One Binary Horn Clause is Enough

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

Jörg Würtz
Deutsches Forschungszentrum für Künstliche Intelligenz – DFKI,
Stuhlsatzenhausweg 3, 66123 Saarbrücken 11, Germany
wuertz@dfki.uni-sb.de

Topics : Logic in Computer Science, Theory of Programming Languages.

Abstract. This paper proposes an equivalent form of the famous Böhm-Jacopini theorem for declarative languages. C. Böhm and G. Jacopini [1] proved that all programming can be done with at most one single while-do. That result is cited as a mathematical justification for structured programming. A similar result can be shown for declarative programming. Indeed the simplest class of recursive programs in Horn clause languages can be defined by the following scheme :

$$\begin{cases} \mathcal{A}_1 \leftarrow . \\ \mathcal{A}_2 \leftarrow \mathcal{A}_3. \\ \leftarrow \mathcal{A}_4. \end{cases} \quad \text{that is} \quad \forall x_1 \cdots \forall x_m [\mathcal{A}_1 \wedge (\mathcal{A}_2 \vee \neg \mathcal{A}_3) \wedge \neg \mathcal{A}_4]$$

where $\mathcal{A}_i$ are positive first–order literals. This class is shown here to be as expressive as Turing machines and all simpler classes would be trivial. The proof is based on a remarkable and not enough known codification of any computable function by unpredictable iterations proposed by [5]. Then, we prove effectively by logical transformations that all conjunctive formulas of Horn clauses can be translated into an equivalent conjuctive 4–formula (as above). Some consequences are presented in several contexts (mathematical logic, unification modulo a set of axioms, compilation techniques and other program patterns).

1 Introduction

This paper is about the computational power of classes of quantificational formulas specified by restrictions on the number of atomic subformulas. Important works have been done about decision problems for such classes. W. Goldfarb and H.R. Lewis in [10] established the undecidability of the class of those formulas containing five atomic formulas as follows

$$\forall x \exists w \forall z_1 \cdots \forall z_m [(\neg \mathcal{A}_1 \vee \mathcal{A}_2 \wedge \mathcal{A}_3) \vee (\neg \mathcal{A}_4 \wedge \mathcal{A}_5)]$$

Indeed, the satisfiability of such a class is equivalent to the halting problem for two–counter machines which is undecidable [16]. H.R. Lewis tried to solve the 4–subformulas case, but without success. This problem remained open until last year and was shown to be undecidable too by two independant ways ([12] and [9]). The main result of this paper is obtained by merging these two proofs and we establish that this undecidability is of the maximal degree, in other words, this class of formulas has, in fact, the same expressive power than Turing machines. Moreover, this class can be reduced to Horn clause formulas :

$$(\mathcal{HC}) \quad \forall x_1 \cdots \forall x_m[(\mathcal{A} \vee \neg\mathcal{A}_1 \vee \ \cdots \ \vee \neg\mathcal{A}_n)] \qquad (\text{denoted } \ \mathcal{A} \leftarrow \mathcal{A}_1 \ \cdots \ \mathcal{A}_n.)$$

Such clauses, useful in logic programming, contain at most one positive subformula (or literal).

Within this particular class, related decision problems have been also studied. For instance, *Implication of Horn clauses*, denoted "$\mathcal{HC}_1 \Rightarrow \mathcal{HC}_2$", also called generalized subsumption, is an important notion in learning theory and compilation. It was shown decidable if clause $\mathcal{HC}_1$ is binary (two–literal) [19] and has been shown recently undecidable if $\mathcal{HC}_1$ is ternary [15].

More precisely, our paper is devoted to establish that all computation on Minsky machines can be expressed as the checking of consistency of the 4–formulas of the following form :

$$\forall x_1 \cdots \forall x_m[\mathcal{A}_1 \wedge (\mathcal{A}_2 \vee \neg\mathcal{A}_3) \wedge \neg\mathcal{A}_4] \qquad \text{that is} \qquad \begin{cases} \mathcal{A}_1 \leftarrow . \\ \mathcal{A}_2 \leftarrow \mathcal{A}_3. \\ \leftarrow \mathcal{A}_4. \end{cases}$$

where, because of the existence of universal Minsky Machines, the three first subformulas, $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ can be fixed in a constructible way and $\mathcal{A}_4$ corresponds to the input datum. This very restrictive class is Turing–complete.

