A GOAL-ORIENTED APPROACH TO COMPUTING THE WELL-FOUNDED SEMANTICS

WEIDONG CHEN* AND DAVID S. WARREN†,‡

Global SLS resolution is an ideal procedural semantics for the well-founded semantics. We present a more effective variant of global SLS resolution, called XOLDTNF resolution, which incorporates simple mechanisms for loop detection and handling. Termination is guaranteed for all programs with the bounded-term-size property. We establish the soundness and (search space) completeness of XOLDTNF resolution. An implementation of XOLDTNF resolution in Prolog is available via FTP.

1. INTRODUCTION

The well-founded semantics [29] provides a natural and robust specification of declarative semantics of arbitrary logic programs. The well-founded partial model coincides with the perfect model of (locally) stratified programs [16] and the smallest three-valued stable model [17]. There remains, however, a challenging problem: how to compute the well-founded semantics more effectively and efficiently.

Several ideal procedural semantics have been developed for the well-founded semantics, including (global) SLS resolution [15, 20]. They cannot be used directly for query evaluation since they may not terminate even for function-free programs. Detection and proper handling of loops (possibly through negation) are indispensable for effective computation of the well-founded semantics.

For definite programs SLD resolution with memoing has been investigated, including extension tables [8], OLDT resolution [25], and QSQR [30]. These

* Supported in part by the National Science Foundation under Grant No. IRI-92-12074.
† Supported in part by NSF Grant CCR 91-02159 and New York State Science and Technology Foundation Grant RDG 90173.
‡ Department of Computer Science, SUNY at Stony Brook, Stony Brook, NY 11794-4400. E-mail: warren@sbsc.sunysb.edu.
Address correspondence to Weidong Chen, Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275-0122. E-mail: wchen@scas.smu.edu.
Received June 1992; accepted May 1993.

THE JOURNAL OF LOGIC PROGRAMMING
©Elsevier Science Publishing Co., Inc., 1993
655 Avenue of the Americas, New York, NY 10010 0743-1066/93/$6.00
methods maintain a table of calls and their corresponding answers. Later occurrences of "similar" calls are resolved using answers instead of program clauses. Memoing not only improves the termination property of SLD resolution, but also provides answer sharing for "similar" calls.

SLD resolution with memoing has been extended to stratified programs, including OLDTNF [24] and OSQR/SLS [12]. The main difference for stratified programs is that more than one table may be maintained at the same time. When a ground negative subgoal \( \sim A \) is selected, a new table is started with respect to which \( A \) is completely evaluated (up to a fixpoint). Obviously, all calls in the new table are completely evaluated when \( A \) is finished. Answers for completely evaluated calls can be shared by "similar" calls later.

A further extension to general programs has been developed, called Well! [4]. Well! extends both OSQR and global SLS resolution, and is effective for nonfloundering function-free programs. The major difference for general programs is that negative loops need to be detected.

This paper presents a more effective variant of (global) SLS resolution, called XOLDTNF resolution. Like Well!, XOLDTNF resolution detects and handles both positive and negative loops. The idea of negative loop detection is rather simple. Each call \( A \) has an associated set of ground negative literals, called negative context in XOLDTNF resolution (or \( \text{Nang} \) in Well!). The negative context is empty for the initial call. During the evaluation of \( A \), suppose that a ground negative literal \( \sim B \) is selected. If \( \sim B \) is not in the negative context of \( A \), \( B \) will be evaluated under the negative context augmented with \( \sim B \). If \( \sim B \) is in the negative context of \( A \), there is a possible negative loop, in which case \( \sim B \) is called a possibly looping negative literal.

There are two major differences between XOLDTNF resolution and Well!. One is in the handling of possibly looping negative literals and answers.

In Well!, computations are separate for lemmas and potential lemmas. They alternate through negation since \( \sim B \) fails if \( B \) does not potentially succeed and \( \sim B \) potentially fails if \( B \) does not succeed. Possibly looping negative literals are treated as failed during the computation of lemmas and as successful during the computation of potential lemmas. This makes it impossible to share results obtained from computation of potential lemmas.

In XOLDTNF resolution, a possibly looping negative literal is immediately replaced with an undefined truth value \( u \) and computation proceeds. Answers for a call \( A \) in XOLDTNF resolution are represented as pairs \((A', u)\), where \( A' \) is an instance of \( A \) and \( u \) is either true or undefined. If \( A \) has two answers that differ only in the truth value, the undefined answer is eliminated. Therefore XOLDTNF resolution provides a uniform representation of both lemmas and potential lemmas and allows more answer sharing.

The other major difference is that XOLDTNF resolution is more general than Well!. Procedural semantics with memoing has to compute lemmas for intermediate subgoals and usually assumes a local computation rule, which selects one of the most recently introduced literals in a query. Well! assumes a local positivistic computation rule and is defined for function-free programs. Due to the different treatment of possibly looping negative literals, XOLDTNF resolution allows an arbitrary local computation rule and is effective for all nonfloundering queries and programs with the bounded-term-size property.
We have also identified a class of finitely negative programs and queries, for which the search space completeness of XOLDTNF resolution is preserved. For programs and queries without this property, a local computation rule is no longer sufficient.

