A Procedural Semantics for Well Founded Negation in Logic Programs

Kenneth A. Ross
Stanford University
kar@cs.stanford.edu

Abstract

We introduce global SLS-resolution, a procedural semantics for well-founded negation as defined by Van Gelder, Ross and Schlipf. Global SLS-resolution extends Przymusinski's SLS-resolution, and may be applied to all programs, whether locally stratified or not.[1] Global SLS-resolution is defined in terms of global trees, a new data structure representing the dependence of goals on derived negative subgoals. We prove that global SLS-resolution is sound with respect to the well-founded semantics, and complete for non-floundering queries. While not effective in general, global SLS-resolution is effective for classes of "acyclic" programs, and can be augmented with a memoing device to be effective for all function-free programs.

This research was supported by the National Science Foundation under grant IRI-87-22886, by a grant from IBM Corporation, and by the United States Air Force Office of Scientific Research under contract AFOSR-88-0266.

A preliminary version of this paper was presented at the Eighth ACM Symposium on the Principles of Database Systems in March, 1989 [24].

Running title: "Procedural Semantics for Well Founded Negation."

Address for correspondence: Ken Ross
Department of Computer Science
Stanford University
Stanford, CA, 94305

[1] Przymusinski has independently described a similar extension of SLS-resolution. See Section 3 for further discussion.

1 Introduction

Much recent work has been concerned with negation in logic programs. Extending Horn clause programs to allow negation has not been a straightforward task, and various alternative semantics have been proposed. These proposals have come from both the logic programming community and the deductive database community, and the various approaches attempt to give an intuitive meaning to negation incorporating some form of default reasoning.

The first approach, due to Clark [6], was to define the "completion" of a program. The semantics of the program is then given by the logical consequences of the completion. For a detailed description of this approach see [27, 28, 13]. An alternative approach was taken by Fitting [7] and Kunen [11], who interpreted the completion in terms of 3-valued logic in order to overcome some anomalies with the completion when interpreted in a 2-valued sense.

Based on the completion, Clark proposed a top-down procedural semantics known as "Negation as Failure," which when combined with SLD-resolution [29] is referred to as SLDNF-resolution. This method is sound with respect to the completion of the program, and is complete for Horn programs (possibly with negative subgoals in the goal only) [9].

Another approach was taken by Przymusinski [22]. Przymusinski defined the class of "perfect models" of a program, and argued that the semantics of the program be given by the logical consequences of the (unique) perfect model. For locally stratified programs (and hence also for stratified programs [5, 1, 30]) there is guaranteed to exist a unique perfect Herbrand model, so the semantics is well-defined in these cases.

Based on the perfect model approach, Przymusinski introduced SLS-resolution [21]. SLS-resolution is a top-down procedural semantics that uses an extension of SLD-resolution to answer queries. Przymusinski showed that for stratified programs with non-floundering queries, SLS-resolution is sound and complete with respect to the perfect model of the program. Unfortunately, SLS-resolution is not effective. However it was argued in [21] that SLS-resolution may be considered a theoretical construct, an ideal query answering procedure to which various effective approximations may be compared.

Various other approaches have been proposed. Minker's "Generalized Closed World Assumption" [16] which is based on minimal models is closely related to McCarthy's "Circumscription" [15]. Gelfond and Lifschitz have defined the class of "Stable Models," and argued that the semantics of a program be determined by these models [8]. In the context of disjunctive databases, Ross and Topor have introduced the "Disjunctive Database Rule" [23].

As a development of these approaches, Van Gelder, Ross and Schlipf [31] introduced the "Well-Founded Semantics" for logic programs with negation. For a discussion of the relationship between the well-founded semantics and the various other semantics see [31].

The purpose of this paper is to present a top-down procedural implementation of the well-founded semantics. We call this procedure, which extends SLS-resolution, "Global SLS-resolution." We show that for arbitrary programs with non-floundering queries, global SLS-resolution is sound and complete with respect to the well-founded model of the (augmented) program. (What we refer to as completeness some authors call "partial completeness" to indicate possible non-termination.) As with SLS-resolution, global SLS-resolution is not effective in general. We cannot expect to have an effective procedure as the well-founded partial model is not recursively enumerable in general, even when restricted to non-floundering literals. Nevertheless, for subclasses of programs we can guarantee termination.

We discuss this further in Section 7.

1.1 Terminology

In this section we describe our notation, and the class of logic programs we consider. Where possible, we use the standard terminology of [13].

Definition 1.1: A *normal program clause* is a clause of the form

$$A \leftarrow L_1, \ldots, L_n$$

where A is an atom, and $L_1, \ldots, L_n$ are (positive or negative) literals. We refer to A as the *head* of the clause and $L_1, \ldots, L_n$ as the *body* of the clause. All variables are assumed to be universally quantified at the front of the clause, and the commas in the body denote conjunction. If the body of the rule is empty, we may omit the "$\leftarrow$" symbol.

A *normal logic program* (abbreviated to *program* hereafter) is a finite set of program clauses. □

Definition 1.2: The *Herbrand universe* of a program P is the set of all variable-free terms that may be formed from the constants and function symbols appearing in P. If there are no constants in P then we treat P as if it had a single extra constant symbol. □

Definition 1.3: A *query* is a set of literals. We interpret a query Q as the conjunction of the literals it contains. We use the notation $\leftarrow Q$ to denote a *goal*, so that resolution may be applied directly to the goal. A *ground substitution* for a goal G is a substitution of terms from the Herbrand universe of the program for all the variables in G. □

Definition 1.4: A *computation rule* is a rule for selecting one or more literals from a query. Note that it may depend on all previous queries as well as the current query. □

Definition 1.5: The *Herbrand instantiation* of a logic program is the set of rules obtained by substituting terms in the Herbrand universe for all the variables in each rule in every possible way. An *instantiated rule* is an element of the Herbrand instantiation. □

Definition 1.6: Let S be a set of literals. We denote the set formed by taking the complement of each literal in S by $\neg \cdot S$.

- We say that the literal q is *inconsistent with* S if $q \in \neg \cdot S$.
- Sets of literals R and S are *inconsistent* if some literal in R is inconsistent with S.
- A set of literals is *inconsistent* if it is inconsistent with itself; otherwise it is *consistent*.

The set of positive literals in S is denoted by $pos(S)$, and the set of negative literals by $neg(S)$. □

Definition 1.7: Given a program P, a *partial interpretation* I is a consistent set of literals whose atoms are in the Herbrand base of P. A *total interpretation* is a partial interpretation that contains every atom in the Herbrand base, possibly negated. (Note that ours is a "3-valued" definition of "interpretation.")

A *total model* is a total interpretation such that every instantiated rule is satisfied. A *partial model* is a partial interpretation that can be extended to a total model. □

2 Unfounded Sets and the Well-Founded Semantics

In this section we present the definition of the well-founded semantics for logic programs. For a more detailed presentation with examples see [31].

Definition 2.1: Let P be a program and H its Herbrand base. Let I be a given partial interpretation. We say $A \subseteq H$ is an *unfounded set of P with respect to I* if each atom $p \in A$ satisfies the following condition: For each (Herbrand) instantiated rule r of P whose head is p, at least one of the following holds:

1. The complement of some literal in the body of r is in I.

2. Some positive literal in the body of r is in A.

A literal that makes either of the above conditions true is called a *witness of unusability* for rule r with respect to I. $\Box$

Definition 2.2: The *greatest unfounded set* of P *with respect to I*, denoted by $U_P(I)$, is the union of all sets that are unfounded with respect to I. (The "greatest unfounded set" is easily seen to be an unfounded set.) $\Box$

Definition 2.3: Transformations T_P, U_P and W_P from sets of literals to sets of literals are defined as follows.

- $p \in T_P(I)$ if and only if there is some (Herbrand) instantiated rule r of P such that r has head p and each literal in the body of r is in I.

- $U_P(I)$ is the greatest unfounded set of P with respect to I, as in Definition 2.2.