The proof is based on two different techniques. The first part (Section 2) is an encoding of periodically linear functions whose iterations have been proved to be equivalent to Minsky machines by [5]. Standard methods for decision problems and computational power use Minsky or Turing machines. It seems to be surprising that the remarkable result proposed by Conway was rarely used. Its formalism is of higher level and much easier to encode. The second part (Section 3) is based on logical transformations using meta–programs and meta–interpreters.

This theorem is clearly equivalent to the famous Böhm–Jacopini theorem to declarative languages. In 1966, they had proven that within imperative languages, every flowchart is equivalent to a while–program with one occurrence of while–do, provided auxiliary variables are allowed. This proof is constructive and usually cited as the mathematical justification for structured imperative programming (see also [13]). We show that in Horn clause languages, any program can be automatically transformed to another one composed of one binary Horn clause and two unit clauses. This transformation preserves both termination and the answer–substitutions (on the original variables). This shows the expressive power of a single Horn clause and can be used as a theoretical tool for decision problems in theorem proving. Some applications on other notions or structures are presented in the last section.

2 An Original Codification of the Conway Functions

Here we present an original proof method, previously defined in [8, 9] where it was the main basis of the proofs of the undecidability of the halting and emptiness problem for one binary recursive Horn clause. It is based on an original codification of some work by J.H. Conway[5], which we will present briefly here.

2.1 The Conway Unpredictable Iterations

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

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

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

1. $\forall\ n \in \mathbb{N}, n \in Dom(f)$ *iff* $\exists (k, j) \in \mathbb{N}^* \times \mathbb{N},\ g^{(k)}(2^n) = 2^j$.
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.

where $\mathbb{N} = \mathbb{N}^* \cup \{0\}$. This result is based on a direct and clever translation of the behavior of Minsky machines [16] (having the same computational power than Turing machines) into Conway functions. Conway and Guy proposed an instance of such Conway functions which produces all the prime number [11]. A more complex characterization of prime numbers had been done within diophantine equations [16].

In the next proofs, we will need to transform Conway iterations "$n \to g(n)$" to Conway equivalences "$n \leftrightarrow g(n)$" by using Corollary 2. Considering the particular partial recursive functions f such that :

$$\forall\ n \in Dom(f) \supset \{0\},\ f(n) = 0$$

and their associated Conway functions (we called them *null Conway functions*) we can define some *Conway equivalence relations* $\equiv_g$, which mean that we do not only consider positive iterates $g^k(2^n)$ resulting in $2^0 (= 2^{f(n)})$ but also negative iterates $g^{-k}(2^0)$ resulting in 2^n where $g^{-k}(n) = \{m \in \mathbb{N}\ \mid\ g^k(m) = n)$. Thus we establish the following corollary :

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

2.2 Conway Functions and Conjuctive 4–formulas

Let us recall some definitions and notations about Horn clauses. A Horn clause is a disjunction of atomic formulas (called literals) of which at most one is positive.

$$(\mathcal{HC}) \quad \forall x_1 \cdots \forall x_m[(\mathcal{A} \vee \neg\mathcal{A}_1 \vee \ \cdots \ \vee \neg\mathcal{A}_n)] \qquad (\text{denoted } \ \mathcal{A} \leftarrow \mathcal{A}_1 \ \cdots \mathcal{A}_n.)$$

$\mathcal{A}$ is called the head part of the clause, and the other literals the body part. A fact (resp. a goal) is a clause whose body part (resp. the head part) is empty. A Horn clause is said to be binary if it is a composition of only two literals. As an example consider :

$$\begin{cases} append([\,], L, L) \leftarrow . & \textit{fact} \\ append([X \mid L], LL, [X \mid LLL]) \leftarrow append(L, LL, LLL). & \textit{binary clause} \\ \leftarrow append(L, LL, [a, b]). & \textit{goal.} \end{cases}$$