Techniques for effective set-at-a-time query evaluation have been studied in deductive databases [2, 3, 18, 23], which are shown to be essentially equivalent to top-down with memoing [6, 23]. These techniques have been extended to programs without negative loops [1, 19, 22]. The key issue becomes maintaining dependencies among subgoals and ensuring that a positive subgoal be fully evaluated before its negative counterpart can be resolved. The Ordered_Search technique in [19] attempts to maintain the dependency information more efficiently by mirroring more closely the top-down computation.

For general programs, the magic-sets transformation does not always preserve the well-founded semantics [10]. Methods proposed in [10] and [11] to solve this problem tend to make too many magic facts true, which means that more calls are evaluated than necessary. A refinement is developed in [14] that generates fewer magic facts.

Techniques of set-at-a-time query evaluation offer more answer sharing since they often maintain a single pool of lemmas. On the other hand, XOLDTNF resolution uses a simpler mechanism for negation as failure that does not need to maintain dependency information explicitly. The advantage of XOLDTNF resolution is that it can be integrated with Prolog computation in a smooth manner. An implementation of XOLDTNF resolution has been carried out as a Prolog metainterpreter that supports both XOLDTNF and Prolog computation.

The rest of the paper is organized as follows. Section 2 describes the intuitive ideas of loop detection and handling, for both positive and negative loops. Section 3 reviews the definition of global SLS resolution [21] and presents the details of XOLDTNF resolution. Section 4 establishes the soundness and (search space) completeness of XOLDTNF resolution. Section 5 concludes with a brief discussion of implementation details and some issues for future work.

2. LOOP DETECTION AND HANDLING

For effective query evaluation, both positive and negative loops have to be detected and handled properly. This section presents the intuitive ideas for loop detection and handling.

2.1. Positive Loops

Consider the well-known transitive closure program and a small cyclic graph:

\[ tc(X, Y) \leftarrow e(X, Y). \]
\[ tc(X, Y) \leftarrow e(X, Z), tc(Z, Y). \]
\[ e(a, b). \quad e(b, c). \quad e(b, a). \]
The SLD tree for the goal

\[ \leftarrow tc(a, V) \]

is shown in Figure 1, which contains an infinite branch.

To detect positive loops, we follow OLDT resolution [25] and maintain a set of calls that have been encountered, where each call is an atom that has been selected at some node. We consider equivalence classes of atoms equal under variable renaming, that is, atoms that are renaming variants of each other are viewed as syntactically identical.

Conceptually, an OLDT forest is maintained that consists of a tree for each call. Given a goal, where \( A \) is the selected atom, we create a tree for \( A \). Instead of labeling a node with a negative clause, we label it with a definite clause, where the body represents the remaining subgoals to be solved, and the head provides a convenient representation of (partial) answers. The root node of the tree for \( A \) is labeled by \( A - A \). A computation rule selects an atom from the body of a label if possible. If the clause labeling a node in a tree for \( A \) has an empty body, it is called an answer for \( A \).

The selected atom at the root node of a tree is resolved using clauses in a program. For the selected atom \( B \) at a nonroot node, a tree for \( B \) is created if there is currently no tree for \( B \). The selected atom \( B \) at a nonroot node will be resolved using only answers in the tree for \( B \). Figure 2 shows the final OLDT forest derived from the goal \( \leftarrow tc(a, V) \). Notice that due to mutual recursion, a looping positive branch cannot be treated simply as failed. A selected atom at a nonroot node should be resolved using any existing answer, as well as any new answer that may be derived later. (Trees for the extensional predicate \( e/2 \) are not shown.) In practice, it is useful to distinguish between Prolog predicates that will be solved by regular Prolog computation and table predicates that will be solved by OLDT computation. Indeed, one of the advantages of XOLDTNF resolution is its smooth integration with ordinary Prolog computation.
2.2. Negative Loops

XOLDTNF resolution is an extension of OLDT resolution with negative loop detection and handling. Negative loops occur due to recursion through negation. Consider the following program [9, 281 and goal:

\[
\text{win}(X) \leftarrow \text{move}(X, Y), \sim \text{win}(Y). \\
\text{move}(a, b), \text{move}(a, c), \text{move}(b, a).
\]

\[\leftarrow \text{win}(a).\]

Figure 3 shows the SLDNF tree for the goal, which contains an infinite negative branch.

A simple mechanism for negative loop detection is to associate with each call a **negative context**. Consider a branch through negation in an SLDNF tree. The negative context of a call on the branch is the set of ground negative literals encountered along the path from the root to the call. In Figure 3, the initial call \(\text{win}(a)\) has an empty negative context. The negative context for \(\text{win}(b)\) is \(\{\sim \text{win}(b)\}\), and the negative context for the second call of \(\text{win}(a)\) is \(\{\sim \text{win}(b), \sim \text{win}(a)\}\).

---

1. A preliminary report on XOLDTNF resolution appeared in [7].
In the tree for the second call win(a), when \( \sim \text{win}(b) \) is selected, it is in the negative context of win(a), indicating that there is a possible negative loop. Our approach is to treat the selected ground negative literal \( \sim \text{win}(b) \) as undefined. It means that this occurrence of \( \sim \text{win}(b) \) in the tree for the second call of win(a) does not contribute to the success or failure of the second call win(a), or ultimately to the success or failure of the previous occurrence of \( \sim \text{win}(b) \).