- $W_P(I) = T_P(I) \cup \neg \cdot U_P(I)$. $\Box$

It is straightforward to show that W_P is monotonic, and so has a least fixpoint. We denote this least fixpoint by $M_{WF}(P)$, and call this the well-founded (partial) model of P.[2] Note that $M_{WF}(P)$ is a "three-valued model" in the sense of Fitting [7]. A ground atom A may appear positively, negatively or not at all in $M_{WF}(P)$.

For ground queries $Q = \{p_1, \ldots, p_n, \neg q_1, \ldots, \neg q_m\}$, $M_{WF}(P) \models Q$ if and only if $Q \subseteq M_{WF}(P)$. Also $M_{WF}(P) \models \neg Q$ (the negation of the conjunction of elements of Q) if and only if either some q_j or some $\neg p_i$ is in $M_{WF}(P)$.

We write $M_{WF}(P) \models \forall Q$, where Q is a query or negated query, possibly containing variables, to mean $M_{WF}(P) \models Q\alpha$ for every ground substitution α of terms from the Herbrand universe for the variables of Q. We write $M_{WF}(P) \models \exists Q$, to mean $M_{WF}(P) \models Q\alpha$ for some ground substitution α. Adopting such a definition of "$\models$" is equivalent to considering only Herbrand models of $M_{WF}(P)$.

We now give an alternative definition of the well-founded partial model that has technical advantages for the proofs of our results. Define

$$\bar{T}_P(I) = T_P(I) \cup I.$$

[2]For a justification that it is a partial model see [31].

Since T is monotonic, so is $\bar{T}$. Let V_P be defined by

$$V_P(I) = (\bigcup_{k=1}^{\infty} \bar{T}_P^k(I)) \cup \neg \cdot U_P(I).$$

Again V_P is monotonic, and has a least fixpoint. We can construct the least fixpoint of V_P by the following transfinite iteration.

Definition 2.4: Let α and β be countable ordinals. The partial interpretations I_α and I^∞ are defined recursively by

1. For limit ordinal α,
$$I_\alpha = \bigcup_{\beta<\alpha} I_\beta$$
Note that 0 is a limit ordinal, and $I_0 = \emptyset$.

2. For successor ordinal $\alpha + 1$,
$$I_{\alpha+1} = V_P(I_\alpha)$$

3. Finally, define
$$I^\infty = \bigcup_\alpha I_\alpha$$

Following [17], for any literal p in I^∞, we define the *stage* of p (written $stage_p$) to be the least ordinal α such that $p \in I_\alpha$. The above definition implies that the stage is always a successor ordinal for literals in I^∞. □

It is straightforward to show that the sequence of partial interpretations I_α is monotonically increasing, i.e., $I_\beta \subseteq I_\alpha$ if $\beta \leq \alpha$.

Lemma 2.1: I^∞ is the least fixpoint of V_P, and is equal to $M_{WF}(P)$.
Proof: That I^∞ is the least fixpoint of V_P is a consequence of classical fixpoint results of Tarski for monotonic operators over complete lattices. It is clear by the definitions of V_P and W_P that for every partial interpretation I, $W_P(I) \subseteq V_P(I)$, and hence $M_{WF}(P) \subseteq I^\infty$.

We now define a sequence of partial interpretations that is similar to the sequence of Definition 2.4, except that we iterate W_P rather than V_P. Let α and β be countable ordinals. The sets I'_α and $M_{WF}(P)$ of partial interpretations are defined recursively by

1. For limit ordinal α, $I'_\alpha = \bigcup_{\beta<\alpha} I'_\beta$. 0 is a limit ordinal, and $I'_0 = \emptyset$.

2. For successor ordinal $\alpha + 1$, $I'_{\alpha+1} = W_P(I'_\alpha)$

3. Finally, $M_{WF}(P) = \bigcup_\alpha I'_\alpha$

From the definitions of I_α and I'_α, it is easily shown by induction that $I_\alpha \subseteq I'_{\omega\alpha}$, where $\omega\alpha$ is the ordinal product of ω and α. The proof uses the observation that $\bar{T}$ is continuous, and hence has closure ordinal ω. Hence $I^\infty \subseteq M_{WF}(P)$, and the result follows. ∎

3 Global Trees and Global SLS-Resolution

In this section we define SLP-trees, which form the basis of the definition of Global Trees. These in turn form the basis of the definition of global SLS-resolution.

Definition 3.1: Let R be a computation rule. We say that R is *safe* if it never selects a non-ground negative literal from a query. We say that R is *positivistic* if it selects positive literals ahead of negative ones. R is *negatively parallel* if given a query containing only negative literals, it selects all (and only) the ground negative literals appearing in the query. The positivistic and negatively parallel conditions imply safety; a positivistic and negatively parallel rule is said to be *preferential.* $\Box$

In order to achieve soundness in a derivation, we require the computation rule to be safe. As we shall see, in order to achieve completeness of global SLS-resolution with respect to the well-founded semantics we will require a positivistic and negatively parallel computation rule. We now define SLP-trees. The "SLP" stands for "Linear resolution using a Positivistic Selection rule."

Definition 3.2: (SLP-trees) Let G be the goal $\leftarrow Q$ and let R be a positivistic computation rule. We define the SLP-tree T_G for G. The root node of T_G is G. If the goal $H = \leftarrow Q'$ is any node of T_G then its children are obtained as follows:

- If Q' contains a positive literal then the literal L selected by R from Q' must be positive. Let U_L be the set of program clauses whose heads unify with L. The children of H are obtained by resolving H with (a variant of) each of the clauses in U_L over the literal L using most general unifiers. If there is no such K then H has no children, and is a *dead* leaf.

- If Q' is empty, or contains only negative subgoals then Q' is an *active* leaf.

A *branch* of T_G is an acyclic path from the root of T_G. We associate with each active leaf L its *computed most general unifier*, which is the composition of the most general unifiers used along the branch to L. $\Box$

See Example 3.1 for examples of SLP-trees. We now define the global tree for a goal in terms of SLP-trees. The global tree may be thought of as an "OR/NOR" tree in the style of "AND/OR" trees.

Definition 3.3: We define the *global tree* Γ_G for a goal G. The nodes of Γ_G are of three types: *negation nodes*, *tree nodes*, and *non-ground nodes*. Tree nodes are actually SLP-trees for intermediate goals. The root of Γ_G is the SLP-tree for the goal G. An *internal* tree node is a tree node that is not the root.

Let T_H be any tree node of Γ_G. The children of T_H are negation nodes, one corresponding to each active leaf of T_H. Note that there will be a negation node corresponding to an empty active leaf.

Let J be a negation node, corresponding to the active leaf $\leftarrow Q$. Let $Q = \{\neg q_1, \ldots, \neg q_n\}$ where $n \geq 0$. J has n children, one corresponding to each q_i.[3] For $i = 1, \ldots, n$, if q_i is

[3] A negation node corresponding to an empty active leaf will have no children.

ground then the child corresponding to q_i is the tree node $T_{\leftarrow q_i}$; otherwise the corresponding child is a non-ground node. Non-ground nodes have no children.

Every node has associated with it a *status* (either successful, failed, floundered or indeterminate) according to the following rules. Successful and failed nodes also have an associated *level*.

1. Every non-ground node is *floundered*.

2. (a) If some child of a negation node J is a successful tree node, then we say that J is *failed*. The level of J is the minimum level of all its successful children.

   (b) If every child of a negation node J is a failed tree node, or if J has no children, then we say J is *successful*. The level of J is the least ordinal upper bound of the levels of the children of J. (If J has no children, then it has level 0.)

   (c) If at least one child of a negation node J is a floundered node, and all children of J are not successful, then we say that J is *floundered*.

3. (a) If every child of a tree node T is a failed negation node, or if T is a leaf of Γ_G (i.e. T has no active leaves) then we say T is *failed*. The level of T is $\alpha + 1$, where α is the least ordinal upper bound of the levels of the children of T. (T has level 1 if it has no children.)

   (b) If some child of a tree node T is a successful negation node, then we say T is *successful*. An internal tree node T has level one more than the minimum level of all its successful children. The root tree node may have several associated levels, one for each successful child; the level of the root tree node with respect to such a successful child is one more than the level of the child.

   (c) If at least one child of a tree node T is a floundered negation node, then we say that T is *floundered*.

