# A LOGICAL RECONSTRUCTION OF PROLOG II

M.H. van Emden* and J.W. Lloyd

*University of Waterloo
University of Melbourne

CS-83-35
November, 1983

# A Logical Reconstruction of Prolog II

*M.H. van Emden*[†] *and J.W. Lloyd*[‡]

[†]University of Waterloo
[‡]University of Melbourne

*ABSTRACT*

Colmerauer has proposed a theoretical model for Prolog II based on tree rewriting rather than logic. In this paper, we show that Prolog II can be regarded as a logic programming language.

## 1. Introduction

We take the view that a logic programming language is one in which a program is a first-order theory and computed answers are correct with respect to this theory [1,8].

One can then pose the question: is Prolog II [2,3] a logic programming language and, if so, in what sense is it? This question naturally arises from Colmerauer's account of his theoretical model for Prolog II. There, all explicit connection with first order logic has been severed. Instead, Prolog II is regarded as a system which manipulates infinite trees. Unification is replaced by transformations on sets of equations. Roughly speaking, Prolog II is standard Prolog without the "occur check". Since it is well known that the lack of an occur check in Prolog can lead to incorrect answers, it is not immediately obvious that Prolog II can be thought of as a logic programming language.

We show that the answer to our question lies in making explicit Prolog II's theory of equality. Once that is done, it is easy to demonstrate that answers computed by Prolog II are correct with respect to a first-order theory consisting of (essentially) the program plus the equality theory.

Section 2 contains a brief account of Prolog II. In section 3, we introduce the idea of the"general procedure", which is an SLD-resolution proof procedure underlying both Prolog and Prolog II. In section 4 we show that Prolog is essentially the general procedure plus the equality theory $\{x = x\}$.[§] In section 5 Prolog II is shown to be essentially the general procedure plus a rather more complicated equality theory. What distinguishes Prolog from Prolog II then is the different way they handle equality. Section 6 contains some concluding remarks.

Throughout, $P$ denotes a Horn-clause logic program not containing the predicate "=". Similarly, $G$ will always denote a goal which does not contain the predicate "=".

## 2. Prolog II

The following brief description of Prolog II is taken from [2].

**Definition** An *equation* is an expression of the form $t_1 = t_2$ where $t_1$ and $t_2$ are terms.

**Definition** A set of equations is in *substitution* form if it is $\{x_1 = t_1, \ldots, x_n = t_n\}$, where $x_1, \ldots, x_n$ are distinct variables and none of $t_1, \ldots, t_n$ is a variable.

§ The meaning of "Prolog" here excludes any form of negation.

**Definition** A set $\{x_1 = t_1, \ldots, x_n = t_n\}$ of equations in *substitution* form *has a loop* if for some $k = 1,\ldots,n$, $t_k$ has an occurrence of $x_k$ or if such an occurrence of $x_k$ can appear after possibly repeated substitutions in $t_k$ using equations of the set.

In Prolog II, the solution of a set of equations is a substitution of trees for variables that makes both sides of each equation the same tree. A set of equations in substitution form is obviously solvable over the domain of rational trees. A set of equations in substitution form without a loop is obviously solvable over the domain of finite trees. Thus, equations can be solved by reducing them to substitution form by applying solution-preserving transformations.

Colmerauer lists the following transformations [2]:

Compaction:

Eliminate any equation of the form $x = x$.

Variable Anteposition

If $x$ is a variable and $t$ is not a variable, then replace $t = x$ by $x = t$.

Splitting

Replace $f(s_1,\ldots,s_n) = f(t_1,\ldots,t_n)$ by $s_1 = t_1, \ldots, s_n = t_n$.

Confrontation

If $x$ is a variable and $t_1$, $t_2$ are not variables and the size of $t_1$ is not greater than the size of $t_2$, then replace $x = t_1$, $x = t_2$ by $x = t_1$, $t_1 = t_2$.

Variable Elimination

If $x$ and $y$ are distinct variables, $x = y$ is in the system and $x$ has other occurrences in that system, then replace these other occurrences of $x$ by $y$.