In XOLDTNF resolution, an answer consists of not only an instance of a query atom, but also a truth value indicating whether the answer is true or undefined. If there are two answers that differ in only the truth value, the undefined answer should be simplified away. Figure 4 shows the XOLDTNF forest for the goal \( \leftarrow \text{win}(a) \).

Note that each call is of the form \((N, A)\), where \(N\) is a set of ground negative literals and \(A\) is an atom. For the initial call, \(N\) is empty.

In the XOLDTNF tree for \((N, A)\), if a positive literal \(B\) is selected at a nonroot node, it corresponds to a call \((N, B)\). If a ground negative literal \(\sim B\) is selected, \(\sim B\) is treated immediately as undefined if \(\sim B \in N\). This means that \(\sim B\) has occurred previously along the branch through negation in the SLDNF tree. By treating the current occurrence of \(\sim B\) as undefined, we are basically saying that the success or failure of the previous occurrence of \(\sim B\) does not depend upon the current one.

If \(\sim B\) is not in \(N\), a tree for \(B\) is explored, but with a larger negative context, namely, \(N \cup \{\sim B\}\). The association of a negative context with each query atom effectively imposes a stratification ordering over calls such that the larger the negative context of a call, the lower the stratum of the call. Since the negative context for the initial call is empty, every negative subgoal upon which the initial call depends on will be properly solved.
In XOLDTNTN resolution, undefined answers are computed explicitly. An XOLDTNTN tree may be successful, undefined, failed, floundered, or indeterminate if its status cannot be determined.

3. XOLDTNTN RESOLUTION

XOLDTNTN resolution is a more effective variant of global SLS resolution [20, 21]. This section reviews the definition of global SLS resolution [21] and presents the details of XOLDTNTN resolution. We assume the standard terminology of [13].
3.1. Global SLS Resolution

If \( A \) is an atom, \( A \) is a positive literal and \( \sim A \) is a negative literal. A program is a finite set of clauses of the form

\[
A \leftarrow L_1, \ldots, L_n
\]

where \( A \) is an atom, \( L_1, \ldots, L_n \) are literals. A goal is of the form \( \leftarrow L_1, \ldots, L_n \), where \( L_1, \ldots, L_n \) are literals. A computation rule \( R \) is a rule that selects exactly one literal from a goal if possible. \( R \) is positivistic if and only if it selects all positive literals before any negative ones.

Global SLS resolution is defined in terms of SLP-trees and global trees. In SLP-trees, positive literals are solved using program clauses.

**Definition 3.1 (SLP-Trees [21])**. Let \( G \) be a goal of the form \( \leftarrow Q \), and let \( R \) be a positivistic computation rule. The root node of the SLP-tree \( T_G \) for \( G \) is \( G \). Let \( H = + Q' \) be any node of \( T_G \). Its children are obtained as follows:

- If \( Q' \) contains a positive literal, then the literal \( L \) selected by \( R \) from \( H \) 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 \( U_L \) is empty, then \( H \) has no children and is a dead leaf.
- If \( Q' \) is empty or contains only negative subgoals, then \( H \) is an active leaf.

A branch of \( T_G \) is a path from the root of \( T_G \). We associate with each active leaf \( L \) its computed substitution, which is the composition of the most general unifiers used along the branch to \( L \).

The global tree for a goal is an OR/NOR tree whose nodes may be SLP-trees.

**Definition 3.2 (Global Tree [21])**. Let \( \Gamma_G \) denote the global tree for a goal \( G \). The nodes of \( \Gamma_G \) are of three types: negation nodes, tree nodes, and nonground nodes. Tree nodes are SLP-trees for intermediate goals.

The root node of \( \Gamma_G \) is the SLP-tree for the goal \( G \). An internal tree node is a tree node that is not the root. Let \( T \) be any tree node of \( \Gamma_G \). The children of \( T \) are negation nodes, one for each active leaf of \( T \).

Let \( J \) be a negation node, corresponding to an active leaf \( \leftarrow \sim A_1, \ldots, \sim A_n \), where \( n \geq 0 \). \( J \) has \( n \) children, one for each \( \sim A_i \). If \( A_i \) is ground, the child corresponding to \( \sim A_i \) is the tree node \( T_{\sim A_i} \); otherwise the corresponding child is a nonground node. Nonground 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 nonground node is floundered.
2. (a) If some child node of a negation node \( J \) is a successful tree node, then \( J \) 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 \( J \) 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 no children of \(J\) are successful, then \(J\) is floundered.

3. (a) If every child of a tree node \(T\) is a failed negation node or if \(T\) is a leaf of \(\Gamma_G\) (i.e., \(T\) has no active leaves), then \(T\) is failed. The level of \(T\) is \(\alpha + 1\), where \(\alpha\) 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 \(T\) is successful. An internal tree node has level one more than the minimum level of all its successful children. The root tree node may have multiple 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 \(T\) 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.

Let \(L\) be an active leaf of a tree node in \(\Gamma_G\). \(L\) is successful, failed, or floundered if and only if the corresponding negation node is successful, failed, or floundered, respectively. The goal \(G\) is successful, failed, or floundered if and only if \(T_G\) is successful, failed, or floundered, respectively.

A successful branch of \(T_G\) is a branch of \(T_G\) that ends at a successful leaf. An answer substitution for \(G\) is the computed substitution of a successful leaf of \(T_G\).