4. Any node that can be proved successful, failed or floundered according to the above rules is said to be *well determined*. Any node that is not well determined is said to be *indeterminate*. □

See Figure 4 in Example 3.1 for an example of a global tree. Note that the definition of the global tree itself is top-down, but that the status of the nodes as successful, failed or floundered is defined bottom-up. The correspondence between the level of a goal and the stage a literal is put in to the well founded model by iterating V_P will be discussed in Section 4.

Let L be an active leaf of a tree node in Γ_G. We may say that L is successful, failed, floundered or indeterminate if the corresponding negation node is respectively successful, failed, floundered or indeterminate. We may also say that the goal G is successful, failed, floundered or indeterminate if T_G is respectively successful, failed, floundered or indeterminate in Γ_G.

Definition 3.4: Let G be a goal. A *successful branch* of T_G is a branch of T_G that ends at a successful leaf. An *answer substitution* for G is given by $\theta = \theta_1\theta_2 \ldots \theta_n$ where the θ_i are the most general unifiers used at each step along a successful branch of T_G. In other words, an answer substitution is the computed most general unifier at a successful leaf. □

A tree node may be both successful and floundered, although no other pair of statuses is possible for a single node. The reason the root tree node is treated differently in rule 3(b) above is that there may be different answer substitutions for the goal that succeed at different levels. Internal tree nodes have ground goals if they are not floundered, and so there cannot be multiple answer substitutions in this case. If we need to distinguish between levels of a goal G with answer substitutions $\theta_1, \theta_2, \ldots$ (each corresponding to a distinct leaf) then we will refer to the level of G *with respect to* θ_i for each i. Failed goals have a unique level.

We may consider a non-ground node as a special type of tree node, in which case Γ_G is a bipartite graph. For every active leaf of a tree node T there will be an edge from T to a negation node. A branch from a negation node to a tree node denotes (negated) membership in the corresponding subgoal.

Definition 3.5: Global SLS-resolution is the top-down process of finding all answer substitutions for a goal G using a preferential computation rule. When a derived goal G' containing only negative literals is encountered, the appropriate SLP-trees for the complements of the ground literals in G' are recursively constructed in parallel. □

Global SLS-resolution is, in essence, an appropriate traversal of the global tree for a goal. This process incorporates the traversal of the SLP-trees corresponding to the tree-nodes of the global tree. Selecting a positive literal from a goal corresponds to moving one node deeper in the SLP-tree for the current goal; selecting negative literals corresponds to moving one level deeper in the global tree, by passing through a negation node.

Observe that a goal can have an infinite level even if it involves only finite recursion through negation.

Example 3.1: (Van Gelder) Let P be the program

$$
\begin{aligned}
e(s(0), s(s(0)))& \\
e(s(X), s(s(X)))&\leftarrow e(X, s(X)) \\
e(s(0), 0)& \\
e(s(X), 0)&\leftarrow e(X, 0)
\end{aligned}
$$

$$
\begin{aligned}
w(X)&\leftarrow \neg u(X) \\
u(X)&\leftarrow e(Y, X), \neg w(Y)
\end{aligned}
$$

In this example, the integer i is represented by the term $s^i(0)$. The transitive closure of the e predicate defines an ordering among the integers; when (X, Y) is in the transitive closure of e we will write $X < Y$. The ordering is non-standard, with 0 representing the ordinal ω :

$$s^1(0) < s^2(0) < s^3(0) < \cdots < 0.$$

The predicate w is true of an integer j if and only if there are no infinite decreasing sequences starting with j. The predicate u is true precisely when w is false.

Let i, w_i and u_i be abbreviations for $s^i(0)$, $T_{\leftarrow w(s^i(0))}$ and $T_{\leftarrow u(s^i(0))}$ respectively. Then the appropriate SLP-trees and global tree for the goal $\leftarrow w(0)$ are given in Figures 1 to 4. We use the symbol $\bullet$ to denote negation nodes, and omit the "$\leftarrow$" symbol from goals for clarity.

For $n \geq 1$, the goal $\leftarrow w(s^n(0))$ has level $2n$, and so the goal $\leftarrow w(0)$ has level $\omega + 2$. Note that $\leftarrow w(0)$ has infinite level despite the fact that every branch of the global tree for

$$\begin{array}{c} w(i) \\ | \\ \neg u(i) \end{array}$$

Figure 1: SLP-trees w_i, for $i \geq 0$

$$\begin{array}{c} u(1) \\ | \\ e(Y,1), \neg w(Y) \end{array} \qquad \begin{array}{ccc} & u(i) & \\ & | & \\ & e(Y,i), \neg w(Y) & \\ & | & \\ & e(Y',i-1), \neg w(Y'+1) & \\ & | & \\ & \vdots & \\ & | & \\ & e(Y^{(i-2)},2), \neg w(Y^{(i-2)}+(i-2)) & \\ / & & \backslash \\ \neg w(i-1) & & e(Y^{(i-1)},1), \neg w(Y^{(i-1)}+(i-1)) \end{array}$$

Figure 2: SLP-trees u_1 and u_i for $i \geq 2$

$$\begin{array}{cccccc} & u(0) & & & & \\ & | & & & & \\ & e(Y,0), \neg w(Y) & & & & \\ / & & \backslash & & & \\ \neg w(1) & & e(Y',0), \neg w(Y'+1) & & & \\ & / & & \backslash & & \\ & \neg w(2) & & e(Y'',0), \neg w(Y''+2) & & \\ & & / & & \ddots & \\ & & \neg w(3) & & & \end{array}$$

Figure 3: SLP-tree u_0

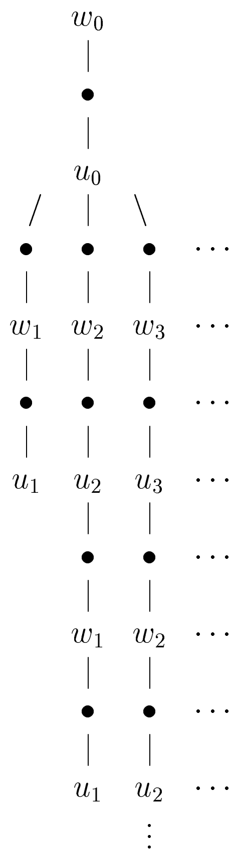

Figure 4: Global tree for $\leftarrow w(0)$. Each w_i is successful, and each u_i is failed.

$\leftarrow w(0)$ is finite. Note that this program does have a well-founded total model, in which $w(0)$ is true, even though it is not locally stratified. $\Box$

In order for global SLS-resolution to find all answer substitutions, and not get "lost" down an infinite branch of an SLP-tree, an appropriate method for searching SLP-trees (such as breadth first search) is needed. For well-determined goals, global SLS-resolution will (given infinite time) traverse the appropriate global tree. For goals that are indeterminate, global SLS-resolution will recurse through an infinite number of negation nodes. For a discussion of the non-effectiveness of global SLS-resolution see Section 7.

There is a close relationship between global SLS-resolution and SLS-resolution. The first difference is that we insist the computation rule be preferential. This restriction is necessary to achieve completeness over the broader class of all programs. (Recall that SLS-resolution is not well-defined for programs that are not locally stratified.)

The second difference is that the definition of SLS-resolution requires a level mapping to be associated with the literals and goals simply in order to define the SLS-tree for a goal. Our construction relaxes this requirement by allowing all subsidiary SLP-trees to be constructed recursively.

Recently, Przymusinski has independently defined a similar extension of SLS-resolution using induction on what he terms the "dynamic stratification" of a program [19]. Dynamic stratification corresponds roughly to what we have called the level of the global tree for a goal. One advantage of our construction is that the level is a *consequence* of the definition of global trees, rather than a *precondition* of its definition.

Another advantage is the explicit representation of the global tree, in which infinite branches correspond to indeterminate derivations. Przymusinski's definition only allows selection of subgoals that are known to be well-determined at lower level. However, in a top-down system, the status of subgoals is unknown until they are themselves expanded, and so such a restriction on the selection of subgoals is unlikely to be useful.

Observe that if R is not positivistic, then we will not be able to achieve completeness.