A Horn clause program is a conjunction of Horn clauses universally quantified. The procedural semantics of such programs is based on the SLD–resolution [14] derived from the refutation procedure of Robinson [18]. According to the Edinburgh syntax ([4]), variables (X, L, LL, ...) are written using capital letters and function symbols (a, b, c) with small letters. $[X_1, X_2 \mid L]$ denotes a list of which X_1, X_2 are the two first elements and L the tail of the list. The SLD resolution, possibly infinite, computes all answer–substitutions ($\{L = [\,], LL = [a, b]\}$, $\{L = [a], LL = [b]\}$, $\{L = [a, b], LL = [\,]\}$ in our example).

The variables occurring in Horn clauses are renamed into fresh variables during the resolution. The simplest way to do is to attach an additional index to the variable, denoted by subscripts as in X_i . Since we only consider one binary Horn clause, we can choose this index to correspond to the number of inferences using this clause.

In [8, 9], using this renaming of variables, we have explicitly described how relations of the form $X_n = X_{g(n)}$, where g is any null Conway function, can be expressed by a Horn Clause of the form :

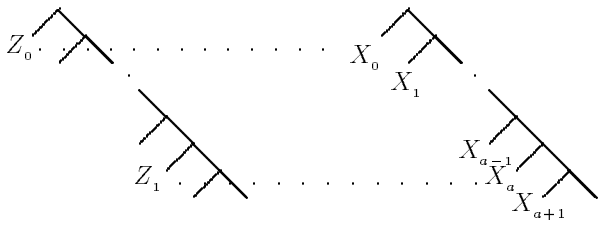
$$p(\textit{left}(X)) \leftarrow \ p(\textit{right}(X)).$$

where *left(X)* and *right(X)* are terms with variable X occurring in it. Thus, considering the case where n is a power of 2, this corresponds to a codification of the Conway equivalence relations into a binary Horn clause ($X_{2^n} = X_{g(2^n)} = X_{2^0} = X_1$).

Indeed, in the following program :

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

the size of the first variable of the Horn clause decreases by a while the size of the second decreases by one, so we have (the trees denote lists) :



If there $b = 0$ 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 :

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

It involves the equalities :

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

It is clear now that as many as needed equalities of the form $X_{ai+b} = X_{a'i+b'}$ can be expressed in one binary Horn clause and one goal.

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 $(a_k d, k a_k) \in \mathbb{N}^2$. Then $(X_n = X_{g(n)})_{n \in \mathbb{N}}$ 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 (as explained above) can be merged in one right–linear binary clause and one goal by merging their arguments. Thus it is clear that for any Conway function, there exist a particular binary Horn clause and a goal.

Consequently, according to the previous corollary, for every recursively enumerable set Σ, containing 0, there exists a binary Horn clause (associated with the corresponding g) which build a list $\mathcal{L} = [X_1, X_2, \cdots, X_n, \cdots]$ with all the X_i linked by relations $X_i = X_{g(i)}$. Thus we can characterize Σ as :

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

Moreover, if at startup X_1 is marked by $\sharp$, then this mark will be propagated to every X_{2^n} such that n belongs to Σ. Thus the clause can be considered as a process of enumeration of the elements of Σ. An element of $\mathcal{L}$ will be instantiated to $\sharp$ iff it is a pure power of 2 and this power belongs to Σ, that is iff $2^n \equiv_g 2^0$.

The following program is an example in the case where Σ is $\mathbb{N}$:

$$p([X|L],[Y,X|LL]) \leftarrow p(L,LL) \ .$$
$$\leftarrow p([\sharp|L],[\sharp|L]) \ .$$

This program puts a $\sharp$ in every n^{th} position of the list $[\#|L]$ iff $n = 2^p$.