Let \(P\) be a program, \(G\) be a goal, and \(J\) be a negation (tree) node \(J\) that is successful or failed in the global tree for \(G\). We associate with \(J\) a foundation \(\mathcal{F}(J)\), which is a set of ground negative literals. A foundation of \(J\) represents the set of ground negative literals that are solved by negation as failure in determining that \(J\) is successful or failed. The rules are as follows:

1. (a) If \(J\) is a failed negation node, the foundation of \(J\) is the union of \(\{\neg B\}\) and the foundation of \(T_{\neg B}\), where \(T_{\neg B}\) is a child of \(J\) with the minimum level among all successful children of \(J\).

(b) If \(J\) is a successful negation node, then the foundation of \(J\) is the union of the foundations of all children of \(J\) and the set of ground negative literals in the active leaf corresponding to \(J\).

2. (a) If \(T\) is a failed tree node, the foundation of \(T\) is the union of the foundations of all children of \(T\).

(b) Suppose that \(T_{\neg A}\) is a successful tree node, where \(A\) is an atom. If \(A\) is ground, the foundation of \(T\) is the foundation of a successful child of \(T\) with the minimum level among all the successful children of \(T\). If \(A\) is not ground, we associate a foundation with each successful child of \(T\); the foundation of \(T\) with respect to such a successful child is the same as the foundation of the child.

If the SLP-tree \(T_{\neg A}\), where \(A\) is an atom, is successful or failed, we also denote the foundation of \(T_{\neg A}\) by \(\mathcal{F}(A)\).
Lemma 3.1. Let $A$ be a ground atom, and let the SLP-tree $T_{\neg A}$ be successful or failed at level $\alpha$. Then $\neg A \not\in \mathcal{Y}(A)$ and, for every $\neg B \in \mathcal{Y}(A)$, the SLP-tree $T_{\neg B}$ is successful or failed at a level $\beta < \alpha$.

PROOF. By the definition of the level of a negation (tree) node, it is obvious that for every $\neg B \in \mathcal{Y}(A)$, $T_{\neg B}$ is successful or failed at a level $\beta < \alpha$. It follows that $\neg A \not\in \mathcal{Y}(A)$. □

Theorem 3.2 (Soundness of Global SLS Resolution [21]). Let $P$ be a program, let $M_p$ be the well-founded partial model of $P$, and let $G = \leftarrow Q$ be a goal. Then the following hold:

- If $G$ is successful with answer substitution $\theta$, then $M_p \models \forall(Q\theta)$.
- If $G$ is failed, then $M_p \models \forall(\neg Q)$.

For completeness, each program $P$ has an augmented program $P'$ [21], where $P' = P \cup \{\bar{p}(\bar{f}(\bar{c}))\}$ such that $\bar{p}, \bar{f}, \bar{c}$ are, respectively, predicate symbol, function symbol, and constant symbol that do not appear in $P$.

Theorem 3.3 (Completeness of Global SLS Resolution [21]). Let $P$ be a program, let $M_p$ be the well-founded partial model of $P$, and let $G = \leftarrow Q$ be a nonfloundering goal involving only symbols from $P$. Let $P'$ be an augmented version of $P$, and let $\phi$ be a substitution for the variables in $Q$. Then we have the following:

- If $M_p \models \exists(Q)$, then $G$ succeeds.
- If $M_p \models \forall(\neg Q)$, then $G$ is failed.
- If $M_p \models \forall(Q\phi)$, then $G$ succeeds with an answer substitution more general than $\phi$.

3.2. XOLDTNF Forest

Intuitively, an XOLDTNF forest is obtained by flattening a global tree and SLP-trees into a forest of XOLDTNF trees, one for each call. A call is of the form $(N, A)$, where $A$ is an atom and $N$ represents the set of ground negative literals encountered along the branch in a global tree to the node in which $A$ is selected.

To represent both true and undefined answers, a node in an XOLDTNF tree is labeled by an $X$-clause, of the form

$$(A, v) \leftarrow L_1, \ldots, L_n$$

where $A$ is an atom, $v$ is either true or undefined, and $L_1, \ldots, L_n$ are literals. If $n = 0$, an $X$-clause is also called an answer clause, which is written simply as $(A, v)$. We identify each call (X-clause) by its equivalence class under variable renaming. That is, Calls (X-clauses) that are variants of each other are considered syntactically identical.

In XOLDTNF resolution, a computation rule selects exactly one literal from the body of an $X$-clause. A selected atom at a root node is resolved using program clauses, while one at a nonroot node is resolved using lemmas only.
Definition 3.3 (XOLD Resolution). Let $G$ be an X-clause $(A, v) \leftarrow L_1, \ldots, L_n$, where $n > 0$, and let $L_i$ be the selected atom. Let $D$ be a clause, and let $D'$, of the form $A' \leftarrow L'_1, \ldots, L'_m$, be a variant of $D$ with variables renamed so that $G$ and $D'$ have no variables in common. $G$ is XOLD resolvable with $D$ if $L_i$ and $A'$ are unifiable. The clause

\[( (A, v) \leftarrow L_1, \ldots, L_{i-1}, L'_1, \ldots, L'_m, L_{i+1}, \ldots, L_n ) \theta \]

is the XOLD resolvent of $G$ with $D$, where $\theta$ is the most general unifier of $L_i$ and $A'$.