Example 3.2: [18] Let P be the program

$$\begin{array}{l} p \leftarrow \neg s, q, \neg r \\ q \leftarrow r, \neg p \\ r \leftarrow p, \neg q \\ s \leftarrow \neg p, \neg q, \neg r \end{array}$$

The model $\{s, \neg p, \neg q, \neg r\}$ for P is well-founded. However, consistently expanding the leftmost literal first (i.e. $\neg s$ before q in the first rule) will give us an apparently indeterminate result rather than a successful result for the goal $\leftarrow s$. $\Box$

Sequential expansion of ground negative subgoals rather than a parallel expansion is not sufficient to achieve completeness.

Example 3.3: Let P be the program

$$\begin{array}{l} p(x) \leftarrow \neg p(f(x)) \\ \quad q \leftarrow \neg p(a), \neg s \\ \quad s \end{array}$$

If a computation rule R chooses the leftmost negative literal first, then the goal $\leftarrow q$ will appear to be indeterminate, although $\neg q$ is in the well-founded (partial) model $\{s, \neg q\}$ for P. However, expanding both negative subgoals in the rule for q in parallel causes q to fail. $\Box$

Since only ground negative literals are expanded, they may be processed independently. We do not apply the same parallelization to positive literals, as they may generate competing bindings, and we do not want to have to resolve such conflicts.

4 Ground SLP-Trees and Ground Global Trees

In the following sections we will present our results on the soundness and completeness of global SLS-resolution. First, though, it will be convenient to consider a simplified version of SLP-trees, which we call *ground* SLP-trees. Ground SLP-trees are SLP-trees in which all goals are ground, and rules used in the construction of branches of the SLP-tree are instantiated rules. *Ground* global trees are constructed from ground SLP-trees in a similar way to the construction of global trees from SLP-trees.

We will obtain a new characterization of the well-founded model in terms of ground global trees, that facilitates the proof of soundness and completeness in the general case.

Definition 4.1: (Ground SLP-trees) Let G be the ground goal $\leftarrow Q$ and let R be a positivistic computation rule. We define the ground SLP-tree T_G^g for G. The root node of T_G^g is G. If the (ground) goal $H = \leftarrow Q'$ is any node of T_G^g, then its children are obtained as follows.

- If Q' contains a positive literal, then the literal L selected by R from Q' must be positive. In this case, the children of H are all goals K that can be obtained by resolving H with an instantiation of one of the program clauses over the literal L. If there is no such K then H has no children, and is a *dead* leaf.
- If Q' is empty, or contains only negative subgoals then Q' is an *active* leaf.

The *depth* of a node in a ground SLP-tree T is the number of edges in the shortest path from the node to the root of T. □

There is a structural similarity between SLP-trees and ground SLP-trees that will be made precise in Section 5. Since the Herbrand universe of the program may be infinite, there may be infinitely many instantiated rules with a given head. Hence, a ground SLP-tree may have an infinite branching factor at any node. SLP-trees have a finite branching factor.

The *ground global tree* Γ_G^g for a (ground) goal G is defined like the global tree for G except that tree nodes are ground SLP-trees rather than SLP-trees. A goal may be *ground successful*, *ground failed* or *ground indeterminate* in the same way that goals are successful, failed or indeterminate respectively. The difference is that the ground global tree is traversed rather than the global tree. Since all goals appearing in ground SLP-trees are themselves ground, there are no floundered nodes in a ground global tree.

For both global trees and ground global trees, the position of a tree node in the tree does not affect its status. More precisely, if G is a goal, and T_G appears in Γ_H for some goal H, then T_G is successful, failed, indeterminate or floundered in Γ_H if and only if it is respectively successful, failed, indeterminate or floundered in Γ_G, since the status of a tree node only depends on its descendants. A similar property holds for ground global trees. Further, since the level of a goal also depends only on its descendants, the level of G is identical in Γ_H and Γ_G. Hence we shall not refer to the appropriate global tree when discussing the status or level of a goal appearing therein.

On every branch to an active leaf of a ground SLP-tree, every positive literal must eventually be selected, as active leaves contain only negative literals. Further, since there is no interaction between positive literals in a ground goal, the order of selection of the positive literals in a path to an active leaf is not important. Hence, the set of active leaves in T_G^g for a given ground goal G is independent of the computation rule used, as long as the rule is positivistic. Since the status of goals in Γ_G^g depends only on the active leaves of its tree nodes, we may conclude that the semantics induced by Γ_G^g is independent of the computation rule used.[4]

In light of this observation, we have the following result.

Lemma 4.1: Let G be the ground goal $\leftarrow p_1, \ldots, p_n, \neg q_1, \ldots, \neg q_m$. Then L is an active leaf of T_G^g if and only if there exist active leaves $L_1, \ldots, L_n$ of $T_{\leftarrow p_1}^g, \ldots, T_{\leftarrow p_n}^g$ respectively such

[4]In the more general case of SLP-trees, the set of leaves is again independent of the computation rule used. Such a result may be proved using a "switching lemma." See [13] for details.

that $L = L_1 \cup \ldots \cup L_n \cup \{\neg q_1, \ldots, \neg q_m\}$. Further, the depth of L in T^g_G is equal to the sum over all i of the depths of L_i in $T^g_{\leftarrow p_i}$.
Proof: Since L is an active leaf independent of the computation rule, we may choose any positivistic computation rule for T^g_G. Let $R_1, \ldots, R_n$ be arbitrary positivistic computation rules for $T^g_{\leftarrow p_1}, \ldots, T^g_{\leftarrow p_n}$ respectively. Let R be the computation rule that first simulates R_1, expanding all "descendants" of p_1, then R_2, R_3 and so on until R_n.

L is an active leaf, at finite depth, if and only if each segment of the path to L in T^g_G (corresponding to $R_1, \ldots, R_n$ respectively) is finite, and does not terminate in a dead leaf. This occurs precisely when each $T^g_{\leftarrow p_i}$ has an active leaf, say L_i, and $L = L_1 \cup \ldots \cup L_n \cup \{\neg q_1, \ldots, \neg q_m\}$. The relation between the depths of the leaves is obvious from the construction. ∎

Lemma 4.2: Let p be a ground atom, and let G be the goal $\leftarrow p$. Let S^+ be a set of positive ground literals, and S^- a set of negative ground literals. Then

1. $p \in \bigcup_{i=1}^{\infty} \bar{T}^i_P(S^-) \iff T^g_G$ has an active leaf whose members are all in S^-.
2. $p \in U_P(S^+) \iff$ every active leaf of T^g_G has a member in $\neg \cdot S^+$.

Proof:

(1, $\Rightarrow$) If $p \in \bigcup_{i=1}^{\infty} \bar{T}^i_P(S^-)$ then $p \in \bar{T}^k_P(S^-)$ for some finite $k \geq 1$. We prove by induction on k that for all p,

$$p \in \bar{T}^k_P(S^-) \Rightarrow T^g_G \text{ has an active leaf whose members are all in } S^-.$$

Base Case: $p \in \bar{T}_P(S^-) \iff$ there is some instantiated rule r with head p and (negative) literals from S^- in the body. By definition, this holds if and only if T^g_G has an active leaf (at depth 1) whose members are all in S^-.

Induction Step: Suppose the statement above is true for $k = N$. We show it is also true for $k = N+1$. Let $p \in \bar{T}^{N+1}_P(S^-)$. Then there is some instantiated rule r with head p such that all literals $l_1, \ldots, l_n$ in the body of r are in $\bar{T}^N_P(S^-)$. Since the only negative elements of $\bar{T}^N_P(S^-)$ are actually in S^-, all of the negative literals in $\{l_1, \ldots, l_n\}$ are in S^-. By the induction hypothesis, for each of the positive literals l_i, (for $i = 1, \ldots, m$ with $m \leq n$), $T^g_{\leftarrow l_i}$ has an active leaf (say L_i) whose members are all in S^-. Let $L = L_1 \cup \ldots \cup L_m \cup \{l_{m+1}, \ldots, l_n\}$. Then L must be an active leaf of T^g_G by Lemma 4.1. Further, $L \subseteq S^-$, thus demonstrating the result for $k = N+1$.

