**Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH**

**Research
Report**

RR-93-09

Satisfiability of the Smallest Binary Program

Philipp Hanschke, Jörg Würtz

February 1993

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
Fax: + 49 (631) 205-3210

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: + 49 (681) 302-5252
Fax: + 49 (681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, Sema Group, and Siemens. Research projects conducted at the DFKI are funded by the German Ministry for Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct systems with technical knowledge and common sense which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

Intelligent Engineering Systems
Intelligent User Interfaces
Computer Linguistics
Programming Systems
Deduction and Multiagent Systems
Document Analysis and Office Automation.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Dr. Dr. D. Ruland
Director

Satisfiability of the Smallest Binary Program

Philipp Hanschke, Jörg Würtz

DFKI-RR-93-09

Appears also in: Information Processing Letters

This work has been supported ba a grant from The Federal Ministry for Research and Technology (FKZ ITWM-8902 C4 and ITW 91050).

ISSN 0946-008X

Satisfiability of the Smallest Binary Program

Philipp Hanschke[*], Jörg Würtz[†]

Abstract

Recursivity is well known to be a crucial and important concept in programming theory. The simplest scheme of recursion in the context of logic programming is the binary Horn clause $P(l_1, \ldots, l_n) \leftarrow P(r_1, \ldots, r_n)$. The decidability of the satisfiability problem of programs consisting of such a rule, a fact and a goal – called *smallest binary program* – has been a goal of research for some time. In this paper the undecidability of the smallest binary program is shown by a simple reduction of the Post Correspondence Problem.

Keywords: Theory of computation

Contents

1 Introduction

Recursivity is well known to be a crucial and important concept in programming theory. For imperative languages, Böhm and Jacopini [Boe66] proved that all programming can be done with at most one while loop. In the context of logic programming a simple scheme of recursion is the binary Horn clause[1]

$$P(l_1, \ldots, l_n) \leftarrow P(r_1, \ldots, r_n)$$

such that l_1, r_1, ..., l_n, r_n are finite terms. But is such a scheme really simple?

It was shown in [Aan71] and in [Boe71] that every computable function can be computed by a program consisting of binary Horn clauses and a number of facts (refined in [Lew76] where only a single predicate letter is used). Hence, general binary programs were shown to be undecidable. Furthermore, the problem whether a clause set consisting of two binary clauses and two ground unit clauses is satisfiable was proved to be undecidable by M. Schmidt-Schauß [SS88]. In this paper we consider the class of programs consisting of a goal, a binary rule and a fact, i.e.,

$$\begin{array}{lcl} & \leftarrow & P(g_1, \ldots, g_n) \\ P(l_1, \ldots, l_n) & \leftarrow & P(r_1, \ldots, r_n) \\ P(f_1, \ldots, f_n) & \leftarrow & \end{array}$$

where all arguments represent finite terms. This is the smallest binary program.[2] The satisfiability of such a clause set is decidable if the fact and the goal are ground [SS88], i.e., they do not contain variable occurrences. Intuitively, since the goal as well as the fact are ground, they define upper bounds on the depth of the terms occurring in the binary rule. After a finite sequence of self-application steps, any further self-application either leaves the rule invariant or increases the depth of the terms occurring in it. Independently, P. Devienne [Dev90] has given a more general result for programs with linear goals and facts, i.e., each variable occurs at most once in the goal and the fact. He uses essentially the same ideas as Schmidt–Schauß, but a specialized technique based on directed weighted graphs. Both do not impose any further restriction on the binary clause. On the other hand the problem whether there exists an answer substitution for the program consisting of arbitrary unit clauses and a binary clause $L \leftarrow R$ such that L and R are unifiable is also decidable [Wür92]. Therein, so called dependency graphs are used to predict the behavior of the binary clause if it is applied to itself. But the decidability of the *satisfiability problem of the smallest binary program* is still open in the general case:

[1] A Horn clause is a disjunction of literals with at most one positive literal. A fact and a goal are Horn clauses consisting of one positive and one negative literal, respectively. We assume the predicate symbol of the head and body literal of a binary clause to be equal to enable recursion.

[2] A program is called binary iff it contains at least one fact, one goal and one binary Horn clause. All further Horn clauses must be facts or binary.

Is there an answer substitution σ such that $\sigma P(g_1, \ldots, g_n)$ is a logical consequence of the binary clause $P(l_1, \ldots, l_n) \leftarrow P(r_1, \ldots, r_n)$ and the fact $P(f_1, \ldots, f_n)$?

In this paper we show the undecidability of the problem and are even able to sharpen this result.