Definition 3.4 (XOLD Answer Resolution). Let $G$ be an X-clause $(A, v) \leftarrow L_1, \ldots, L_n$, where $n > 0$, and let $L_i$ be the selected atom of $G$. Let $B'$ be a variant of an atom $B$ with variables renamed so that $G$ and $B'$ have no variables in common. $G$ is XOLD answer-resolvable with an answer clause $(B, v')$ if $L_i$ and $B'$ are unifiable. The clause

\[( (A, v^*) \leftarrow L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n ) \theta \]

is the XOLD answer resolvent of $G$ with $(B, v')$, where $\theta$ is the most general unifier of $L_i$ and $B'$, and $v^*$ is $t$ if both $v$ and $v'$ are $t$ and is $u$ otherwise.

Since a selected atom may be resolved using an answer clause, our definitions of XOLDTNF forest and the status of nodes are mutually recursive. Recall that for negative loop checking, each call is of the form $(N, A)$, where $N$ is a set of ground negative literals, and $A$ is an atom.

Definition 3.5 (XOLDTNF Forest). Let $P$ be a program, let $R$ be an arbitrary but fixed computation rule, and let $Q$ be a set of atoms. The XOLDTNF forest $\mathcal{T}_Q$ is constructed as follows. Initially, $\mathcal{T}_Q$ contains one XOLDTNF tree for each call $\{(\), A\}$, where $A \in Q$.

Let $\mathcal{T}_{(N, A)}$ be an XOLDTNF tree for a call $(N, A)$. The root of $\mathcal{T}_{(N, A)}$ is an X-clause $G = (A, t) \leftarrow A$. For each clause $D$ in $P$, with which $G$ is XOLD resolvable, the root has one child that is the XOLD resolvent of $G$ with $D$. If there is no such clause in $P$, then $G$ is a failed leaf.

Let $H = (B, v) \leftarrow L_1, \ldots, L_n$ be a nonroot node in an XOLDTNF tree $\mathcal{T}_{(N, A)}$. If $n = 0$, $H$ is an answer leaf, in which case $H$ is successful if $v$ is $t$ and undefined if $v$ is $u$. Otherwise, let $L_i$ be the selected literal.

1. (a) If $L_i$ is an atom and $\mathcal{T}_Q$ currently does not contain a tree $\mathcal{T}_{(N, L_i)}$, then add the tree $\mathcal{T}_{(N, L_i)}$ to $\mathcal{T}_Q$, whose root node is labeled $(L_i, t) \leftarrow L_i$.

(b) If $L_i$ is an atom and there is an answer node $(B', v')$ in the XOLDTNF tree $\mathcal{T}_{(N, L_i)}$ and there is no edge from $H$ that is labeled with $(B', v')$, then $H$ has a child that is the XOLD answer resolvent of $H$ with $(B', v')$. The edge from $H$ to the new child is labeled with $(B', v')$.

2. (a) If $L_i$ is a nonground negative literal, then $H$ is a floundered leaf.

(b) If $L_i$ is a ground negative literal $\sim B$ and $\sim B \in N$, the $H$ has exactly one child $H'$, of the form

\[( B, u ) \leftarrow L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n \]
(c) If \( L_i \) is a ground negative literal \( \sim B \) and \( \sim B \notin N \) and \( \mathcal{T}_Q \) currently does not contain a tree \( \mathcal{T}_{(N \cup \sim B, B)} \) then add the tree \( \mathcal{T}_{(N \cup \sim B, B)} \) to \( \mathcal{T}_Q \), whose root node is labeled \((B, t) \leftarrow B\).

(d) Suppose that \( L_i \) is a ground negative literal \( \sim B \), \( \sim B \notin N \), and \( \mathcal{T}_Q \) contains the tree \( \mathcal{T}_{(N \cup \sim B, B)} \).

(i) If \( \mathcal{T}_{(N \cup \sim B, B)} \) is successful, \( H \) is a failed leaf.

(ii) If \( \mathcal{T}_{(N \cup \sim B, B)} \) is undefined, \( H \) has exactly one child \( H' \), of the form

\[
(B, u) \leftarrow L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n
\]

(iii) If \( \mathcal{T}_{(N \cup \sim B, B)} \) is failed, \( H \) has exactly one child \( H' \), of the form

\[
(B, v) \leftarrow L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n
\]

Let \( N \) be a set of negative literals, and let \( \mathcal{C} \) be a set of calls with negative context \( N \). \( \mathcal{C} \) is completed if, for every \((N, A) \in \mathcal{C}\), there is an XOLDTNF tree \( \mathcal{T}_{(N, A)} \) in \( \mathcal{G} \) such that either

- \( A \) is ground and \( \mathcal{T}_{(N, A)} \) has a successful leaf \((A, t)\) or
- for every selected literal \( L \) at a nonroot node \( H \) in \( \mathcal{T}_{(N, A)} \) in \( \mathcal{G} \),
  - if \( L \) is an atom, then \((N, L) \in \mathcal{C}\) and for every answer node \((L', v')\) in \( \mathcal{T}_{(N, L)} \), \( H \) has a child node that is the XOLD answer resolvent of \( H \) with \((L', v')\), and the edge from \( H \) to the child is labeled \((L', v')\);
  - if \( L \) is of the form \( \sim B \), then \( B \) is ground and either \( \sim B \in N \) or \((N \cup \sim B, B) \) is completed.

