The Emptiness Problem of One Binary Recursive Horn Clause is Undecidable

Philippe Devienne, Patrick Lebègue, Jean-Christophe Routier
Laboratoire d'Informatique Fondamentale de Lille – UA CNRS 369
Université des Sciences et Technologies de Lille
59655 Villeneuve d'Ascq Cedex, FRANCE
{devienne,lebegue,routier}@lifl.fr

Abstract

The simplest recursive program in Horn clause languages is of the form :

$$\begin{cases} p(fact) \leftarrow & . \\ p(left) \leftarrow p(right) & . \\ \leftarrow p(goal) & . \end{cases}$$

This corresponds to *append*–like programs.

The two most relevant problems concerning this class are the halting and the emptiness (existence of at least one solution) problems. The halting problem has been proved undecidable in the general case in [7]. Here we establish the undecidability of the emptiness problem in the general case.

The (non–)linearity (each variable occurs at most once) of the terms *fact*, *left*, *right* and *goal* is crucial. We prove that as soon as three of them are linear, the emptiness problem becomes decidable. For the halting problem, the linearity of *goal* or *left* is sufficient.

Moreover, the undecidability of the emptiness problem implies the unsatisfiability of the class of quantificational formulas with one 2-clause and two unit clauses which was opened for twenty years.

1 Introduction

Quantificational formulas have been subject to a lot of works. A class of quantificational formulas is said to be decidable if and only if there exists an effective procedure that determines, for each formula in the class, whether or not it is satisfiable.

Considering the quantificational formulas with a small number of subformulas, the satisfiability of the class

$$\forall t_1, t_2, t_3, t_4 [P(t_1) \wedge (Q(t_2) \vee R(t_3)) \wedge S(t_4)]$$

where P, Q, R and S are any positive or negative predicates, is open [9]. A particular subclass (the only interesting one) is the following one :

$$\forall t_1, t_2, t_3, t_4 [P(t_1) \wedge (P(t_2) \vee \neg P(t_3)) \wedge \neg P(t_4)]$$

for which the problem of the consistency corresponds to the problem ot the existence of solution for the Prolog program :

$$\begin{cases} P(t_1) \leftarrow . \\ P(t_2) \leftarrow P(t_3). \\ \leftarrow P(t_4). \end{cases}$$

That is the class of the simplest recursive Prolog programs built from one fact, one goal and one binary recursive Horn clause. A representative member of this class is the most famous example in Prolog community : *Example 1.*

$$\begin{cases} append([\,], L, L) \leftarrow \ . \\ append([H|L], LL, [H|LLL]) \leftarrow append(L, LL, LLL) \ . \\ \leftarrow append(?, ?, ?) \ . \end{cases}$$

□

Concerning this class, the two most relevant problems are the halting problem and the emptiness problem, that is the problem of the existence of at least one solution. Different behaviours are possible : finite or infinite computation ; null, finite or infinite number of solutions. The computational power of this class is also interesting.

M. Schmidt–Schauss [17] has shown that the two problems are decidable when *goal* and *fact* are ground[1]. M. Dauchet, P. Devienne and P. Lebègue [4] [6] studied the linear[2] case and proved it decidable as well. W. Bibel, S. Hölldobler and J. Würtz [2] have considered the emptiness problem and have proved it decidable for some particular cases (see also [16, 18]).

In [7], we have proved the halting problem to be undecidable in the general case. In this paper, using a similar proof technique based on the codification of the unpredictable iterations of J.H. Conway within number theory [3] which code Minsky machines [14], we will show that the emptiness problem is undecidable in the general case (another proof, established independently at the same time, of this result can be found in [10]). Let us note, that although the basic technique (our (original) codification of Conway functions) is the same as in our former paper, it is used differently. We study as well some particular subclasses depending on the (non–)linearity of the terms.