If the starting list is $\mathcal{L} = [\sharp, \_, \cdots, \_, \flat, \_, \cdots]$, where $\flat$ is in the $(2^n)^{th}$ position, then the program will stop (since a unification fails) iff the equality $X_1 = X_{2^n}$ occurs, that is iff we have $2^n \equiv_g 2^0$, i.e. iff $n \in \Sigma$. It is undecidable for every n and recursive enumerable set Σ whether n belongs to Σ. Thus, we can prove that the termination, when a goal is given, of one binary Horn clause is undecidable. This result was first presented in [8].

In [9], using the same codification in a different way, it is proved that the satisfiability problem is undecidable too. In this proof, a linear propagation of the mark $\sharp$ is created and the existence of solutions for our minimal program structure is shown to be equivalent to the fact that Σ is not total.

2.3 The Main Lemma

Using the above codification, we establish the following lemma which is required to prove the main theorem of this paper. In the following statement, the term "program" corresponds to the intuitive meaning. You can consider, if you prefer, it denotes "a machine (in sense of Turing machine) which computes a partial recursive function". This lemma needs a sharp analysis of how the codification of Conway functions works. The proof is technical, therefore we will not present it here. It will appear soon in a extended report.

Main Lemma 3. *For every program Π with input I, there exists a binary Horn clause $\mathcal{R}_\Pi$ such that there exists a goal, depending on I, such that $\mathcal{R}_\Pi$ stops after at least n iterative applications iff Π stops after n elementary steps with input I, and does not stop otherwise.*

Proof. To appear soon, it can already be communicated to every interested person.

3 Logical transformations

In the previous section, we have presented how to express Minsky machines into binary Horn clauses. In this part, we show that there exists a constructible reduction from the Horn clause programs into the 1–binary Horn clause programs. This result is obtained by a combination of the proof techniques of [12] and [8, 9]. The first one is used to generate the SLD resolution, the second to assure the halting of the build meta–interpreter.

3.1 A Word Generator

In this section we show how to generate all words over the alphabet $\{a, b\}$. The codification is similar to the one used in [12] to encode the Post Correspondence Problem:

$$gen([W \mid R] - RR\ , W) \leftarrow .$$
$$gen([W \mid R] - [[a \mid W], [b \mid W] \mid RR], AW) \leftarrow gen(R - RR, AW).$$
$$\leftarrow gen([[] \mid R] - R, Word).$$

where $R - RR$ denotes a difference list [3, 6].

By unifying the goal and the fact we obtain the solution $Word = []$. Using the binary clause once, results in the new goal

$$gen([[a], [b] \mid RR^1] - RR^1, AW^1)$$

producing the solution *Word = [a]*. Resolving this new goal with the binary clause instead of the fact results in the goal

$$gen([[b], [aa], [ba] \mid RR^2] - RR^2, AW^2)$$

resulting in the solution *Word = [b]* etc. Observe that a and b serve as prefixes of two new words such that the suffix of these words is the first element of the list generated so far [1]. These two words are concatenated to the tail of the list generated so far. In other words, the difference–list can be seen as a LIFO (Last In First Out) stack.

3.2 A First Meta–Interpreter

Let Π be a set of Horn clauses $\{clause_1, \cdots, clause_n\}$, and "$\leftarrow g_1 \ \cdots g_n$." be a goal, the following meta–program generates the same answer–substitutions in the same order as a standard SLD interpreter :

$$\begin{array}{l} solve([\,],[\,]) \leftarrow . \\ solve([Goal \mid R_1], [[Goal \mid R_2] - R_1 \mid L]) \leftarrow solve(R_2, \mathcal{P}). \\ solve(Goals, [Clause \mid Rest]) \leftarrow solve(Goals, Rest). \\ \leftarrow solve(\mathcal{G}, \mathcal{P}). \end{array}$$

$\mathcal{G}$ denotes list $[g_1, g_2, ...g_n]$ and $\mathcal{P}$ is the list of encoded clauses of a program Π, i.e., $\mathcal{P}$= $[clause_1, \cdots, clause_n]$. A clause $a \leftarrow b_1, \ldots, b_n$ of Π is encoded by the difference list $[a, b_1, \ldots, b_n \mid R] - R$. The first binary clause serves for choosing the first clause in the current clause list for reduction. The second clause discards the first clause in the current clause list. The complexity of the meta–program have been shown to be linearly dependent on that of the original program (see [17]).