Let \( N \) be a negative context and \( S \) be a set of atoms. We define \( \text{depend}_N(S) \) to be the union of \( S \) and the set of atoms selected in XOLDTNF tree \( \mathcal{T}_{(N, B)} \) for all \( B \in S \). We denote by \( \text{depend}_N^-(S) \) the least closure of \( S \) under \( \text{depend}_N \).

A branch of \( \mathcal{T}_{(N, A)} \) is a path from the root of \( \mathcal{T}_{(N, A)} \) to some leaf node. We associate with each branch and each XOLDTNF tree a status (either successful, failed, undefined, floundered, or indeterminate) according to the following rules. We associate a level with true answer clauses, ground negative literals, branches, and trees that are successful or failed.

1. The level of an answer clause \((A', t)\) in an XOLDTNF tree \( \mathcal{T}_{(N, A)} \) is the minimum level of all branches in \( \mathcal{T}_{(N, A)} \) whose leaf nodes are \((A', t)\).
2. Let \( \sim B \) be a ground negative literal selected from a node in an XOLDTNF tree \( \mathcal{T}_{(N, A)} \). If \( \sim B \in N \), the level of \( \sim B \) in \( \mathcal{T}_{(N, A)} \) is 0; if \( \sim B \notin N \) and \( \mathcal{T}_{(N \cup \sim B, B)} \) is either successful or failed, then the level of \( \sim B \) in \( \mathcal{T}_{(N, A)} \) is the level of the XOLDTNF tree \( \mathcal{T}_{(N \cup \sim B, B)} \).
3. (a) A branch is successful (undefined, failed, floundered) if and only if its leaf node is successful (undefined, failed, floundered).
(b) The level of a successful branch is the least ordinal upper bound of the levels of answer clauses used in ground negative literals selected along the branch.
(c) The level of a failed branch is the level of the ground negative literal selected at the last step. The level is 0 if an atom is selected at the last step.
4. An XOLDTNF tree $T_{(N,A)}$ is **failed** if and only if $(N,A)$ is completed, and there is no answer leaf in $T_{(N,A)}$. The level of $T_{(N,A)}$ is $\alpha + 1$, where $\alpha$ is the least ordinal upper bound of the levels of all failed branches in $T_{(N,B)}$ for all $B \in \text{depend}_N^*(\{A\})$.

(b) An XOLDTNF tree $T_{(N,A)}$ is **successful** if and only if it has a successful leaf. $T_{(N,A)}$ may have several associated levels, one for each successful leaf; the level of $T_{(N,A)}$ with respect to such a successful leaf is one more than the level of the answer clause.

(c) An XOLDTNF tree $T_{(N,A)}$ is **floundered** if and only if it has a floundered leaf. $T_{(N,A)}$ is **undefined** if and only if $(N,A)$ is completed and $T_{(N,A)}$ has an undefined leaf, but no successful leaf.

**XOLDTNF resolution** is the top-down process of constructing the XOLDTNF forest $F_Q$. $Q$ is **floundered** if and only if XOLDTNF tree in $F_Q$ is floundered.

XOLDTNF resolution is more effective than global SLS resolution in two aspects. First, an SLP-tree in global SLS resolution is flattened into a forest of XOLDTNF trees, one for each distinct call. Atoms selected from nonroot nodes are resolved using only answer clauses that have been computed or may be computed later. This avoids positive loops.

Second, indeterminate branches in a global tree of global SLS resolution with repeated negative literals are turned into finite ones by replacing later occurrences of negative literals with an undefined truth value $u$. When a ground negative literal $\sim B$ is selected, it is immediately replaced with $u$ if $\sim B$ is in the current negative context. Otherwise, an XOLDTNF tree for $B$ is started, but with a larger negative context, namely, the current negative context augmented with $\sim B$. By associating with each call a negative context, we separate calls into different strata such that the larger the negative context of a call, the lower stratum the call has.

The level of an XOLDTNF tree corresponds to the level of SLP-tree in global SLS resolution, while the level of a branch corresponds to the level of a negative node in global SLS resolution. In XOLDTNF resolution, the level of a failed branch is determined by the level of the first selected ground negative literal that is failed. In global SLS resolution, the level of a failed negation node is the minimum level of all its successful children.

### 4. CORRECTNESS OF XOLDTNF RESOLUTION

This section establishes the soundness, completeness, and termination properties of XOLDTNF resolution. Instead of proving the correctness of XOLDTNF resolution directly with respect to the well-founded semantics, we show that XOLDTNF resolution computes answers that are derived in global SLS resolution. First we need to resolve the difference between computation rules in global SLS resolution and XOLDTNF resolution.

#### 4.1 Computation Rule

Let $P$ be a program and $A$ be an atom. An SLP-tree for $\leftarrow A$ can be constructed in which a positivistic computation rule is used that selects all positive literals
before negative ones. If we ignore negative literals, \( P \) becomes a definite program
and the construction of an SLP-tree reduces to the construction of an SLD-tree. The following theorem shows the independence of computation rule.