In next section, we introduce binary Horn clauses and their resolution. In Section 3, we present the Minsky machines formalism and the Conway unpredictable iterations. It is shown how they can be simulated by binary clauses in Section 4. We present the main result in Section 5 and some subcases depending on the linearity of the terms in Section 6. The last section summarizes the results. Some proofs are presented in Annex.

[1] A term t is said to be *ground* when it does not contain any variable occurence [13].
[2] A term t is said to be *linear* when each variable occurs at most once.

2 Preliminaries

2.1 Binary Horn Clause

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.

2.2 Variable Indexation

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 indice on all formal variables, which corresponds, for instance, to the number of the inference.

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

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

$$\cdots \boxed{\mathrm{left}_1 \leftarrow \mathrm{right}_1}\boxed{\mathrm{left}_2 \leftarrow \mathrm{right}_2} \cdots \boxed{\mathrm{left}_{n-1} \leftarrow \mathrm{right}_{n-1}}\boxed{\mathrm{left}_n \leftarrow \mathrm{right}_n} \cdots$$

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 :

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

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

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

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.

3 Theoretical Tools

3.1 Minsky Machines

3.1.1 Presentation

The Minsky machines [14, 3] are state–register machines, the registers (in finite number) may hold non–negative integers and two types of transitions are allowed :

- "in the state Q_i, add 1 to register a and proceed to state Q_j".
- "in the state Q_i, if $\mid a \mid > 0$ (where $\mid a \mid$ denotes the content of the register a) then substract 1 to register a and proceed to state Q_j, else simply proceed to Q_k".

These machines have the same computational power as Turing machines (two registers are sufficient [14]). Indeed, for any partial recursive function f, there is a Minsky machine which started with register contents $n, 0, 0, \cdots$ ends with register contents $f(n), 0, 0, \cdots$.
Let us recall some usual definitions and properties :

- The domain of a Minsky machine $\mathcal{M}$ is : $\{n \in \mathrm{N} \mid \mathcal{M}(n) \text{ is finite}\}$.
- A Minsky machine $\mathcal{M}$ is said to be total iff its domain is N.
- It is undecidable to determine whether, given a Minsky machine, this machine is total or not.

3.1.2 A particular class of Minsky machines

In the proofs of Section 5, we use a particular class of Minsky machines defined by the two following definitions.

Definition 2 *A Minsky machine $\mathcal{M}$ is said to be null if :*

- $0 \in Dom(\mathcal{M})$

- *all the registers are null at the final computation–state, that means that the associated partial function, f, verifies : $\forall\ n \in Dom(f),\ f(n) = 0$*

Definition 3 *A Minsky machine $\mathcal{M}$ is said to be linear if there exists α a natural integer such that for all input $n \in Dom(\mathcal{M})$, if $n > 0$ then $\mathcal{M}(n)$ is computed in less than $\alpha \times n$ steps.*

It is easy to see that such Minsky machines exist. Given an integer α greater than 0, the set of α–linear and null Minsky machines is infinite.

Theorem 3.1 *There is no algorithm that, when given a linear and null Minsky machine $\mathcal{M}$, always decides in a finite number of steps whether or not $\mathcal{M}$ is total.*

Proof. in Annex □

Definition 4 *A recursive set Σ_r is said to be linear if there exists a linear and null Minsky machine of which domain is Σ_r*

Corollary 3.2 *Knowing that a linear recursive set is equal to* N *is undecidable.*