He asserts that for any finite set of equations, application of the transformations in any order is only possible a finite number of times. Then either a set is obtained which is in substitution form or the set contains an equation of the form $t_1 = t_2$ where $t_1$ and $t_2$ have different outermost functions symbols. In the latter case the set has no solution over the domain of rational trees.

In Prolog II the clauses of a program are regarded as rules for rewriting a tree to a possibly empty sequence of trees. A query consists of a sequence of trees and a set of equations. A query is rewritten to another according to

$$< [A_1, \ldots, A_{i-1}, A_i, A_{i+1}, \ldots, A_n], E > \rightarrow$$

$$< [A_1, \ldots, A_{i-1}, B_1, \ldots, B_m, A_{i+1}, \ldots, A_n], E' >$$

if there is a rule $B \rightarrow B_1, \ldots, B_m$ $(m \geq 0)$ in the program, if $E \cup \{B = A_i\}$ can be transformed to substitution form and if $E'$ is such a form.

The final query in a derivation has an empty sequence of trees. The corresponding set of equations is the answer.

Now that we have given a brief overview of Prolog II, we are in a position to explain in what sense it is possible to give a logical reconstruction of Prolog II.

The domain of interest for Prolog II is the set of infinite trees. What we have to do is find a first-order theory for which the intended interpretation is a model and also for which every answer computed by Prolog II is correct with respect to this theory. Naturally, the main part of this theory is the program itself. The remainder is simply a theory of equality. We have to find an equality theory so that each of the transformations employed by Prolog II (compaction, etc.) can be justified because they always produce a set of equations that is a logical consequence of the parent set of equations plus the equality theory.

## 3. The General Procedure

**Definition** The *homogeneous form* of a clause $p(t_1, \ldots, t_n) \leftarrow B_1, \ldots, B_m$ is

$$p(x_1, \ldots, x_n) \leftarrow x_1 = t_1, \ldots, x_n = t_n, B_1, \ldots, B_m$$

where $x_1, \ldots, x_n$ are distinct variables not appearing in the original clause.

**Definition** Let $P$ be a program. The *homogeneous form* $P'$ of $P$ is the collection of homogeneous forms of each of its clauses.

**Definition** An atomic formula, whose predicate symbol is "=", is called an *equation*.

We now describe the *general procedure*. We call it "general" because, depending on the theory of equality invoked after it, we get Prolog, Prolog II or other specialized languages.

The general procedure uses the homogeneous form $P'$ of a program $P$ and produces an SLD-derivation [7,4]. It consists of constructing, from some initial goal $G$, an SLD-derivation using input clauses from $P'$, while never selecting an equation. The general procedure terminates if a goal consisting solely of equations is reached. Note that because of the homogeneous form of $P'$ the general procedure never constructs bindings for the variables in the initial goal.

For a particular language, the general procedure needs to be supplemented by a theory $E$ of equality. $E$ is used to prove the equations resulting from the general procedure. During the proving of the equations, substitutions for the variables in the initial goal are produced. If the equation-solving process is successful (that is, the empty goal is eventually produced), then these substitutions for the variables in the initial goal are output as the answer.

The equation-solving process would normally be done by resolving goal clauses with clauses from the equality theory. However, other methods are possible. For example, the last step in the equation solving process for Prolog II is not a resolution step.

The introduction of the general procedure is purely a didactic device to explain which parts of Prolog and Prolog II are the same. Obviously, it would be very inefficient in practice since unsolvability of a set of equations is not detected until near the end of a computation. A practical system must perform some equation solving throughout a computation and, of course, this is what both Prolog and Prolog II do.

## 4. Equality theory for Prolog

**Proposition 1.** Let $P$ be a program, $G$ a goal and $P'$ the homogeneous form of $P$. Then $P \cup \{G\}$ is unsatisfiable iff $P' \cup \{x = x\} \cup \{G\}$ is unsatisfiable.