**Theorem 4.1 (Independence of Computation Rule [13]).** Let \( P \) be a (general) program, \( G \) a goal, and \( R \) a positivistic computation rule. Suppose that there is an active leaf in the SLP-tree for \( G \) via \( R \) with a computed substitution \( \theta \). Let \( R' \) be any positivistic computation rule. Then there exists an active leaf in the SLP-tree for \( G \) via \( R' \) with a computed substitution \( \theta' \) and \( G\theta' \) is a variant of \( G\theta \).

We are interested in positivistic computation rules that are local. A computation rule is local if it always selects one of the most recently introduced literals [31].

**Definition 4.1.** A positivistic computation rule \( R \) is local if it satisfies the following property. Let \( G = \leftarrow L_1, ..., L_{i-1}, L_i, ..., L_n \) be an arbitrary goal, and \( L_i \) be an atom selected by \( R \) from \( G \). Then for any goal \( G' \) of the form

\[
\leftarrow (L_1, ..., L_{i-1}, L'_1, ..., L'_k, L_{i+1}, ..., L_n) \theta
\]

obtained by resolving \( G \) with some variant \( H \leftarrow L'_1, ..., L'_k \) of a clause in \( P \) on \( L_i \), with a most general unifier \( \theta \), all atoms in \((L'_1, ..., L'_k)\theta \) will be selected by \( R \) before the remaining atoms in \( G' \).

Let \( R_x \) be any computation rule in XOLDTNF resolution, which selects a literal from the body of an X-clause. There is a corresponding local positivistic rule \( R_p \), that satisfies the following property. For any goal \( G' \) of the form

\[
\leftarrow (L_1, ..., L_{i-1}, L'_1, ..., L'_k, L_{i+1}, ..., L_n) \theta
\]

obtained by resolving \( G \) with some variant \( H \leftarrow L'_1, ..., L'_k \) of a clause in \( P \) on \( L_i \), with a most general unifier \( \theta \), the order in which atoms in \((L'_1, ..., L'_k)\theta \) are selected by \( R_p \) from \( G' \) is exactly the same as the order in which atoms are selected by \( R_x \) from an X-clause whose body is of the form \((L'_1, ..., L'_k)\theta \).

Informally, XOLDTNF resolution can be viewed as flattening SLP-trees in global SLS resolution into a forest of trees. The same idea has been explored in OLDT resolution [25] and QSQR [30], where proof segments in an SLD-tree are identified from which lemmas for intermediate subgoals are extracted. Each proof segment in an SLD-tree can be replaced by one step of lemma resolution, and vice versa. This idea can be extended to general programs as far as local positivistic computation rules are used and negative literals are ignored.

**Lemma 4.2.** Let \( P \) be a program, let \( R_x \) be any computation rule in XOLDTNF resolution, let \( R_p \) be the corresponding local positivistic computation rule of \( R_x \), let \( N \) be a negative context, and let \( A \) be an atom.

(a) If \((B, v)\) is an answer leaf in an XOLDTNF tree \( T^{(N,A)} \), then there is a corresponding active leaf in the SLP-tree for \( \leftarrow A \) via \( R_p \). The active leaf contains only ground negative literals and has a computed substitution \( \theta \), such that \( B \) and \( A\theta \) are variants of each other.
(b) The converse holds if for every negative literal \( \sim A' \) in the active leaf, \( A' \) is ground and either
\[ \sim A' \notin N \]
or
\[ \sim A' \notin N \text{ and } T(N \cup \{ \sim A' \}, A') \text{ is either failed or undefined.} \]

**Proof.** We define the *depth* of an answer clause as the minimum of the depths of branches that end with the answer clause, where the *depth* of a branch is 1 plus the sum of the depths of answer clauses that are used in XOLD answer resolution on the branch.

The proof of (a) is based on an induction on the depth of an answer clause. Let \((B, \nu)\) be an answer clause in \(T(N, A)\) with depth \(d\). Then there must be a renaming variant
\[ H < L_1, \ldots, L_n \]
of some clause in \(P\) such that \(A\) and \(H\) have a most general unifier \(\delta\). The root of \(T(N, A)\) has a child
\[ ((A, t) \leftarrow L_1, \ldots, L_n) \delta \]
In the SLP-tree for \( \leftarrow A\), the root has a child
\[ \leftarrow (L_1, \ldots, L_n) \delta \]
If all \( L_i\)'s are negative, they must all be ground and \( B \) is \( A \delta\). (a) holds for this basis case \(d = 1\).

Otherwise, let \(L_i \delta\) be the selected atom by \( R_e\), and let \((B_i, \nu_i)\) be an answer clause in the XOLDTNF tree \(T(N, L_i \delta)\) that is used for resolving \(L_i \delta\). \((B_i, \nu_i)\) has a depth that is less than \(d\). By inductive hypothesis, the SLP-tree for \( \leftarrow L_i \delta\) has an active leaf with a computed substitution \(\theta_i\) such that \(B_i\) and \(L_i \delta \theta_i\) are variants of each other. By XOLD answer resolution, a new node is derived in the XOLDTNF tree \(T(N, A)\):
\[ ((A, t) \leftarrow L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n) \delta \theta_i \]
Let
\[ \leftarrow L_i \delta, \leftarrow Q_1, \ldots, \leftarrow Q_m \]
be the branch for the corresponding active leaf in the SLP-tree for \( \leftarrow L_i \delta\), and let \(\xi_1, \ldots, \xi_m\) be the corresponding sequence of substitutions. Then \(\theta_i = \xi_1 \cdots \xi_m\) and \(Q_m\) contains only ground negative literals.