(1, $\Leftarrow$) This argument is by induction on the depth of the active leaf. We show that if T^g_G has an active leaf at depth d, all of whose members are in S^-, then $p \in \bar{T}^d_P(S^-)$. The base case of this induction is identical to the base case of the previous argument.

Induction Step: Suppose the statement is true for all $d \leq N$. We show it is true for $d = N+1$. If T^g_G has an active leaf L at depth $N+1$ such that $L \subseteq S^-$, then some child G' of G has L as an active leaf at depth N. Let r be the instantiated rule used in deriving G' from G, given by $p \leftarrow p_1, \ldots, p_n, \neg q_1, \ldots, \neg q_m$, so that $G' = \{p_1, \ldots, p_n, \neg q_1, \ldots, \neg q_m\}$. By Lemma 4.1, for every positive literal p_i in

G', $T^g_{\leftarrow p_i}$ has an active leaf at depth at most N, whose members are all in L, and hence in S^-. By our induction hypothesis, $p_i \in \bar{T}_P^N(S^-)$. Each $\neg q_i$ is in S^- (since it is in L), and hence in $\bar{T}_P^N(S^-)$. Thus $p \in \bar{T}_P^{N+1}(S^-)$.

(2, $\Rightarrow$) Suppose $p \in U_P(S^+)$. Then for every instantiated rule r with head p, either some positive literal in the body is also in $U_P(S^+)$, or some negative literal is in $\neg \cdot S^+$. Hence every goal appearing in T^g_G has either a positive literal in $U_P(S^+)$ or a negative literal in $\neg \cdot S^+$. Since active leaves have no positive literals, it follows that every active leaf of T^g_G contains a negative literal in $\neg \cdot S^+$.

(2, $\Leftarrow$) Suppose every active leaf in T^g_G has a member in $\neg \cdot S^+$. We show $p \in U_P(S^+)$ by constructing an unfounded set U (with respect to S^+) containing p. Since $U_P(S^+)$ contains all unfounded sets, the result will follow. Let U be the set of ground atoms defined by

$$U = \{q : \text{Every active leaf of } T^g_{\leftarrow q} \text{ has a member in } \neg \cdot S^+\}$$

$p \in U$ by assumption. Let q' be an arbitrary element of U. We claim that every goal appearing in $T^g_{\leftarrow q'}$ contains either

- a positive literal p' such that $p' \in U$, or
- a negative literal l such that $l \in \neg \cdot S^+$.

We prove the claim by contradiction. Suppose the contrary, i.e., that for some goal $H = \{p_1, \ldots, p_n, \neg q_1, \ldots, \neg q_m\}$ in $T^g_{\leftarrow q'}$, no p_i is in U and no $\neg q_j$ is in $\neg \cdot S^+$. Then for each p_i there must be an active leaf (say L_i) of $T^g_{\leftarrow p_i}$ containing no members of $\neg \cdot S^+$ (by the definition of U, and by Lemma 4.1). Hence there must be an active leaf L of $T^g_{\leftarrow q'}$ that is a descendent of H, and is given by $L_1 \cup \ldots \cup L_n \cup \{\neg q_1, \ldots, \neg q_m\}$. But no element of L is in $\neg \cdot S^+$, contradicting the assumption that $q' \in U$, and thus proving the claim.

In particular, every child of $\leftarrow q'$ in $T^g_{\leftarrow q'}$ satisfies the above claim. But the children of $\leftarrow q'$ are simply the bodies of all instantated rules with q' as head. Any literal that makes the claim above true for such a rule body is a witness of unusability for that rule; the claim shows that every rule for q' has a witness of unusability. Since q' is arbitrary, by the definition of unfounded sets, U is unfounded with respect to S^+ and the result follows.

∎

The above results are false if we do not insist that S^+ and S^- contain only positive and negative literals respectively. However, as the following result shows, Lemma 4.2 is sufficient for our purposes.

Lemma 4.3: Let l be a ground literal in $M_{WF}(P)$. Suppose $l \in I_{\alpha+1}$ according to the iteration of Definition 2.4. Then

- If l is positive, then $l \in \bigcup_{i=1}^{\infty} \bar{T}_P^i(neg(I_\alpha))$.

- If l is negative, then $l \in \neg \cdot U_P(pos(I_\alpha))$.

Proof: The proof is by transfinite induction on α.

Case 1: α is a successor ordinal. Let $\alpha = \beta + 1$ and assume the result for β.

For the first part, we know $l \in \bigcup_{i=1}^{\infty} \bar{T}_P^i(I_\alpha)$ and so $l \in \bigcup_{i=1}^{\infty} \bar{T}_P^i(I'_\alpha)$ for some finite subset I'_α of I_α. (This compactness property follows from the finiteness of the bodies of instantiated rules.) By hypothesis, and since I'_α is finite, there is some $k \geq 0$ such that all the positive literals in I'_α are in $\bar{T}_P^k(neg(I_\beta))$. ($\bar{T}(S) \supseteq S$ implies that the finite union to the k^{th} term simplifies to the final term.) By monotonicity $pos(I'_\alpha) \subseteq \bar{T}_P^k(neg(I_\alpha))$. Hence $l \in \bigcup_{i=1}^{\infty} \bar{T}_P^i(\bar{T}_P^k(neg(I_\alpha)))$, and the result follows.

For the second part, suppose $l = \neg p$. We know that $neg(I_\alpha) \subseteq \neg \cdot U_P(pos(I_\beta))$ by our induction hypothesis. Hence, by monotonicity, $neg(I_\alpha) \subseteq \neg \cdot U_P(pos(I_\alpha))$. Let w be any witness of unusability for a rule r having head p, with respect to I_α. If $w \in pos(I_\alpha)$ then it certainly remains a witness with respect to $pos(I_\alpha)$. If $w \in neg(I_\alpha)$ then $w \in \neg \cdot U_P(pos(I_\alpha))$ by the above, and hence w is still a witness for r with respect to $pos(I_\alpha)$. Hence $p \in U_P(pos(I_\alpha))$ and $l \in \neg \cdot U_P(pos(I_\alpha))$.

Case 2: α is a limit ordinal. The proof in this case is a simple extension of the arguments in the successor ordinal case. The case $\alpha = 0$ is trivial since $I_\alpha = \emptyset$. For the first part, where l is positive, the proof is essentially the same once we observe that I'_α, being finite, must be a subset of I_γ for some successor ordinal $\gamma < \alpha$.

For the second part, observe that if $l \in neg(I_\alpha)$ then $l \in neg(I_{\beta+1})$ for some successor ordinal $\beta + 1$. By hypothesis, $l \in \neg \cdot U_P(pos(I_\beta))$, and since l is arbitrary, $neg(I_\alpha) \subseteq \bigcup_{\beta<\alpha} \neg \cdot U_P(pos(I_\beta))$. We can extend the argument of the successor ordinal case to show that $neg(I_\alpha) \subseteq \bigcup_{\beta<\alpha} \neg \cdot U_P(pos(I_\beta))$ implies $neg(I_\alpha) \subseteq \neg \cdot U_P(\bigcup_{\beta<\alpha} pos(I_\beta))$, by monotonicity, and hence $neg(I_\alpha) \subseteq \neg \cdot U_P(pos(I_\alpha))$. The remainder of the proof is identical.

∎

Lemma 4.4: Let l be a ground literal. Then

- If l is positive, then $l \in I_{\alpha+1} \iff l \in \bigcup_{i=1}^{\infty} \bar{T}_P^i(neg(I_\alpha))$.
- If l is negative, then $l \in I_{\alpha+1} \iff l \in \neg \cdot U_P(pos(I_\alpha))$.

Proof: The implications from left to right follow from Lemma 4.3. The implications from right to left follow by monotonicity. ∎

We now demonstrate the precise correspondence between the well-founded semantics and ground global trees.

Theorem 4.5: Let p be a ground atom, G the goal $\leftarrow p$ and α a countable ordinal. Then

- G is ground successful at level $\leq \alpha \iff p \in I_\alpha$.

- G is ground failed at level $\leq \alpha \iff \neg p \in I_\alpha$.

Proof: The argument is by induction on α.