For encoding an arbitrary program Π only the first binary and the goal have to be adapted in a appropriate way. Let us suppose now that Π is one of known Horn clause meta–interpreters (i.e. a universal Horn clause program), then this codification defines a constructible meta–interpreter in the form :

$$\begin{cases} \mathcal{A}_1 \leftarrow . \\ \mathcal{A}_2 \leftarrow \mathcal{A}_3. \\ \mathcal{A}_4 \leftarrow \mathcal{A}_5. \\ \leftarrow \mathcal{A}_6. \end{cases} \quad \text{or} \quad \forall x_1 \cdots \forall x_m [\mathcal{A}_1 \wedge (\mathcal{A}_2 \vee \neg \mathcal{A}_3) \wedge (\mathcal{A}_4 \vee \neg \mathcal{A}_5) \wedge \neg \mathcal{A}_6]$$

[1] Note that this generator can be easily extended for any finite alphabet.

3.3 A Binary Meta–Interpreter

In this section we combine the two programs of Section 3.1 and 3.2.

Assume a program Π consisting of the two binary clauses $left_1 \leftarrow right_1$ and $left_2 \leftarrow right_2$, one goal $goal$, and one fact $fact$. Consider the word–generator where $\mathcal{A} = right_1, left_1$ and $\mathcal{B} = right_2, left_2$.

$$\begin{array}{l} meta([W \mid R] - RR\ , W) \leftarrow . \\ meta([W \mid R] - [[\mathcal{A} \mid W], [\mathcal{B} \mid W] \mid RR], [H \mid RRR]) \\ \qquad\qquad\qquad\qquad \leftarrow meta(R - RR, [H, X, X \mid RRR]). \\ \leftarrow meta([[goal \mid L\ ] \mid R] - R, [fact \mid LL\ ]). \end{array}$$

After n times using the binary clause for resolving we obtain as second argument of the new goal the list

$$[fact, X_1, X_1, X_2, X_2, \ldots, X_n, X_n \mid T\ ].$$

Furthermore, we obtain after some iterations as the head of the first argument (cf. Section 3.1)

$$[right_{i_m}, left_{i_m}, \ldots, right_{i_1}, left_{i_1}, goal \mid T\ ].$$

By resolving a current meta–goal with the fact of the meta–program above we obtain the unification problem

$$right_{i_m} \doteq fact, left_{i_m} \doteq right_{i_{m-1}}, \ldots, left_{i_1} \doteq goal.$$

Note that by construction it is assured that the list containing the fact is of sufficient length to obtain these equations. This set is solvable if and only if there exists a corresponding refutation of the original program Π using the resolution order imposed by the equations.

Since the meta–interpreter of Section 3.2 is such a program Π, it is possible to associate to any logic program an equivalent program (i.e., with the same solutions) containing a binary clause and two unit clauses. Unfortunately, this codification does not preserve termination. In the next section we will show how to construct from this non–terminating interpreter a terminating one.

Let *solve* be the first meta–interpreter (Section 3.2) and all SLD derivation whose length is n w.r.t. *solve* is evaluated w.r.t. its meta–program before at most 2^n resolution steps. In other words, all answer–substitution computed after n resolution steps w.r.t. *solve* will be computed before 2^n resolution steps w.r.t. its meta–program.

3.4 A Technical Preliminary

MP_{nS} denotes the non–stopping meta–interpreter defined in Section 3.2. Let us consider the following program Π, with input a Horn clause program P :

1. input P
2. evaluate P by a breadth–first strategy and keep the solutions in $\mathcal{S}_1$

3. compute $MP_{nS}(P)$ and keep the solutions in $\mathcal{S}_2$, stop as soon as $\mathcal{S}_2 = \mathcal{S}_1$ and write 0

It is clear that Π stops iff P stops and its stopping time (that is the number of steps before termination) with input P is greater than the time used by $MP_{nS}(P)$ for producing all the solutions of P.