**Proof** We first prove that $P$ is a logical consequence of $P' \cup \{x = x\}$. Let $M$ be a model for $P' \cup \{x = x\}$. We have to show $M$ is a model for $P$. Take in $P$ any clause $p(t_1, \ldots, t_n) \leftarrow B_1, \ldots, B_m$ with variables $y_1, \ldots, y_d$. Suppose that for some assignment of these variables $B_1 \wedge \cdots \wedge B_m$ is true in $M$. Consider the homogeneous form

$$p(x_1, \ldots, x_n) \leftarrow x_1 = t_1, \ldots, x_n = t_n, B_1, \ldots, B_m$$

of this clause in $P'$. Let $x_i$ be the element assigned to $t_i$ for the above assignments of the $y_j$'s, for $i = 1, \ldots, n$. By the axiom $x = x$ and the assumption that $B_1 \wedge \cdots \wedge B_m$ is true in $M$, we have that $p(x_1, \ldots, x_n)$ is true in $M$. That is, $p(t_1, \ldots, t_n)$ is true in $M$. Consequently, $M$ is a model for $P$ and so $P$ is a logical consequence of $P' \cup \{x = x\}$.

It follows from this that if $P \cup \{G\}$ is unsatisfiable, then so is $P' \cup \{x = x\} \cup \{G\}$.

Conversely, suppose $P' \cup \{x = x\} \cup \{G\}$ is unsatisfiable. Let $M$ be a model for $P$. Then we can extend $M$ to a model $M'$ for $P' \cup \{x = x\}$ by assigning the identity relation to "=". Thus $G$ is false in $M'$ and hence in $M$. Hence $P \cup \{G\}$ is unsatisfiable. □

Proposition 1 shows that the equality theory for Prolog is the single axiom $\forall\, x\ x = x$.

**5. Equality theory for Prolog II**

The equality theory $E$ for Prolog II is rather more complex than the one for Prolog and consists of the following axioms:

1. $\forall x\ x = x$
2. $\forall x \forall y\ \ x = y \rightarrow y = x$
3. $\forall x \forall y \forall z\ x = y \wedge y = z \rightarrow x = z$
4. $\forall x_1 \cdots \forall x_n \forall y_1 \cdots \forall y_n\ (x_1 = y_1) \wedge \cdots \wedge (x_n = y_n)$
   $\rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$,
   for all function symbols $f$.
5. $\exists\, x_1 \cdots \exists x_n\ \exists\, y_1 \cdots \exists\, y_k\ \ (x_1 = t_1) \wedge \cdots \wedge (x_n = t_n)$,
   where the $x_i$'s are distinct variables, the $t_i$'s are terms and $\{x_1, \ldots, x_n, y_1, \ldots, y_k\}$ is the set of all variables in the formula.

Note that axioms 4 and 5 are actually axiom schemas. The first task is to show that all the above axioms are true for the intended interpretation of " = " as the identity relation on the domain of infinite trees. Axioms 1 to 4 are the usual axioms for = and are certainly true in the intended interpretation. Axiom 5 is true by Colmerauer's solvable-form theorem [2]. This theorem states that a system of equations $\{x_1 = t_1, \ldots, x_n = t_n\}$ has a solution in the domain of infinite trees, provided the $x_i$'s are distinct variables.

Now we are in a position to prove our main result, which amounts to the soundness of Prolog II. Intuitively, it states that every answer computed by Prolog II is correct with respect to the first order theory consisting of the homogeneous form of the program plus the equality theory $E$.

**Proposition 2.** Let $P$ be a program, $P'$ its homogeneous form, $G$ a goal and $E$ the above equality theory for Prolog II. If Prolog II solves the goal $G$, then $P' \cup E \cup \{G\}$ is unsatisfiable.

**Proof** Since the general procedure uses resolution, it produces intermediate goals all of which are a logical consequence of $P' \cup \{G\}$. We now verify that each of the five transformations of Prolog II can be justified on the basis of resolution steps using the equality theory $E$.

**Compaction**

Consider a goal $\leftarrow y = y, e_1, \ldots, e_k$, where $e_1, \ldots, e_k$ are equations. Elimination of $y = y$ is justified by resolving the goal with the equality axiom $\forall x\ x = x$. Thus $\leftarrow e_1, \ldots, e_k$ is a logical consequence of $\{\leftarrow y = y, e_1, \ldots, e_k\} \cup E$.