Proof. By application of Theorem 3.1. □

3.2 Conway Unpredictable Iterations

Here we present some work by J.H. Conway [3] which has studied a generalization of the Collatz conjecture. The exact origin of this conjecture – also called "Syracuse conjecture" or "$3x + 1$ problem" [11] – 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 opposite 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
```

3.2.1 Presentation

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

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

where $a_0, \cdots, a_{d-1}$ are rational numbers such that $g(n) \in \mathrm{N}$. These are exactly the functions $g : \mathrm{N} \to \mathrm{N}$ 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 3.3 (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. $\forall\ n \in \mathrm{N}, n \in Dom(f)$ *iff* $\exists (k,j) \in \mathrm{N}^* \times \mathrm{N},\ g^{(k)}(2^n) = 2^j$.
3. $g^{(k)}(2^n) = 2^{f(n)}$ *for the minimal* $k \geq 1$ *such that* $g^{(k)}(2^n)$ *is a power of* 2.

Remark. By construction, the number of iterations used from $g(2^n)$ to $2^{f(n)}$ is equal to the number of transitions used by $\mathcal{M}$ from n to $f(n)$.

3.2.2 Conway Equivalence Relations

We study the null Conway functions and define some equivalence relations based on these particular functions.

Definition 5 *Let g be a Conway function, the domain of g is :*

$$Dom(g) \ = \ \{n \in \mathrm{N} \ \mid \ \exists (k,p) \in \mathrm{N}^* \times \mathrm{N}, g^{(k)}(2^n) = 2^p\}$$

A Conway function g is said to be total if its domain is N.

Let us consider the class of linear and null Minsky machines and their associated Conway functions, called also *linear and null Conway functions.*

Proposition 3.4 *Let g be a null Conway function then*[3] *:*

$$\exists k \in \mathrm{N},\ 2^0 \in g^{(-k)}(2^n) \ \ \Rightarrow \exists k' \in \mathrm{N},\ g^{(k')}(2^n) = 2^0 \ .$$

Proof. in Annex □

Because of the features of the null Conway functions, negative iterations of function g can be taken into account. In other words, the Conway transitions $n \to g(n)$ will be extended to equivalence relations $n \equiv g(n)$. Indeed according to the definition of these functions, the only power of 2 you can reach by iterating g from any 2^n is 2^0 (and you reached it iff $n \in Dom(f)$). Conversely, by iterating $g^{(-1)}$ from 2^0 the only powers of 2 you reach (and you reach all of them), are the 2^n such that $n \in Dom(f)$. So we can consider that, ignoring loops on 2^0, there is *only one* path form 2^n to 2^0 (if any) using positive iterations of g and therefore *only one* path (the *same* in reverse sense) from 2^0 to 2^n using negative iterates.

Definition 6 *A Conway equivalence relation is defined from a null Conway function and its basic equivalence relations :* $\forall\ n \in \mathrm{N},\ n \equiv_g g(n)$

Corollary 3.5 *For every recursively enumerable set,* Σ *containing* $\{0\}$*, there exists a Conway equivalence relation* $\equiv_g$ *such that :* $\Sigma = \{n \in \mathrm{N} \mid 2^n \equiv_g 1\}$

Proof. It is obvious that for every recursively enumerable set, Σ containing $\{0\}$, there exists a null Minsky machine $\mathcal{M}$ which domain is Σ. Then if g is the null Conway function associated with $\mathcal{M}$, $\equiv_g$ satisfies the assertion. □

[3] We denote $\forall\ k \in \mathrm{N},\ g^{(-k)}(n) = \{m \in \mathrm{N} \ \mid \ g^k(m) = n\}$.

4 Recursively Enumerable Set and Binary Horn Clause

Let us show that the SLD resolution of a goal w.r.t. one binary Horn clause can codify any recursively enumerable subset of N containing $\{0\}$.