Now, according to the main lemma, there exists a binary clause $\mathcal{R}_\Pi$ and then a program $P_\Pi = MP_S$:

$$MP_S \; : \begin{cases} Stop(fact_1) \leftarrow . \\ Stop(left_1) \leftarrow Stop(right_1). \\ \leftarrow Stop(goal_1). \end{cases}$$

which, with input (in the goal) a program P[2], stops iff P stops and its stopping time is greater than the one of $MP_{nS}(P)$. Observe that P is used only in the goal.

3.5 The smallest Meta–Interpreter

Using MP_{nS} and MP_S we prove the following theorem :

Theorem 4. *There exists a meta–interpreter for Horn clauses in the form of a program with only one binary Horn clause, a fact and goal, which, given as input a Horn clause program P, has exactly the same solutions as P and terminates iff P terminates.*

Proof. In this proof, we use the meta program MP_{nS} defined previously, we denote its program :

$$MP_{nS} \; : \begin{cases} meta(fact_2) \leftarrow . \\ meta(left_2) \leftarrow meta(right_2). \\ \leftarrow meta(goal_2) \end{cases}$$

We merge MP_{nS} and MP_S in a new meta–interpreter MP :

$$MP \; : \begin{cases} TheMeta(fact_1, fact_2) \leftarrow . \\ TheMeta(left_1, left_2) \leftarrow TheMeta(right_1, right_2). \\ \leftarrow TheMeta(goal_1, goal_2). \end{cases}$$

such that, with input a Horn clause program P, it produces all the solutions of P (because of the MP_{nS} part) and then will stop iff P terminates (because of the MP_S part).

Thus we have an *append*–like meta–program which preserves the solutions (produced in the same order as in a breadth–first strategy) and the termination of any Horn clause program given as input. □

An immediate consequence of this theorem is :

[2] Based on the Gödel number of P.

Theorem 5. *The class of programs with only one binary Horn clause and two unit clauses is Turing–complete.*

Proof. Since we have a meta–interpreter for Horn clauses containing only one binary recursive clause, we can assert that this class of programs has the same computational power as Horn clause programs and consequently as Turing machines. □

This meta–interpreter is effectively the simplest constructible one. Indeed, the consistency of the formulas with three sub–formulas is clearly decidable.

Let us remark that there is therefore a universal binary Horn clause+one unit clause which are explicitly constructible. This theorem still remains true if the binary clause and one of the two unit clauses are fixed. In the same manner, unification with or without occur–check, SLD strategies, bottom–up or top–down resolution algorithms do not alter anything.

4 Conclusion

The first consequence of this theorem is that, as Lewis conjuctered in 1973, the consistency of the class of formulas containing 4–subformulas is undecidable too, even for conjuctive formulas of Horn clauses with prefixes of the form $\forall \cdots \forall$.

The second consequence is that like in imperative languages [1], the simplest non–trivial program scheme can express any partial recursive function. Like in the Böhm–Jacopini proof, the transformation can be done automatically. A meta–interpretation of the academic *append* predicate has been implemented and its space complexity was so important that it can simulate only the four first original inferences under Sictus 2.1 on SUN machine with 12MB RAM.

A third way to express this theorem is in unification modulo a set of axioms. The usual unification that is, modulo the set of equality axioms, is unary (there is one most general unifier) and decidable. As soon as a single arbitrary axiom is added, the unification becomes infinitary and undecidable (see for instance the remarkable encoding proposed by [7]). This remains true even if the terms used have a special symbol (predicate name) which appears at their root and nowhere else, and even if this additional axiom is linear and without local variables, that is, all variables occurs once and only once in every term of the axiom.