Case 1: α is a successor ordinal. Suppose $\alpha = \beta + 1$, and suppose the statement is true for the ordinal β.

G is ground successful at level $\leq \alpha$

$\iff$ T^g_G has an active leaf $L = \{\neg p_1, \ldots, \neg p_n\}$, say, such that each $T^g_{\leftarrow p_i}$ $(i = 1, \ldots, n)$ is ground failed at level $\leq \beta$

$\iff$ $L \subseteq neg(I_\beta)$, by hypothesis

$\iff$ $p \in \bigcup_{i=1}^{\infty} \bar{T}^i_P(neg(I_\beta))$, by Lemma 4.2

$\iff$ $p \in I_\alpha$, by Lemma 4.4

G is ground failed at level $\leq \alpha$

$\iff$ every active leaf of T^g_G contains a literal $\neg q$ such that $T^g_{\leftarrow q}$ is ground successful at level $\leq \beta$

$\iff$ every active leaf of T^g_G has a member whose complement is in $pos(I_\beta)$, by hypothesis

$\iff$ $p \in U_P(pos(I_\beta))$, by Lemma 4.2

$\iff$ $\neg p \in I_\alpha$, by Lemma 4.4

Case 2: α is a limit ordinal. The truth of the above statements for $\alpha = 0$ is trivial. By the construction of global trees, goals can only be successful or failed at a level that is a successor ordinal. Also, the stage of every ground literal must be a successor ordinal, as observed in Definition 2.4.

Let us consider first the implication from left to right. For limit ordinals $\alpha > 0$, G is ground successful (respectively, ground failed) at level $\leq \alpha$ implies that G is ground successful (ground failed) at level β for some successor ordinal $\beta < \alpha$. Hence by hypothesis, p $(\neg p)$ is in I_β and hence in I_α by monotonicity.

Conversely, p (respectively $\neg p$) is in I_α implies that p $(\neg p)$ is in I_β for some successor ordinal $\beta < \alpha$. Hence p is ground successful (ground failed) at level $\leq \beta$ by hypothesis, and the result follows since $\beta < \alpha$.

∎

Corollary 4.6: Let p be an atom, G the goal $\leftarrow p$ and α a countable ordinal. Then

- G is ground successful at level $\alpha \iff stage_p = \alpha$
- G is ground failed at level $\alpha \iff stage_{\neg p} = \alpha$

∎

Theorem 4.7: Let $Q = \{p_1, \ldots, p_n, \neg q_1, \ldots, \neg q_m\}$ be an arbitrary ground query, and let G be the goal $\leftarrow Q$.

- G is ground successful $\iff M_{WF}(P) \models Q$

- G is ground failed $\iff M_{WF}(P) \models \neg Q$
- G is ground indeterminate $\iff M_{WF}(P) \not\models Q$ and $M_{WF}(P) \not\models \neg Q$

Proof: G is ground successful if and only if each p_i is ground successful, and each q_j is ground failed, by Lemma 4.1. This happens precisely when each p_i and each $\neg q_j$ is in $M_{WF}(P)$, by Theorem 4.5, i.e., when $M_{WF}(P) \models Q$.

Similarly, G is ground failed if and only if some q_j is ground successful or some p_i is ground failed, again by Lemma 4.1. This happens precisely when some $\neg p_i$ or some q_j is in $M_{WF}(P)$, by Theorem 4.5, i.e., when $M_{WF}(P) \models \neg Q$. ∎

5 Soundness

Now that the correspondence between ground global trees and the well founded semantics has been established, we investigate the correspondence between ground global trees and (general) global trees.

Lemma 5.1: Let P be a program and G a goal. Let L' be an arbitrary active leaf of T_G, with computed most general unifier θ. Then for every ground substitution δ for $G\theta$, $T^g_{G\theta\delta}$ has an active leaf L that is an instance of L'.
Proof: Let $r_1, \ldots, r_n$ be the rules used in the branch ending in L' in T_G, and let $\theta_1, \ldots, \theta_n$ be the corresponding most general unifiers. By definition, $\theta = \theta_1\theta_2 \ldots \theta_n$. We prove the result by induction on the depth, n.

When $n = 0$, G is itself the only active leaf of T_G, with the identity computed most general unifier. $G\delta$ is then the only active leaf of $T^g_{G\delta}$, and is clearly an instance of G.

Suppose the result is true for $n = k$. We show it is true for $n = k + 1$. Let G be $\leftarrow p_1, \ldots, p_l, \neg q_1, \ldots, \neg q_m$, and suppose L' is at depth $k + 1$ in T_G. Suppose that r_1 is the rule $p \leftarrow b_1, \ldots, b_{l'}, \neg c_1, \ldots, \neg c_{m'}$, and that p unifies with p_i using most general unifier θ_1. Then the resultant goal G' is

$$\leftarrow (p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_l, \neg q_1, \ldots, \neg q_m, b_1, \ldots, b_{l'}, \neg c_1, \ldots, \neg c_{m'})\theta_1.$$

(Recall that the order of literals in a goal is unimportant.)

L' is a leaf of $T_{G'}$ at depth k, with computed most general unifier $\theta' = \theta_2 \ldots \theta_{k+1}$. Let γ be a substitution such that $G'\theta'\delta\gamma$ is ground. By our induction hypothesis, there is an active leaf L of $T^g_{G'\theta'\delta\gamma}$ that is an instance of L'. But $G'\theta'\delta\gamma$ is a child of $G\theta\delta$ in $T^g_{G\theta\delta}$ using the rule $r_1\theta\delta\gamma$ which must be ground. Hence $T^g_{G\theta\delta}$ has a leaf L that is an instance of L'. Since δ is arbitrary, the result follows. ∎

We now investigate the converse of Lemma 5.1.

Lemma 5.2: Let P be a program, G a goal, and δ a ground substitution for G. Let L be an arbitrary active leaf of $T^g_{G\delta}$. Then there exists an active leaf L' of T_G, with computed most general unifier θ, such that L is an instance of L' and θ is more general then δ.
Proof: (Sketch)

We may consider the branch to L in $T^g_{G\delta}$ as an *unrestricted derivation* from G, i.e., a derivation in which we do not insist that unifiers be most general. (In the ground SLP-tree, such unifiers always make the resulting goal ground.) The proof of this lemma is then very similar to the proof of the "mgu lemma" in [13], and the details are omitted here. ∎

Lemma 5.3: Let G be a goal. Then

- If G is successful with answer substitution θ, then $G\theta\delta$ is ground successful for every ground substitution δ for $G\theta$.

- If G is failed then $G\delta$ is ground failed for every ground substitution δ for G.

Proof: The proof is by induction on the level of G with respect to θ. Since no goals succeed or fail at limit ordinals (including the base case, 0), the result for limit ordinals is trivial. We now consider the successor ordinal case.

(First part) Suppose that G is successful with respect to θ at level α, and that $L = \{\neg p_1, \ldots, \neg p_m\}$ is the successful leaf with answer substitution θ. Then each p_i must be ground, and each $\leftarrow p_i$ failed at level $\beta_i < \alpha$. By our induction hypothesis, each $\leftarrow p_i$ is ground failed.

Since L is ground, L must appear as a leaf of $T^g_{G\theta\delta}$ for every ground substitution δ, by Lemma 5.1. Finally, since each $\leftarrow p_i$ is ground failed, $G\theta\delta$ is ground successful.

(Second part) Suppose G is failed at level α, and that $L' = \{\neg p_1, \ldots, \neg p_m\}$ is an arbitrary active leaf of T_G. Then some p_i must be ground, and $\leftarrow p_i$ successful at level $\beta_i < \alpha$. By our induction hypothesis, $\leftarrow p_i$ is ground successful. Since L' is arbitrary, every active leaf of T_G contains such a ground subgoal.

Let δ be an arbitrary ground substitution for G. Every active leaf of $T^g_{G\delta}$ is an instance of an active leaf in T_G, by Lemma 5.2. Hence every active leaf of $T^g_{G\delta}$ contains a literal $\neg q$ such that $\leftarrow q$ is ground successful. By definition, $G\delta$ is ground failed. Since δ is arbitrary, the result follows. ∎

We may now prove the soundness of global SLS-resolution.