Theorem 4.1 *Let $\sharp$ be a special symbol. For every recursively enumerable set Σ containing $\{0\}$, there exist a right–linear binary Horn clause and a goal such that any natural integer n, belongs to Σ iff from a certain number of SLD resolution steps, the first argument of the initial goal must be a list of which the $(2^n)^{th}$ element is marked by $\sharp$.*

Remark. The following program is an illustration in the case where Σ is N :

$$\Pi_1 \quad \begin{cases} p([X|L],[Y,X|LL]) \leftarrow p(L,LL) \ . \\ \leftarrow p([\sharp|L],[\sharp|L]) \ . \end{cases}$$

This program put a $\sharp$ in all the $(2^n)^{th}$ positions of $[\sharp|L]$. Let us note that the propagation of the mark is here "1–linear", that is, the $(2^n)^{th}$ element will be marked after at most 2^n SLD–resolution steps.

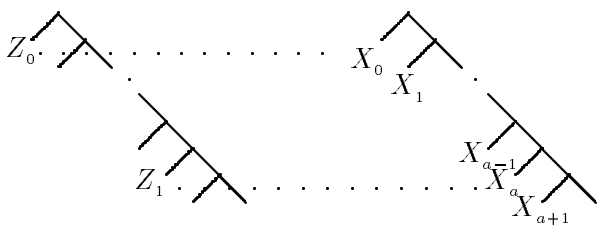
Lemma 4.2 *For every natural integers a, a', b, b', there exist a variable X, a right–linear binary clause $p(t) \leftarrow p(tt)$ and a goal $\leftarrow p(\gamma)$ such that :*

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

Proof. The following program :

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

The size of the first variable of the Horn clause decreases in a while the one of the second decreases in one, so we have :



If there was no b in the goal, the equality of the two arguments would have generate : $Z_i = X_{ai}$, the b shifts this equation then we have : $Z_i = X_{ai+b}$.

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

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

It involves the equalities :

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

□

Remark. If we want to code the relation $X_{ai+b} = X_{a'i+b'}$ with $b < a$ and $b' < a'$, it is possible to write :

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

Lemma 4.3 *For every Conway function g, there exist a variable X, a right–linear binary clause $p(t) \leftarrow p(tt)$, and a goal $\leftarrow p(\gamma)$ such that :*

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

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

Proof. in Annex □

Proof of the theorem. According to the previous lemmas, let X be the variable which codes the Conway equivalence relation of Σ (as told in Corollary 3.5), then the list L is linearly built as $[X_1, X_2, \cdots, X_n, \cdots]$ with all the X_i linked by the relations $X_i = X_{g(i)}$. Consequently, according to Corollary 3.5 :

$$\Sigma = \{n \in \mathrm{N} \mid X_{2^n} \equiv_g X_1\}$$

Moreover at startup, if variable X_1 is marked by $\sharp$, then this mark will be propagated to all X_{2^n} where n belongs to Σ. □

The next example is an illustration of Lemma 4.2. It shows that binary Horn clauses can produce relations like $X_{ai+b} = Y_{ci+d}$ and thus easily express very complex problem like, here, the Collatz problem.

Example 2. Back to the Collatz conjecture, the Collatz's program can be translated into equivalence relations on $Var \times \mathrm{N}$:

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

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 $k \in \mathrm{N}$ 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 :

$$\begin{cases} p(\overbrace{[X \mid U]}^{L_1}, \overbrace{[Y, X \mid V]}^{L_2}, \overbrace{[\_, \_, \_, Y, \_, \_ \mid W]}^{L_3}) \leftarrow p(U, V, W). \\ \leftarrow p(Z, Z, Z). \end{cases}$$

Through the inferences the solved systems of equations increases as :

$$\begin{array}{lllllllll} L_2 = & [Y_1, & X_1, & Y_2, & X_2, & Y_3, & X_3, & \cdots, Y_n, X_n | V_n] \\ L_1 = & [X_1, & X_2, & X_3, & X_4, & X_5, & X_6, & \cdots, X_{n-1}, X_n | U_n] \\ L_3 = & [\_, & \_, & \_, & Y_1, & \_, & \_, & \cdots, \_, Y_n, \_, \_ | W_n] \end{array}$$

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

1. $L_1 = L_2 \ \Rightarrow\ X_{2i-1} = Y_i$ and $X_{2i} = X_i$
2. $L_1 = L_3 \ \Rightarrow\ X_{6i-2} = Y_i$.

that is

$$X_i = X_{2i} \text{ and } X_{2i-1} = X_{6i-2}$$