The negative result justifies pragmatic or heuristics approach to logic programming analysis, like in abstract interpretation or type inference. There is no hope to define any formal and complete methods to control the most basic recursive scheme. Even in such restrictive classes of programs, most of the interesting properties (groudness, sharing, freeness, necessity of occur–check, ...) to provide more efficient compilation technics are undecidable. For instance the property of total decoration can deal to efficient translation from logic programming to attribute grammars [2]. Roughly speaking, a SLD resolution is said to be totally decorated iff at each step of resolution all Horn clauses can be used to infer. The open problem concerning the total–decoration property can be proved easily to be undecidable. Indeed for logic programs which contain only one binary clause,

this property is equivalent to the non–halting problem that we have shown to be undecidable.

Another program scheme studied by [15] for implication of clause is the following one :

$$\begin{cases} \mathcal{G}_1 \leftarrow . \\ \cdots \cdots \\ \mathcal{G}_n \leftarrow . \\ \mathcal{A} \leftarrow \mathcal{A}_1, \mathcal{A}_2 . \\ \leftarrow \mathcal{G}_{n+1} . \end{cases}$$

where all $\mathcal{G}_i$ are ground (without variables) and the Horn clause is now ternary.

They proved, using a sophisticated technique based on semi–Thue systems and SLD resolution that the consistency for such a class is undecidable. Here the halting problem of the SLD resolution can be shown to be undecidable too and additional restrictions can be added like n $\leq$ 2 and the ternary Horn clause is explicitly chosen.

Acknowledgements : We would like to thank Anne Parrain for her helpful collaboration. She contributes a lot to the work on meta–program. We would like to thank Philipp Hanschke for valuable discussions. Last, we would like to thank anonymous referees for their valuable comments.

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. Bouquard J.L. "Logic programming and Attribute grammars." *Ph.D. Thesis, Orléans* 1992.
3. Bratko I. "*Prolog Programming for Artificial Intelligence*". Addison–Wesley Publishers limited. 1986.
4. William F. Clocksin and Christopher S. Mellish, *"Programming in Prolog". Springer–Verlag.* 1981.
5. Conway J.H. "Unpredictable Iterations." *Proc. 1972 Number Theory Conference. University of Colorado, pp 49–52.* 1972.
6. Clark K.L., Tärnlund S.A. "A First Order Theory of Data and Programs." *in Proc. IFIP 77. pp. 939–944.* 1977.
7. Dauchet M. "Simulation of Turing Machines by a regular rewrite rule." *Journal of Theoretical Computer Science. n°103. pp. 409–420.* 1992.
8. 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. February 1993.
9. Devienne P., Lebègue P., Routier J.C. "The Emptiness Problem of One Binary Recursive Horn Clause is Undecidable." *In proceedings of ILPS'93, Vancouver. MIT Press. pp 250–265.* October 1993.
10. Goldfarb W. and Lewis H.R. "The decision problem for formulas with a small number of atomic subformulas" *J. Symbolic Logic 38(3)*, pp.471–480, 1973.

11. Guy R.K. " Conway's Prime Producing Machine." *Mathematics Magazine. n°56. pp. 26–33.* 1983.
12. Hanschke P., Würtz J. "Satisfiability of the Smallest Binary Program." *Information Processing Letters, vol. 45, n°5. pp. 237–241.* April 1993.
13. Harel D. "On folk theorems" *CACM, vol. 23, n°7. pp. 379–389.* 1980.
14. Kowalski R. A. *"Logic for Problem Solving." North Holland. New York.* 1979.
15. Marcinkowski J., Leszek Pacholski "Undecidability of the Horn–Clause Implication Problem" *Proc. of the 33rd FOCS. 1992.*
16. Minsky M. *"Computation : Finite and Infinite Machines."* Prentice–Hall. 1967.
17. Parrain A., Devienne P., Lebègue P. "Prolog programs transformations and Meta–Interpreters." *Logic program synthesis and transformation, Springer–Verlag, LOPSTR'91, Manchester.* 1991.
18. Robinson J. A. "A Machine–oriented Logic Based on the Resolution Principle." *J. ACM n°12, pp. 23–45.* Januar 1965.
19. Schmidt–Schauss M. "Implication of clauses is undecidable." *Journal of Theoretical Computer Science, n°59, pp. 287–296.* 1988.