Theorem 5.4: (Soundness of global SLS-resolution) Let P be a program, and let $G = \leftarrow Q$ be a goal. Then

1. If G is successful with answer substitution θ then $M_{WF}(P) \models \forall(Q\theta)$.

2. If G is failed then $M_{WF}(P) \models \forall(\neg Q)$.

Proof: (First part) Let δ be a ground substitution for $G\theta$. By Lemma 5.3, G is successful with answer substitution θ implies that $G\theta\delta$ is ground successful. By Theorem 4.7, $M_{WF}(P) \models Q\theta\delta$. Since δ is arbitrary, it follows that $M_{WF}(P) \models \forall(Q\theta)$.

(Second part) By Lemma 5.3, G is failed implies that $G\delta$ is ground failed for all ground substitutions δ for G. By Theorem 4.7, $M_{WF}(P) \models \neg Q\delta$ for all such δ, and hence $M_{WF}(P) \models \forall(\neg Q)$. ∎

6 Completeness

In this section we address the completeness of global SLS-resolution. First, though, we remark that our restriction to non-floundering goals in this section does not result in any loss of expressive power. To see this point, assume we have a unary predicate *term* that binds its argument to every term in the Herbrand universe of the program. By adding the atom *term*(X) to the body of each rule for all variables X in the rule, and by adding this atom also to the goal, one can guarantee that floundering will not occur, without changing the well-founded model of the program. Incidentally, this shows that even when restricted to non-floundering literals, the well-founded model is not, in general, recursively enumerable.

Constructing such a predicate *term* is simple: For each constant c appearing in the program include the rule $term(c)$, and for each n-ary function symbol f in the program include the rule

$$term(f(X_1, \ldots, X_n)) \leftarrow term(X_1), \ldots, term(X_n).$$

Lemma 6.1: (Lifting lemma) Let P be a program, G a non-floundering goal and let δ range over all ground substitutions for G. Then

- If for some δ, $G\delta$ is ground successful then G is successful with an answer substitution θ more general than δ.
- If for all δ, $G\delta$ is ground failed, then G is failed.

Proof: The proof is by induction on the level of $G\delta$. (For the second part, the induction is on the *maximum* over all δ of the level of $G\delta$.) Again, since goals cannot succeed at limit ordinals, the limit ordinal case, including the base case 0, is trivial.

(First part) Suppose $G\delta$ is ground successful at level α, and let $L = \{\neg p_1, \ldots, \neg p_n\}$ be a successful leaf in $T_{G\delta}$. By Lemma 5.2, T_G has a leaf $L' = \{\neg p'_1, \ldots, \neg p'_n\}$ such that each p_i is an instance of p'_i, and the computed most general unifier θ at L' is more general than δ.

Since $G\delta$ is ground successful, each $\leftarrow p_i$ is ground failed at strictly lower level. If $p'_i = p_i$ (i.e., p'_i is ground) then p'_i is failed by our induction hypothesis. If p'_j was not ground for some j, then the negation node corresponding to L would have a non-ground child. Since no $\leftarrow p'_i$ can be successful (it must be either failed or floundered), this would contradict our assumption that G is not floundered.

Hence L' is ground and every $\leftarrow p'_i$ is failed. Thus, G is successful with answer substitution θ more general than δ.

(Second part) Let $L' = \{\neg p'_1, \ldots, \neg p'_n\}$ be an arbitrary leaf of T_G, with computed most general unifier θ. Let γ be a ground substitution for $G\theta$, and let $\delta = \theta\gamma$. Then $T^g_{G\delta}$ has an active leaf $L = \{\neg p_1, \ldots, \neg p_n\}$ that is an instance of L', by Lemma 5.1. Since $G\delta$ is ground failed by assumption, some $\leftarrow p_i$ is ground and successful at strictly lower level.

Consider $\leftarrow p'_i$, which is more general than $\leftarrow p_i$. If p'_i is ground, then $p'_i = p_i$ and so $\leftarrow p'_i$ is successful by the induction hypothesis. If p'_i is not ground, then since G is not floundered, some other p'_j is ground and successful. In either case, L' is a failed leaf in T_G. Since our choice of L' was arbitrary, all active leaves of T_G are failed, and hence G is failed. ∎

The restriction to non-floundering goals in the above lemma cannot be omitted due to programs of the form

$$\begin{array}{l} p(x) \leftarrow \neg q(f(x)) \\ q(a) \end{array}$$

for which the goal $\leftarrow p(x)$ flounders, while every ground instance of this goal succeeds. This example indicates that disallowing floundering goals altogether is perhaps too harsh. However, if we add the rule $q(f(a))$ to the program above then we are faced with somehow trying to represent the success set for $\leftarrow p(x)$ as "x may be anything except $f(a)$," a concept that requires a broader notion of answer substitution. Some results in this direction have been presented in [12], and some more recent work has described a process called "constructive negation" in which negative subgoals are used to generate negative bindings [4, 20]. Whether such methods will be useful in practice, or whether ground negation is sufficient for most purposes remains to be seen. Restricting programs and goals to be "allowed" [13], for example, guarantees freedom from floundering.

Before we present our completeness result, we first address what has been termed the "universal query problem" [21]. Certain anomalies occur when working only with Herbrand interpretations. We define the augmented program, first introduced in [31], but discussed implicitly in [14], and show how this allows us to restrict ourselves to Herbrand interpretations.

Example 6.1: Consider the program P given by the single rule

$$p(a)$$

The only Herbrand model of P is $\{p(a)\}$ and so $\forall x\ p(x)$ is true in all Herbrand models, although it is not a logical consequence of the program. However, if we add the apparently unrelated fact $q(b)$ to P, $\forall x\ p(x)$ becomes false in some Herbrand models. Further, no resolution-type procedure will give the identity answer substitution for the query $\{p(x)\}$. As we shall see, $\forall x\ p(x)$ is not true in all Herbrand models of the augmented program P', and so use of the augmented program overcomes such anomalous behaviour. □

Definition 6.1: For any program P we define the *augmented program* P'. Let $\bar{p}$, $\bar{f}$ and $\bar{c}$ be a predicate symbol, function symbol and constant symbol respectively, none of which appear in P. Define $P' = P \ \cup \ \{\bar{p}(\bar{f}(\bar{c}))\}$. □

Our motivation for introducing the augmented program is that it allows us to overcome some anomalies such as in Example 6.1 by assuring that the Herbrand universe contains infinitely many ground terms that do not appear explicitly in the program.

Przymusinski proposes an alternative solution. In proving the soundness and completeness of SLS-resolution with respect to the perfect model semantics, he considers *all* perfect models rather than just perfect Herbrand models [21]. These models have to be models of a set of equality axioms known as Clark's equality axioms [6].

Theorem 6.2: (Completeness of global SLS-resolution) Let P be a program, and $G = \leftarrow Q$ a non-floundering goal involving only symbols from P. Let P' be the augmented version of P. Let ϕ be a substitution for the variables of Q. Then

1. If $M_{WF}(P) \models \exists Q$ then G succeeds
2. If $M_{WF}(P) \models \forall(\neg Q)$ then G is failed
3. If $M_{WF}(P') \models \forall(Q\phi)$ then G succeeds with an answer substitution more general than ϕ.

Proof:

1. If $M_{WF}(P) \models \exists Q$, then $M_{WF}(P) \models Q\delta$ for some ground substitution δ. By Theorem 4.7, $G\delta$ is ground successful. By Lemma 6.1 G is succeeessful.

2. If $M_{WF}(P) \models \forall \neg Q$ then for every ground substitution δ, $M_{WF}(P) \models \neg Q\delta$. Hence, by Theorem 4.7 $G\delta$ is ground failed. By Lemma 6.1 G is failed.

3. (We assume ϕ does not mention the symbols $\bar{f}$ or $\bar{c}$. Extending the proof below in the case that $\bar{f}$ or $\bar{c}$ does appear in ϕ is straightforward.)