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. □

We use the previous theorem and the undecidability of the belonging of an element to a recursive enumerable set to prove :

Theorem 4.4 *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. It is a direct consequence of Theorem 4.1. By initializing L in the goal as $[\sharp, X_2, \cdots, X_{2^n-1}, \flat \mid LL]$ where mark $\flat$ is put on the $(2^n)^{th}$ element of L, then the resolution stops iff equation $(\sharp = \flat)$ occurs, that is, iff n is an element of Σ. Since there is no algorithm that, when given an integer n and a recursively enumerable set Σ, always decides in a finite number of steps whether or not n belongs to Σ, the result is proved. It is easy to check that the occur–check does not play any role in the proof. □

This result was first established in [7] with a slightly different proof.

5 Emptiness Problem

Remark. Another proof, established independently at the same time, of this problem can be found in [10], it is based on a codification of the Post correspondence problem into an *append*–like program.

Let us suppose that the Conway equivalence relation is "linear", then the propagation of mark $\sharp$ is linear too. Then it is possible to write a program for which a solution at the $(2^n)^{th}$ step is equivalent to say that n does not belong to the linear recursive set. Therefore, we cannot decide whether or not such Horn clauses have no solution because of Corollary 3.2.

Theorem 5.1 *There is no algorithm such that, when given a program of the following form :*

$$\begin{cases} p(fact) \leftarrow \ . \\ p(left) \leftarrow p(right) \ . \\ \leftarrow p(goal) \ . \end{cases}$$

where $fact$, $right$ are linear terms, always decides in a finite number of steps, whether or not this program has at least one solution.

Lemma 5.2 *For every linear recursive set Σ_r (containing $\{0\}$), there exist a right–linear binary clause and a goal such that any natural integer, n, belongs to Σ_r iff after at most 2^n SLD resolution steps, the first argument of the initial goal must be a list of which $(2^n)^{th}$ element is marked by $\sharp$.*

Proof. Let Σ_r be a linear recursive set, by definition, there exists a linear and null Conway function of which domain is Σ_r. It is easy to check that the associated Conway equivalence relation is at worst α–linearly computed by the binary Horn clause C_2 and the goal obtained by the Lemma 4.3 codification. In other words, mark $\sharp$ is linearly propagated in the first list–argument of the goal. It is now easy to define a "1–linear" pair (binary Horn clause C_1, goal) from this "α–linear" pair (C_2, goal), each resolution step of C_1 corresponding to α steps of C_2. □

Proof of the theorem. Let us write a right–linear binary Horn clause and a goal for which resolution constructs a characteristic list of powers of 2 :

$$\Pi_2 \quad \begin{cases} p([X|L], [Y, X|LL], [\flat, Z|LLL]) \leftarrow p(L, LL, LLL) \ . \\ \leftarrow p([\sharp, \sharp|L], [\sharp, \sharp|L], L) \ . \end{cases}$$

In comparison to example Π_1, a third argument has been added in order to instanciate the non–powers of 2 with $\flat$. This program is such that after n resolution steps the first argument $\mathcal{L}$ is :

$$\mathcal{L} = [X_1, X_2, \cdots, X_n, ...]$$

where $\forall\ k < n,\ X_k = \sharp$ if k is a power of 2 and $X_k = \flat$ otherwise.

Let us code now a class of programs for which the existence of solutions is undecidable. Let Σ_r be any linear recursive set, and its characteristic pair (right–linear Horn clause, goal) (Cf. Lemma 5.2), let us denote it as follows :

$$\Pi_3 \quad \begin{cases} p(t_1, t_2, \cdots, t_k) \leftarrow p(tt_1, tt_2, \cdots, tt_k) \ . \\ \leftarrow p(g_1, g_2, \cdots, g_k) \ . \end{cases}$$

