

Single Rules form Canonical Logic Programs

Michael J. Maher

IBM Thomas J. Watson Research Center,
P.O. Box 704, Yorktown Heights, NY 10598, U.S.A.

11 January, 1988

Abstract

It has been conjectured [Blair] that every definite clause logic program containing exactly one rule is canonical. This note presents a simple proof of this conjecture.

1 Preliminaries

We assume the underlying language contains at least one constant a . The *Herbrand universe* is the set of ground (i.e. variable-free) terms in the underlying language, and the *Herbrand base* HB is the set of ground atoms. We use \equiv to denote syntactic identity. We measure the size of terms as follows:

$size(x) = 1$ when x is a variable or a constant

$size(f(t_1, \dots, t_n)) = 1 + \sum_{i=1}^n size(t_i)$ when f is an n -ary function or predicate symbol

A *substitution* is a mapping from variables to terms which is the identity function on all but a finite number of variables. The natural extensions mapping terms to terms and atoms to atoms are also called substitutions. The identity substitution is denoted by ε . A *renaming* is an invertible substitution, that is, a substitution α such that for some substitution α^{-1} , $\alpha \circ \alpha^{-1} = \alpha^{-1} \circ \alpha = \varepsilon$. A substitution μ is *idempotent* if $\mu = \mu \circ \mu$. A substitution α is *more general than* a substitution β if $\beta = \alpha \circ \gamma$ for some substitution γ . A *unifier* of two atoms A and B is a substitution θ such that $A\theta \equiv B\theta$. With respect to a unification problem there is a class of unifiers (the *most general unifiers*) which are more general than every other unifier. These unifiers differ only by the composition of a renaming, and the class contains an idempotent substitution [LMM].

A *definite clause* or *rule* has the form $A \leftarrow B_1, B_2, \dots, B_n$ where A is an atom (the *head*) and B_1, B_2, \dots, B_n denotes a conjunction of atoms B_i , (the *body*). A *program* P is a collection of definite clauses.

For a logic program P a function T_P is defined to map subsets of the Herbrand base to subsets of the Herbrand base as follows [VEK]:

$$T_P(I) = \{A \in HB : \text{there is a ground instance } A \leftarrow B_1, B_2, \dots, B_n \\ \text{of a clause in } P \text{ such that } \{B_1, B_2, \dots, B_n\} \subseteq I \}$$

The following sets are defined by induction:

$$\begin{aligned} T_P \downarrow 0 &= HB \\ T_P \downarrow (k + 1) &= T_P(T_P \downarrow k) \\ T_P \downarrow \omega &= \bigcap_{k < \omega} T_P \downarrow k \end{aligned}$$

T_P is monotonic on the complete lattice of subsets of HB ordered by set inclusion. Consequently T_P has a greatest fixedpoint $gfp(T_P)$.

A *variant* of a syntactic object is the result of renaming the variables in that object. We will assume that, whenever a variant is used, it contains only new variables.

A *goal* is a collection of atoms. A *derivation* for a program P and initial goal G_0 is a (finite or infinite) sequence of goals $\{G_i\}$. Consecutive goals are related in the following manner: there is an idempotent substitution θ_i such that for each $A \in G_i$ there is a variant $H \leftarrow B_1, \dots, B_n$ of a clause of P such that θ_i is a unifier of A and H , θ_i is a most general such unifier, and $G_{i+1} = (\cup_{A \in G_i} \{B_1, \dots, B_n\})\theta_i$. This derivation step is called *BF-resolution* [WML]. This formalization of execution is chosen to simplify the proof. BF-resolution is equivalent to any fair SLD-resolution strategy for success, finite failure and non-termination [WML].

A derivation is infinite unless, for some goal in the derivation, there is no next goal. There are two cases. A derivation is *successful* if some G_i is empty. Otherwise the derivation is said to be *finitely failed*. A ground derivation is a derivation except that θ_i is a unifier of atoms in the goal and their respective heads such that G_{i+1} is ground. It is equivalent to say that a ground derivation is a derivation using the ground instances of clauses of P .

There are three sets of ground atoms which correspond to finitary computations: the success set $SS(P) = \{ A : A \text{ has a successful derivation for } P \}$, the finite failure set $FF(P) = \{ A : A \text{ has a finite failed SLD tree for } P \}$ and the ground finite failure set $GFF(P) = \{ A : \text{every ground derivation of } A \text{ for } P \text{ is finitely failed} \}$. BF-resolution guarantees that A is in $FF(P)$ exactly when the goal A terminates with failure. Success or finite failure of an atom can be discovered finitely using BF-resolution. Thus $SS(P)$ and $FF(P)$ correspond to terminating computations. However $GFF(P)$ does not represent only terminating computations; although every ground derivation of an atom in $GFF(P)$ is finitely failed, there may be infinitely many of them.