   Let $\{x_0, \ldots, x_n\}$ be the variables appearing in $Q\phi$. Let δ be the (ground) substitution $\{x_0|\bar{c}, x_1|\bar{f}(\bar{c}), \ldots x_n|\bar{f}^n(\bar{c})\}$ of terms from the Herbrand universe of the augmented program for the variables in $Q\theta$. Then $M_{WF}(P') \models Q\phi\delta$, and so $G\phi\delta$ is ground successful by Theorem 4.7.

   By Lemma 6.1, G succeeds with an answer substitution θ such that for some substitution γ
   $$\theta\gamma = \phi\delta.$$
   Now θ cannot contain any substitutions involving $\bar{f}$ or $\bar{c}$ since G contains only symbols from P, and the predicate $\bar{p}$ in P' appears nowhere in P. Hence the only occurrences of $\bar{f}$ and $\bar{c}$ in the left side of the equality above are in γ. Let γ' be formed from γ by replacing every occurrence of $\bar{f}^i(\bar{c})$ by the variable x_i. Then $\theta\gamma' = \phi$, and so θ is indeed more general than ϕ.

∎

We cannot substitute P for P' in the third item in Theorem 6.2 as illustrated by Example 6.1. Some texts (for example [13]) make the implicit assumption that extra ground terms exist in order to prove completeness results. The purpose of the augmented program is to formally include sufficiently many such ground terms in the Herbrand universe.

Corollary 6.3: For non-floundering goals $G = \leftarrow Q$,

$$T_G \text{ is indeterminate} \Leftrightarrow \text{neither } M_{WF}(P) \models \exists(Q) \text{ nor } M_{WF}(P) \models \forall(\neg Q). \quad ∎$$

7 Discussion

The well-founded semantics is a declarative semantics that unifies a number of approaches in a robust fashion. In order to be able to *use* well-founded negation in logic programs, a corresponding procedural semantics is necessary. This paper presents such a procedural semantics.

Global SLS-resolution is not effective, as discussed below. However, for certain subclasses of programs, global SLS-resolution is effective. For acyclic programs [3, 2] or, more generally, for modularly acyclic programs [25] global SLS-resolution is effective. In the absence of function symbols, the Herbrand Base is finite, and so effective procedures exist.[5] For function-free programs global SLS-resolution may be made effective by using the memoing techniques of [10, 26] to prune positive loops, and by also pruning negative loops.

Global SLS-resolution (and hence the well-founded semantics) captures the meaning of all well-behaved programs in the sense that every program without *infinite* recursion through

[5] A polynomial time algorithm for constructing the well-founded model for function-free programs, that is bottom-up in nature, is given in [32].

negation is given a semantics in which every ground atom is either true or false. For the perfect model approach, finite recursion through negation is only guaranteed for locally stratified programs.

Furthermore, for programs that do involve infinite recursion through negation, those portions that recurse infinitely through negation are left undefined, while the remainder of the program is given the expected semantics.

There are three sources of non-effectiveness in global SLS-resolution:

1. Infinite branches of an SLP-tree are treated as failed.
2. The SLP-tree for a goal may have an infinite number of branches, (although only a finite branching factor at any particular depth).
3. If a goal is indeterminate, global SLS-resolution will recurse infinitely through negation.

Although a preferential computation rule selects positive literals ahead of negative literals, in practice a sub-process may be spawned to expand a negative subgoal as soon as it becomes ground. Such spawning will not only improve performance but may allow the earlier pruning of long branches.

Note that SLDNF-resolution using a safe computation rule is sound with respect to the well-founded semantics for all programs. However, even with a preferential computation rule, SLDNF-resolution is incomplete as it does not treat infinite branches of an SLP-tree as failed.

I would like to thank Teodor Przymusinski, Rodney Topor, Jeff Ullman and Allen Van Gelder for helpful comments on earlier drafts of this paper. I would also like to thank the three anonymous referees for their suggested improvements to the paper, particularly for pointing out an error in the appendix of [24].

References

[1] K. R. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148, Los Altos, CA, 1988. Morgan Kaufmann.

[2] K.R. Apt and M. Bezem. Acyclic programs. In *Proceedings of the Seventh International Conference on Logic Programming*, 1990.

[3] L. Cavedon. Continuity, consistency, and completeness properties for logic programs. In *Proceedings of the Sixth International Conference on Logic Programming*, 1989.

[4] David Chan. Constructive negation based on the completed database. In *Proc. Fifth International Conference and Symposium on Logic Programming*, 1988.

[5] A. Chandra and D. Harel. Horn clause queries and generalizations. *Journal of Logic Programming*, 2(1):1–15, 1985.

[6] K. L. Clark. Negation as failure. In Gallaire and Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, New York, 1978.

[7] M. Fitting. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 1985.

[8] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. Fifth International Conference and Symposium on Logic Programming*, 1988.

[9] J. Jaffar, J.-L. Lassez, and J. Lloyd. Completeness of the negation-as-failure rule. In *Int'l Joint Conf. on Artificial Intelligence*, pages 500–506, 1983.

[10] D. Kemp and R. Topor. Completeness of a top down query evaluation procedure for stratified databases. In *Proc. Fifth International Conference and Symposium on Logic Programming*, 1988.

[11] K. Kunen. Negation in logic programming. *Journal of Logic Programming*, 4(4):289–308, 1987.

[12] J. L. Lassez and K. Marriott. Explicit representation of terms defined by counter examples. *Journal of Automated Reasoning*, 3:301–317, 1987.

[13] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, New York, 2nd edition, 1987.

[14] M. J. Maher. Equivalences of logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 388–402, Los Altos, CA, 1988. Morgan Kaufmann.

[15] J. McCarthy. Circumscription – a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1):27–39, 1980.

[16] J. Minker. On indefinite databases and the closed world assumption. In *Proc. Sixth Conference on Automated Deduction*, pages 292–308. Springer Verlag, 1982.

[17] Y. N. Moschovakis. *Elementary Induction on Abstract Structures.* North-Holland, New York, 1974.

[18] H. Przymusinska and T. Przymusinski. Weakly perfect model semantics for logic programs. In *Proc. Fifth International Conference and Symposium on Logic Programming*, 1988.

[19] T. Przymusinski. Every logic program has a natural stratification and an iterated fixed point model. In *ACM Symposium on Principles of Database Systems*, 1989.

[20] T. Przymusinski. On constructive negation in logic programming. In *Proceedings, North American Conference on Logic Programming*, 1989.

[21] T. Przymusinski. On the declarative and procedural semantics of logic programs. *Journal of Automated Reasoning*, 5:167–205, 1989.

[22] T. C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216, Los Altos, CA, 1988. Morgan Kaufmann.

[23] K. Ross and R. W. Topor. Inferring negative information from disjunctive databases. *Journal of Automated Reasoning*, 4:397–424, 1988. Also appeared as Melbourne University Technical Report 87/1.

[24] K. A. Ross. A procedural semantics for well-founded negation in logic programs. In *Proceedings of the Eighth ACM Symposium on Principles of Database Systems*, 1989. Also Stanford University Technical Report STAN-CS-88-1233.

[25] K.A. Ross. Modular acyclicity and tail recursion in logic programs. Submitted for publication, 1990.

[26] Hirohisa Seki and Hidenori Itoh. A query evaluation method for stratified programs under the extended cwa. In *Proc. Fifth International Conference and Symposium on Logic Programming*, 1988.

[27] J. C. Shepherdson. Negation as failure, II. *Journal of Logic Programming*, 2(3):185–202, 1985.

[28] J. C. Shepherdson. Negation in logic programming. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 19–88, Los Altos, CA, 1988. Morgan Kaufmann.

[29] M. H. Van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *JACM*, 23(4):733–742, 1976.

[30] A. Van Gelder. Negation as failure using tight derivations for general logic programs. In *Proc. Third IEEE Symposium on Logic Programming*, Salt Lake City, Utah, September 1986. Springer-Verlag. (Preliminary version also appears in *Foundations of Deductive Databases and Logic Programming* (J. Minker, Ed.), Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1988.).

[31] A. Van Gelder, K. A. Ross, and J. S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *Proc. Seventh ACM Symposium on Principles of Database Systems*, 1988. Full version to appear in J.ACM.

[32] A. Van Gelder, K. A. Ross, and J. S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. *JACM*, (to appear). (Full paper).