Now follows our particular class of programs :

$$\Pi_4 \quad \begin{cases} p(Y_1, Y_2, \cdots, Y_k, \flat, Z, [\sharp|L], LL, LLL) \leftarrow \ . \\ p(t_1, t_2. \cdots, t_k, W, [U|V], [X|L], [Y, X|LL], [\flat, Z|LLL]) \\ \qquad\qquad \leftarrow p(tt_1, tt_2, \cdots, tt_k, U, V, L, LL, LLL) \ . \\ \leftarrow p(g_1, g_2, \cdots, g_k, X, g_1, [\sharp, \sharp|L], [\sharp, \sharp|L], L) \ . \end{cases}$$

The n first arguments codify Σ_r, the two following arguments allow to extract, at the n^{th} iteration, the n^{th} argument of the characteristic list of Σ_r and the three last arguments codify the characteristic list of powers of 2. Because of the fact, there is a solution at the n^{th} step iff :

- n is a power of 2 (the three last arguments)
- the n^{th} element of the characteristic list of Σ_r is not marked by $\sharp$, because it must be unifiable with $\flat$ (the $(n+2)$ first arguments)

In other words, since we know that the marking (by $\sharp$) is "1–linear", there is a solution at the $(2^n)^{th}$ step iff n does not belong to Σ_r. Therefore, Π_4 has no solution iff Σ_r is equal to N. According to Corollary 3.2, that is undecidable. □

Because of the symmetry of the problem, we can strengthen the previous theorem in an obvious way :

Corollary 5.3 *There is no algorithm such that, when given a program of the following form :*

$$\begin{cases} p(fact) \leftarrow \ . \\ p(left) \leftarrow p(right) \ . \\ \leftarrow p(goal) \ . \end{cases}$$

where goal and left are linear always decides in a finite number of steps, whether or not this program has at least one solution.

Proof. Because of the symmetry of the problem. □

As a consequence, we can establish that :

Theorem 5.4 *The class of quantificational formulas with four subformulas is undecidable with respect to the consistency. There is no algorithm which, given a quantificational formula with four subformulas, decides in a finite number of steps, whether or not the formula is consistent.*

This result solves the last open problem in mathematical logic concerning the satisfiability of quantificational formulas with a small number of subformulas. The 5-formulas case was solved in [9].

6 Linear Horn Clause and Other Subcases

We have proved that when the Horn clause is right– *or* left–linear, the problems were undecidable. Now it is natural to study the behaviour of this small program depending on the linearity of the terms *goal*, *left*, *right* and *fact*. We prove that the halting problem becomes decidable as soon *goal* or *left* are linear. The emptiness problem remains undecidable in the linear Horn clause case. The proof in the first case is based on the weighted graphs [4, 5, 12, 6]. In the second case, we use the same method as for Theorem 5.1, we simply transform any *append*–like program in an equivalent one by linearizing the Horn clause. We are not going to give the detailed proofs here, they will appear soon in a extended report and can be actually communicated to all interested people.

So we state :

Theorem 6.1 *There exists an algorithm that, when given a left–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.*

Theorem 6.2 *For the class of programs :*

$$\begin{cases} p(fact) \leftarrow \ . \\ p(left) \leftarrow p(right) \ . \\ \leftarrow p(goal) \ . \end{cases}$$

where

1. *left, right are linear and fact and goal are any terms, the emptiness problem is undecidable.*
2. *left, right and fact (resp. goal) are linear and goal (resp. fact) is any, the emptiness problem is decidable.*
3. *fact or goal is ground, the emptiness problem is decidable.*

Sketch of Proof.

1. Let us consider the following program :