A program P is said to be *canonical* [JS] if $T_P \downarrow \omega = GFP(T_P)$. Equivalently, P is canonical if $FF(P) = GFF(P)$ [JLM]. Consequently, to show that a program is canonical it suffices to show that every ground atom which has an infinite derivation has an infinite *ground* derivation. Interest in canonical programs arises from the fact that it is sufficient to consider only Herbrand models of the Clark completion of P when giving a model-theoretic characterization of finitely failed ground goals.

2 The Result

Let P contain a single rule, which we will also denote by P . Clearly if P has an empty body or P contains more than one predicate symbol then P is canonical. Otherwise let p be the predicate symbol of P and let p have arity n .

Consider the execution of the goal $p(x_1, \dots, x_n)$, which we denote by G . If G fails finitely then P must be canonical, so we assume that the execution is non-terminating. Let $\alpha_i = \theta_1 \dots \theta_i$ be the substitution computed after i steps. If $\text{size}(G\alpha_i)$, $i = 1, 2, \dots$ is not bounded then every ground goal $p(t_1, \dots, t_n)$ must fail finitely. This is because we can view execution of $p(t_1, \dots, t_n)$ as execution of $p(x_1, \dots, x_n), x_1 = t_1, \dots, x_n = t_n$, and the bindings generated by $p(x_1, \dots, x_n)$ must eventually disagree with the equations. In this case P must be canonical.

If $\text{size}(G\alpha_i)$, $i = 1, 2, \dots$ is bounded then there must be a j such that for every $i \geq j$, $G\alpha_j$ and $G\alpha_i$ can differ only by a renaming of variables which do not appear in G . We say that G has *stabilized* after j steps.

Clearly execution of any atom $p(t_1, \dots, t_n)$ which is unifiable with $G\alpha_j$ will stabilize after j steps. Execution of an arbitrary goal must either finitely fail or else stabilize after j steps since, apart from variable renamings, the execution will not produce any bindings which affect the computed answer substitution after the j 'th step. (In the latter case variables would be assigned terms which are the most general instances of the terms they would be assigned by executing each atom in the goal separately.) Consequently any variable introduced during such an execution will have received bindings which stabilize after j steps.

Consider the execution of a ground atom A which has an infinite derivation. (Because P contains only one rule A has only one derivation.) Every variable which appears in the derivation either is eventually bound to a non-variable term which has stabilized j steps after the variable was introduced, or is bound only to other variables. We can construct an infinite ground derivation by using ground instances of the original input clauses, obtained by replacing variables bound only to other variables with the constant a , and replacing variables bound to a non-variable term by the stabilized term further instantiated by replacing any variables in that stabilized term a . This choice of ground input clauses ensures that the ground derivation does not fail, and so it must be infinite.

Since every ground atom with an infinite derivation has an infinite ground derivation P must be canonical

Proposition 1 *A logic program which consists of a single rule is canonical.*

Clearly the important property of P is the stabilization of terms to which variables are bound. Thus this argument shows that any program which is such that every variable in every derivation of a ground atom has bindings which eventually stabilize, is canonical.

It is easy to construct programs containing two rules which are non-canonical. For example

```
q          :- p(Y)
p(f(X))  :- p(X)
```

is non-canonical (where we assume that the underlying language contains a constant). Since programs containing only 0-ary predicate symbols and programs without non-constant function symbols are canonical, the above program would seem to be the simplest non-canonical program.

References

- [Blair] H. Blair, Decidability in the Herbrand Base, *Workshop on Foundations of Deductive Databases and Logic Programming*, 1987.
- [VEK] M.H. van Emden and R.A. Kowalski, The Semantics of Predicate Logic as a Programming Language, *Journal of the ACM* 23, 4 (1976), 733–742.
- [JLM] J. Jaffar, J-L. Lassez and M.J. Maher, Some Issues and Trends in the Semantics of Logic Programming, *Proc. 3rd. Int. Conf. on Logic Programming*, London, 1987, Lecture Notes in Computer Science 225, 223–241.
- [JS] J. Jaffar and P.J. Stuckey, Canonical Logic Programs, *Journal of Logic Programming* 3, 2, 143–155, 1986.
- [LMM] J-L. Lassez, M.J. Maher and K.G. Marriott, Unification Revisited, in: *Foundations of Deductive Databases and Logic Programming*, J. Minker (Ed), Morgan-Kaufmann, to appear.
- [WML] D.A. Wolfram, M.J. Maher and J-L. Lassez, A Unified Treatment of Resolution Strategies for Logic Programs, *Proc. 2nd International Logic Programming Conference*, Uppsala, Sweden, July 1984, 263–276.

Note: This paper was Research Report RC 13528 of the IBM - T.J. Watson Research Center, Yorktown Heights, NY, USA. It has been reconstituted with optical character recognition from hard copy. Pagination, layout and symbols have changed slightly. There might still be errors in the text.