So far the theoretical aspects. Binary Horn clauses are quite common in automated theorem proving and logic programming. It is well known that clauses allowing self-application are the source of non-terminating queries [SGG86]. As a first step research in the special case of binary clauses was started. Much effort was devoted to control the self-applicability of binary clauses in order to detect non-terminating queries and to speed up the computation in terminating cases; cf. for instance [Dev90], [DVB90] or [UvG88].

2 Reduction

For basic notions such as substitution, unification, SLD-resolution from logic programming we refer to [Llo87]. The undecidability of the satisfiability of the smallest binary program is shown by reducing the Post Correspondence Problem to it.

First, we recall the definition of the Post Correspondence Problem. Let Σ be a finite alphabet. A *Post Correspondence System* (PCS) over Σ is a nonempty finite set $S = \{(l_i, r_i); i = 1, ..., m\}$ where the l_i, r_i are words over Σ. A nonempty sequence of indices $1 \leq i_1, ..., i_n \leq m$ is called a *solution* of the system S iff $l_{i_1} \cdots l_{i_m} = r_{i_1} \cdots r_{i_m}$. It is well-known that the *Post Correspondence Problem*, i.e., the question whether there exists a solution for a given system, is in general undecidable if the alphabet contains at least two symbols [Pos46].

Elements of the alphabet Σ will be represented as unary function symbols and a word $w = a_1 \cdots a_n$ over Σ thus becomes a term $a_1(a_2(\ldots(a_n(\epsilon))\ldots))$ where ϵ is a constant corresponding to the empty word. So, composition of words is associative since composition of functions is associative. For convenience we also write $w(\epsilon)$ and $u(v(\epsilon)) = uv(\epsilon)$ where u and v correspond to words over Σ. For instance, if $w_1 = ab$, $w_2 = ba$, $v_1 = a$, and $v_2 = bba$, then $w_1(w_2(t)) = a(b(b(a(t)))) = abba(t) = v_1v_2(t)$ for any term t.

In our reduction we use the dot "." as a binary, right-associative infix operator. I.e., for terms r, s, and t we consider $r.s.t$ as an abbreviation for $r.(s.t)$. A term $r_1.\cdots.r_n$ is called a *list*. In a term $r.s$ we refer to r as the *car* and to s as the *cdr*. To append something to a list using unification we use the concept of *difference lists*. A difference list is a pair of a list with the last *cdr* being a variable X and the variable X. Now consider the unification of the difference list $(1.2.3.X, X)$ with the pair $(H.R, 4.5.6.Z)$. Obviously, $\sigma = \{X \mapsto 4.5.6.Z,\ H \mapsto 1,\ R \mapsto 2.3.4.5.6.Z\}$ is a most general unifier and $\sigma(R, Z) = (\sigma R, \sigma Z) = (2.3.4.5.6.Z, Z)$ is again a difference list.

To explain the encoding of a PCS we adopt SLD resolution as an operational semantics for the logic program. The search space of possible sequences of indices inherent to a PCS is not encoded in the and/or tree of the logic program. Instead we encode it in two (difference) lists L and R. At the beginning of the computation L and R is $l_1(\epsilon).\cdots.l_m(\epsilon).X, X$ and $r_1(\epsilon).\cdots.r_m(\epsilon).Y, Y$, respectively. This encodes all possible sequences of indices of length 1 (i.e., $1, 2, \ldots, m$). In the next step we select the sequence 1 and replace it by all sequences that have length 2 and as suffix 1. In terms of the lists L and R: we remove $l_1(\epsilon)$ and $r_1(\epsilon)$ (representing the sequence 1 of length 1) and append $l_1(l_1).\cdots.l_m(l_1).X$ and $r_1(r_1).\cdots.r_m(r_1).Y$, respectively (representing the sequences 11, 12, ..., $1m$ of length 2).

In the general case we select in each step a sequence $i_1 \ldots i_j$ of indices and replace it by all sequences that have length $j+1$ and $i_1 \ldots i_j$ as suffix. Always selecting the *car*s of L and R and appending the extensions is a fair strategy. I.e., it ensures that successively all possible sequences appear as *car*s of the two difference lists.

Given a PCS as above, the following binary program has a SLD refutation, iff the PCS has a solution.

$$
\begin{array}{lcl}
 & \leftarrow & P(l_1(\epsilon).\cdots.l_m(\epsilon).X,\ X, \\
 & & \quad r_1(\epsilon).\cdots.r_m(\epsilon).Y,\ Y) \\
P(C.L,\ l_1(C).\cdots.l_m(C).X, & & \\
\quad D.R,\ r_1(D).\cdots.r_m(D).Y) & \leftarrow & P(L,\ X,\ R,\ Y) \\
P(E.H_1,\ H_2,\ E.H_3,\ H_4) & \leftarrow &
\end{array}
$$

Note that L, X and R, Y form difference lists, respectively. The fact checks whether the *car*s of the current goal are equal, i.e., encode a solution of the PCS. In Figure 1 the sequence of goals is depicted that is induced by a SLD resolution with a search rule always taking the binary rule for the next SLD resolution step.