$$\begin{cases} p(\_,\_,\_,L,L) \leftarrow \quad . \\ p([X|LX],[\overbrace{U,\_,\cdots,\_}^{a}|LU],[\overbrace{V,\_,\cdots,\_}^{c}|LV],LLU,LLV) \\ \qquad\qquad \leftarrow p(LX,LU,LV,[U|LLU],[V|LLV]) \quad . \\ \leftarrow p(L,[\underbrace{\_,\cdots,\_}_{b}|L],[\underbrace{\_,\cdots,\_}_{d}|L],[],[]) \quad . \end{cases}$$

   it produces the equality :

$$X_{ai+b} = X_{ci+d}$$

   And no other different relation on X is defined.
   It is easy to extend it to create several different equalities on X, and therefore to the codification of Conway equivalence relations.
   Now it is quite clear that a proof similar to the one of Theorem 5.1 can be made to prove the result in the linear clause case.

2. using similar proof's method to the one of Theorem 6.1.

3. based on weighted graphs formalism.

□

7 Summarize

The below tabulars summarize the known results concerning the halting and emptiness problems depending on the form of the characteristic elements *goal*, *fact*, *left* and *right* for *append*–like programs

goal	*left*	*right*	Halting Problem
ground	any	any	decidable [17]
linear	any	any	decidable [6]
any	linear	any	decidable[here]
any	any	linear	undecidable [7]

goal	*left*	*right*	*fact*	Emptiness Problem
ground	any	any	ground	decidable [17]
linear	any	any	linear	decidable [6]
ground	any	any	any	decidable [here]
any	any	any	ground	
linear	linear	linear	any	decidable [here]
any	linear	linear	linear	
any	any	linear	linear	undecidable [10][here]
linear	linear	any	any	
any	linear	linear	any	undecidable[here]

Linearity seems to mark the border between decidability and undecidability. Concerning halting problem, as soon as *goal* or *left* are linear, it becomes decidable. For the emptiness problem, three linear terms insure the decidability.

The technique based on our original codification of the Conway functions provides an homogeneous frameproof concerning the study of the binary recursive Horn clauses. Indeed, it allows to solve the halting and emptiness problems and a lot of other properties (boudedness,...). Recently, using this codification (and some other results), we have proved that the *append*–like programs have the same computational power as Turing machines [8]. This result can be seen as the Böhm–Jacopini [1] theorem to Logic Programming.

Endly, our problem seems to be closed to implication of clause. Considering the program :

$$\begin{cases} p(fact) \leftarrow \ . \\ p(left) \leftarrow p(right_1), p(right_2) \ . \\ \leftarrow p(goal) \ . \end{cases}$$

where $fact$ and $goal$ are ground. The existence of solutions for this program is equivalent to the implication of clause $A \Rightarrow B$. Where A is $left \leftarrow right_1, right_2$ and B is $fact \leftarrow goal$. The problem has been shown undecidable for the case where B is any in [15]. We hope our technique can solve this particular case.

Acknowledgements : We would like to thank Prof. Jean–Paul Delahaye, the basic idea of proof of Theorem 3.1 is due to him.

References

[1] 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.

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

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

[4] Dauchet M., Devienne P., Lebègue P. "Weighted Graphs : a Tool for Logic Programming." *11th Colloquium on Trees in Algebra and Programming (CAAP86).* 1986.

[5] Devienne P. "Les Graphes orientés pondérés : un outil pour l'étude de la terminaison et de la complexité dans les systèmes de réécritures et en programmation logique." *Ph. D. Thesis. Lille.* 1987.

[6] 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.

[7] Devienne P., Lebègue P., Routier J.C. "Halting Problem of One Binary Horn Clause is Undecidable." Proceedings of STACS'93, LNCS n°665, pp. 48–57, Springer–Verlag. Würzburg. 1993.

[8] Devienne P., Lebègue P., Routier J.C., Würtz J. " The Böhm–Jacopini Theorem for Horn Clause Languages." *Technical Report IT 252. LIFL. Lille.* Juin 1993.

