidable." *LIFL Technical Report n°IT 241, Lille.* 1992.

[GM87] Gaiman, Mairson "Undecidable optimisation problems for database logic programs." *Symposium on Logic in Computer Science, New–York, pp. 106–115.* 1987.

[Lag85] Lagarias J.C. "The $3x + 1$ problem and its generalizations." *Amer. Math Monthly 92, pp. 3–23.* 1985.

[Min67] Minsky M. *"Computation : Finite and Infinite Machines."* Prentice–Hall. 1967.

[PDL91] Parrain A., Devienne P., Lebègue P. "Prolog programs transformations and Meta–Interpreters." *Logic program synthesis and transformation, Springer–Verlag, LOPSTR'91, Manchester.* 1991.

[Rog87] Rogers H. *"Theory of Recursive Functions and Effective Computability."* The MIT Press. 1987.

[SS88] Schmidt–Schauss M. "Implication of clauses is undecidable." *Journal of Theoretical Computer Science, n°59, pp. 287–296.* 1988.