As SLD resolution is sound and complete we have the following theorem.

Theorem *The satisfiability of the smallest binary program is undecidable.*

Observe that we can generalize the theorem by restricting the class of binary programs to those with a right-linear rule, i.e., the right-hand side is linear. Since a PCS has infinitely many solutions if it has one solution, we can conclude from our result that it is undecidable whether the smallest binary program has infinitely many answers.

3 Related Work

In this paper we consider the question whether there exists a solution at all. In [DLR93] the authors show that it is undecidable whether or not there exists a finite number of answer substitutions for the smallest binary program. They also show that it is undecidable whether the resolution process of a goal and a

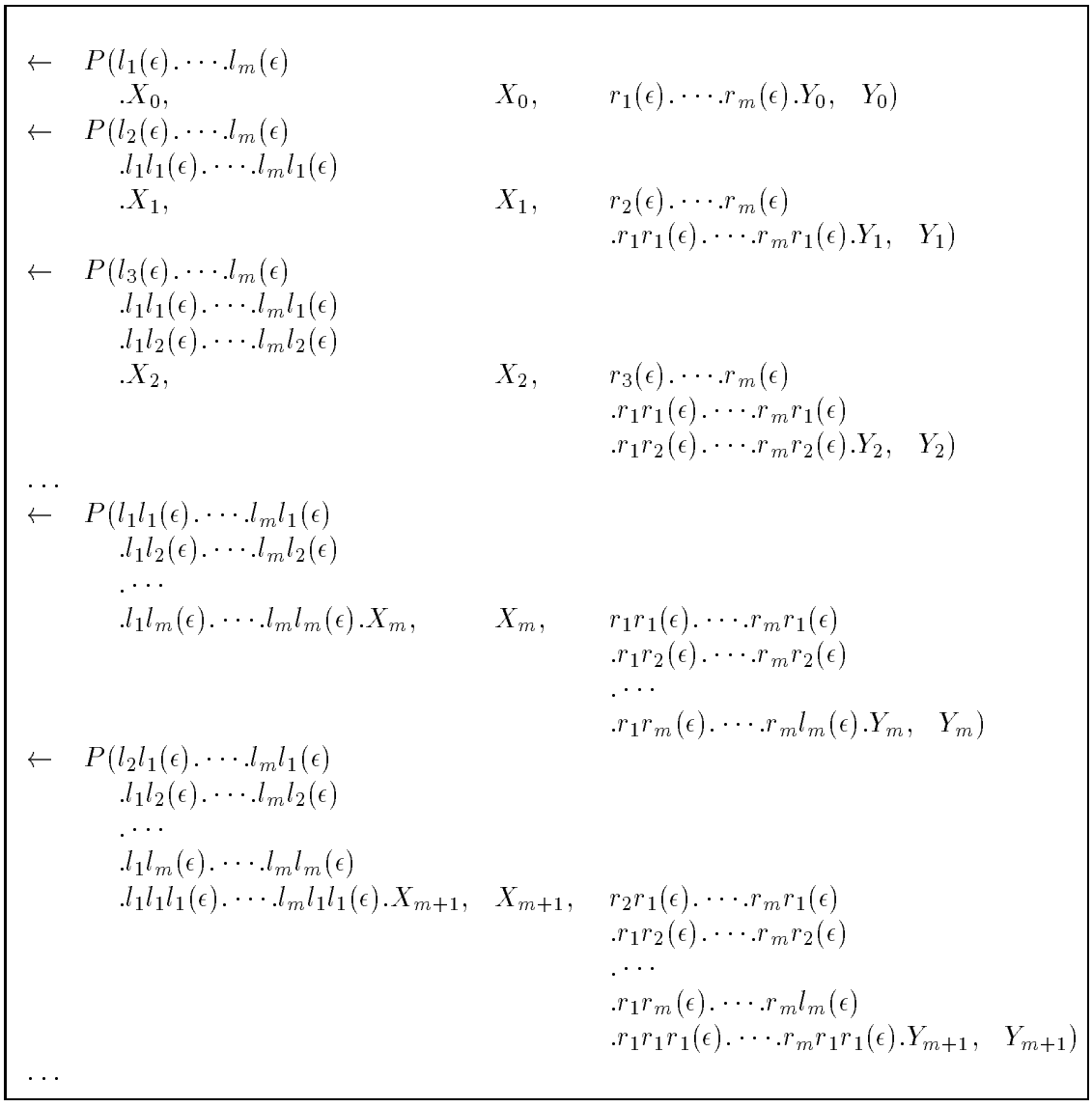

Figure 1: A goal sequence induced by the logic program

right-linear Horn clause stops. As mentioned above, we can conclude from our result that it is undecidable whether the smallest (right-linear) binary program has infinitely many answers.