[9] Goldfarb W., Lewis H.R. "The decision problem for formulas with a small number of atomic subformulas" *J. Symbolic Logic 38(3)*, pp.471–480, 1973.

[10] Hanschke P., Würtz J. "Satisfiability of the Smallest Binary Program." *Information Processing Letters, vol. 45, n° 5. pp. 237–241.* April 1993.

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

[12] Lebègue P. "Contribution à l'Etude de la Programmation Logique par les Graphes Orientés Pondérés" *Ph. D. Thesis. Lille.* 1988.

[13] Lloyd J.W. *"Foundations of Logic Programming." Second, Extended Edition* Springer–Verlag. 1987. Springer Verlag

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

[15] Marcinkowski J., Leszek Pacholski "Undecidability of the Horn–Clause Implication Problem" *Proc. of the 33rd FOCS. 1992.*

[16] Salzer G. "Solvable Classes of Cycle Unification Problems." *IMYCS, Smolenice (CSFR).* 1992.

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

[18] Würtz J. "Unifying Cycles." *Proceedings of the European Conference on Artificial Intelligence. pp. 60–64.* August 1992.

8 Annex

Proof of Theorem 3.1 From every Minsky machine $\mathcal{M}_?$ with zero as input, $\mathcal{M}$ a null Minsky machine can be defined as follows.

Let α be a natural integer, and n be the input of $\mathcal{M}$,

1. Compute $\alpha \times n$ and put it in a new register r.
2. if $\mid r \mid = 0$ then goto 5 else substract one from r and continue
3. Proceed one computation–step of $\mathcal{M}_?(0)$
4. If $\mathcal{M}_?(0)$ have reached one of its final computation–state then go into "an infinite loop" else goto 2
5. end : put zero in all the registers and halt.

Any natural integer n belongs to the domain of $\mathcal{M}$ iff the "infinite loop" is not reached, that is, $\mathcal{M}_?(0)$ is computed in more than $\alpha \times n$ steps. Thus by construction this null Minsky machine is total iff $\mathcal{M}_?(0)$ does not stop.

Let us compute the complexity of $\mathcal{M}$ for any $n \in Dom(\mathcal{M})$. Step 1. may be done in $(\alpha \times n)$ transitions, $\mathcal{M}$ reach step 5. after $(2\alpha \times n)$ transitions. Once in this step, the sum of all the contents of the k registers of $\mathcal{M}_?$ is, by construction, at most $(\alpha \times n)$. Consequently it takes at worst $(\alpha \times n + k)$ transitions to put 0 in all the registers of $\mathcal{M}_?$ then of $\mathcal{M}$. Hence the complexity of $\mathcal{M}$ is $(4\alpha \times n + k)$.

Thus by construction, the null Minsky machine $\mathcal{M}$ is linear and is total iff $\mathcal{M}_?(0)$ does not stop, that is undecidable. □

Proof of Proposition 3.4 Since 0 belongs to the domain, there exists $k > 0$ such that $g^{(k)}(2^0) = 2^0$. Thus, if there exists $k' > 0$ such that $2^0 \in g^{(-k')}(2^n)$ then by definition :

$$g^{(k')}(2^0) = 2^n \text{ and } \forall\ \alpha > 0, g^{(-k'+k'+\alpha\times k)}(2^0) = 2^0.$$

Hence for all $\alpha > 0$ we have :

$$g^{(-k'+\alpha\times k)}(g^{(k')}(2^0)) = g^{(-k'+\alpha\times k)}(2^n) = 2^0$$

Since there exists $\alpha_0 > 0$ such that $-k' + \alpha_0 \times k > 0$, we have the result. □

Proof of Lemma 4.3 Let g be a periodically piecewise linear function defined by $d, a_0, \cdots, a_{d-1}$. For all $n = \alpha d + k$ $(0 \leq k \leq d)$, we have $g(n) = g(\alpha d + k) = a_k n = (a_k d)\alpha + k a_k$, with $(a_k d, k a_k) \in \mathbf{N}^2$. Then the $(X_n = X_{g(n)})_{n\in\mathbf{N}}$ can be decomposed into a finite number of equivalence relations in the form $(X_{ai+b} = X_{a'i+b'})_{i>0}$. All the right–linear binary clauses and goals which characterized these relations (see Lemma 4.2) can be merged in one right–linear binary clause and one goal by merging their arguments. □