**Variable Anteposition**

This is justified in a similar way to compaction, but using axiom 2.

**Splitting**

Resolve with axiom 4.

**Confrontation**

It suffices to show that $\leftarrow x = t_1,\ t_1 = t_2$ is a logical consequence of $\{\leftarrow x = t_1,\ x = t_2\} \cup E$. Indeed we have the following derivation:

$\leftarrow x = t_1,\ x = t_2$

$\leftarrow x = t_1,\ x = t_1,\ t_1 = t_2$ (resolving with an instance of axiom 3)

$\leftarrow x = t_1,\ t_1 = t_2$

**Variable elimination**

We let $s[x/y]$ denote the result of replacing in $s$ all occurrences of $x$ (if any) by $y$. The following lemma will be useful.

**Lemma**

$x = y \rightarrow s = s[x/y]$

and $x = y \rightarrow s[x/y] = s$

are logical consequences of $E$.

The proof is by repeated applications of axioms 1 and 4, plus an application of axiom 2.

To justify variable elimination, it suffices to show that $\leftarrow x = y,\ s[x/y] = t[x/y]$ is a logical consequence of $\{\leftarrow x = y,\ s = t\} \cup E$. Indeed we have the following derivation:

$\leftarrow x = y,\ \ s = t$

$\leftarrow x = y,\ \ s = s[x/y],\ s[x/y] = t$ (axiom 3)

$\leftarrow x = y,\ \ x = y,\ s[x/y] = t$ (lemma)

$\leftarrow x = y,\ \ s[x/y] = t$

$\leftarrow x = y,\ \ s[x/y] = t[x/y],\ t[x/y] = t$ (axiom 3)

$\leftarrow x = y,\ \ s[x/y] = t[x/y],\ x = y$ (lemma)

$\leftarrow x = y,\ \ s[x/y] = t[x/y]$.

Finally, the last step in a Prolog II computation is the application of the solvable form theorem. From a logical point of view, this is equivalent to an application of axiom 5 above.

This completes the proof of the proposition. □

## 6. Concluding Remarks

In [3], Colmerauer extends the theoretical model of Prolog II to cope with inequalities. We have not attempted to deal with these.

Note that the general procedure can be followed by the use of any theory of equality. We have given two useful theories in this paper. It should be interesting to consider other equality theories. We are particularly interested in theories suggested by two existing systems related to Prolog. The first is Goebel's DLOG [5] logic-based database management system which uses two different equality theories: one for equality of descriptions and the other for heuristic evaluation of queries. The second is Kornfeld's version of Prolog [6] with

an extended unification.

## 7. Acknowledgments

We are indebted to Imperial College of the University of London for its hospitality which made this work possible. MHvE gratefully acknowledges support from the UK Science and Engineering Research Council. We also acknowledge our debt to the Canadian National Science and Engineering Research Council for providing document preparation facilities.

## 8. References

[1] K.L. Clark: Predicate logic as a computational formalism. Research Report 79/59, Department of Computing, Imperial College, 1979.

[2] A. Colmerauer: Prolog and infinite trees. pp. 231-251 in: K.L. Clark and S.A. Tarnlund (eds.): Logic Programming. Academic Press, 1982.

[3] A. Colmerauer et al.: Prolog II documentation.

[4] M.H. van Emden: Programming with resolution logic. pp. 266-299 in: E.W. Elcock and D. Michie (eds.): Machine Intelligence 8. Ellis Horwood, 1977.

[5] R.G. Goebel: A logic data model for the machine representation of knowledge. PhD Dissertation, Dept. of Computer Science, University of British Columbia, 1983.

[6] W.A. Kornfeld: Equality for Prolog. Proc. 8th International Joint Conference on Artificial Intelligence. A. Bundy (ed.), William Kaufmann, Los Altos, 1983.

[7] R.A. Kowalski: Predicate logic as a programming language. Proc. IFIP74, pp. 569-574.

[8] J.W. Lloyd: Foundations of logic programming. Technical Report 82/7, University of Melbourne, revised May 1983.