The solution of the implication $A \Rightarrow B$ of two clauses A and B is usually interpreted as the formula $(\forall x_1, \ldots, x_n A) \Rightarrow (\forall y_1, \ldots, y_m B)$ where $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_m\}$ are the variables occurring in A and B, respectively. Clause implication is equivalent to the nonsatisfiability problem of a clause set consisting of clause A and ground unit clauses obtained by negating of clause B. Hence, the result of M. Schmidt-Schauß cited in Section 1 is an implication of his result that clause implication $A \Rightarrow B$ is decidable in case of A being a binary clause.

If A contains four or more literals, the problem is undecidable. Marcinkowski and Pacholski [Mar92] have recently shown that clause implication is undecidable in case of $L_1 \leftarrow R_{11}, R_{12} \Longrightarrow L_2 \leftarrow R_{21}, \ldots, R_{2k}$.

The subject of [BHW92] and [Wür92] is cycle unification. Therein, a binary clause which can be applied to itself is called a cycle. In order to be able to control a cycle efficiently the authors ask for the existence of an algorithm which enumerates a minimal and complete set of solutions for a cycle unification problem. The answer to this question may significantly increase the power of automated theorem provers. The most general result concerns the class where the left-hand side and the right-hand side of the cycle are unifiable. For this class an upper limit of necessary self-applications of the cycle can be computed such that all further self-applications lead to variants of previously computed solutions. Whereas the problems in [BHW92] and [Wür92] only allow finitely many different solutions, G. Salzer [Sal92] has proved a class of cycle unification problems to be decidable which allows for infinitely many different solutions (he uses so called R-terms as a means to represent infinite sets of first order terms finitely).

Acknowledgment: We would like to thank Manfred Schmidt-Schauß and Gernot Salzer for encouraging this work and for valuable comments.

References

[Aan71] S.O. Aanderaa. On the decision problem for formulas in which all disjunctions are binary. *Proceedings of the 2nd Scandinavian Logic Symposium*, 1–18, 1971.

[BHW92] W. Bibel, S. Hölldobler, and J. Würtz. Cycle unification. In D. Kapur, editor, *Proceedings of the Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 94–108. Springer, June 1992.

[Boe66] C. Böhm and G. Jacopini. Flow diagrams, Turing machines and languages with only two formation rules. *Communication of the ACM*, 9:366–371, 1966.

[Boe71] E. Börger. Reduktionstypen in Krom- und Hornformeln. *Dissertation*, Universität Münster, 1971.

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

[DLR93] P. Devienne, P. Lebegue, and J.-C. Routier. Halting problem of one binary Horn clause is undecidable. *STACS*, To appear.

[DVB90] D. DeSchreye, K. Verschaetse, and M. Bruynooghe. A practical technique for detecting non–terminating queries for a restricted class of Horn clauses, using directed, weighted graphs. In *Proceedings of the International Conference on Logic Programming*, pages 649–663, 1990.

[Lew76] H.R. Lewis. Krom formulas with one dyadic predicate letter. *Journal of Symbolic Logic*, 46(2):341–362, 1976.

[Llo87] J.W. Lloyd. *Foundations of Logic Programming.* Springer, 1987

[Mar92] J. Marcinkowski and L. Pacholski. Undecidability of the Horn-clause implication problem. FOCS, 1992.

[Pos46] E. M. Post. A variant of a recursively unsolvable problem. *Bull. Am. Math. Soc.*, 52:264–268, 1946.

[Sal92] G. Salzer. Solvable classes of cycle unification problems. *IMYCS, Smolenice (CSFR)*, 1992.

[SGG86] D.E. Smith, M.R. Genesereth, and M.L. Ginsberg. Controlling recursive inference. *Artificial Intelligence*, 30:343–389, 1986.

[SS88] M. Schmidt-Schauß. Implication of clauses is undecidable. *Journal of Theoretical Computer Science*, 59:287–296, 1988.

[UvG88] J.D. Ullman and A. van Gelder. Efficient tests for top–down termination of logical rules. *Journal of the ACM*, 35(2):345–373, 1988.

[Wür92] J. Würtz. Unifying cycles. In B. Neumann, editor, *Proceedings of the European Conference on Artificial Intelligence*, pages 60–64. John Wiley & Sons, August 1992.