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.

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

* This work has been partially supported by GRECO de Programmation of CNRS.
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 | L], LL,[X | 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.

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

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

\[
... \leftarrow_i \rightarrow_i \leftarrow_2 \rightarrow_2 \leftarrow \rightarrow_n \leftarrow_{n-1} \rightarrow_{n-1} \leftarrow_n \rightarrow_n ...
\]