Since \(R_p\) is the corresponding local positivistic rule of \(R_e\), \(L_i \delta\) will also be selected first in the SLP-tree for \( \leftarrow A\). The corresponding branch in the SLP-tree for \( \leftarrow A\) can be expanded to
\[ \leftarrow A \]
\[ \leftarrow (L_1, \ldots, L_n) \delta \]
\[ \leftarrow (L_1, \ldots, L_{i-1}) \delta \xi_1, Q_1, (L_{i+1}, \ldots, L_n) \delta \xi_1 \]
\[ \vdots \]
\[ \leftarrow (L_1, \ldots, L_{i-1}) \delta \xi_1 \cdots \xi_m, Q_m, (L_{i+1}, \ldots, L_n) \delta \xi_1 \cdots \xi_m \]
Since $\theta_i = \xi_1 \cdots \xi_m$ and $Q_m$ is ground, the last goal is

$$(L_1, \ldots, L_{i-1}) \delta \theta_i, Q_m, (L_{i+1}, \ldots, L_n) \delta \theta_i$$

By repeatedly expanding each step of XOLD answer resolution, we obtain a branch in the SLP-tree for $\leftarrow A$ that ends with an active leaf, whose computed substitution is the composition $\delta \theta_i \cdots \delta \theta_1$, and $B = A \delta \theta_i \cdots \delta \theta_1$, where $L_i$ is the last selected atom.

For (b), the derivation of an answer clause is the reverse of the above construction of a branch in the SLP-tree for $\leftarrow A$, assuming that for every $\sim A'$ in the active leaf, either $\sim A' \in N$, or $\sim A' \notin N$ and $T_{(N \cup \{ \sim A' \}, A')}$ is either failed or undefined. The assumption implies that every negative literal that will be selected in XOLDTNF resolution can be deleted or replaced with an undefined truth value $u$. Therefore all the relevant atoms that are selected by $R$, in global SLS resolution will be selected by $R$, in XOLDTNF resolution in the same order, as long as negative literals are ignored. □

4.2. Soundness

Theorem 4.3 (Soundness of XOLDTNF Resolution). Let $P$ be a program, and let $Q$ be a set of atoms. For any call $(N, A)$ such that $T_Q$ contains the XOLDTNF tree $T_{(N, A)}$,

(a) if the XOLDTNF tree $T_{(N, A)}$ is successful with a true answer clause $(A', \delta)$, then the SLP-tree $T_{\leftarrow A}$ is successful with an answer substitution $\delta$ such that $A'$ and $A\delta$ are variants of each other;

(b) if the XOLDTNF tree $T_{(N, A)}$ is failed, then the SLP-tree $T_{\leftarrow A}$ is failed.

Proof. The proof is by induction on the level(s) of $T_{(N, A)}$. Since no XOLDTNF trees succeed or fail at limit ordinals (including the base case, 0), we need to prove only the case of successor ordinals.

Suppose that $T_{(N, A)}$ is successful with a true answer clause $(A', \delta)$. Then $T_{(N, A)}$ has a level $\alpha + 1$, where $\alpha$ is the level of $(A', \delta)$. By Lemma 4.2, there exists an active leaf in the SLP-tree $T_{\leftarrow A}$ with a computed substitution $\delta$ such that $A'$ and $A\delta$ are variants of each other, and all negative literals in the corresponding active leaf are ground. By the assumption of answer clause $(A', \delta)$, for every $\sim B$ in the active leaf, $\sim B \notin N$ and the XOLDTNF tree $T_{(N \cup \{ \sim B \}, B)}$ is failed, at a level less than or equal to $\alpha$. By inductive hypothesis, the SLP-tree $T_{\leftarrow B}$ is failed. Therefore the SLP-tree $T_{\leftarrow A}$ is successful.

Suppose that $T_{(N, A)}$ is failed. The level of $T_{(N, A)}$ is 1 plus the least ordinal upper bound of the levels of all failed branches in $T_{(N, B)}$ for all $B \in \text{depend}_N^T((A))$. Consider any branch in the SLP-tree $T_{\leftarrow A}$ that ends with an active leaf, with a computed substitution $\delta$. It can be flattened into a sequence of branches in XOLDTNF trees with negative context $N$, one for each selected atom. The set of negative literals in the active leaf is also distributed to all the branches, depending upon the selected atom through which they are introduced.

Since $T_{(N, A)}$ is failed, there is a set $E$ of calls that is completed, one of which is $(N, A$). By definition, every negative literal $\sim B$ in the active leaf is ground, and either $\sim B \in N$, or $\sim B \notin N$ and $(N \cup \{ \sim B \}, B)$ is completed. Thus $(N \cup \{ \sim B \}, B)$
is either successful, undefined, or failed. There exists at least one $B \in N$ such that
$\sim B \in N$ and $\mathcal{T}_{(N \cup \{ \sim B \}, B)}$ is successful. (Otherwise, $\mathcal{T}_{(N, A)}$ will have an answer
leaf, contradictory with the assumption that $\mathcal{T}_{(N, A)}$ is failed.) By definition, the
level of $\mathcal{T}_{(N \cup \{ \sim B \}, B)}$ is less than the level of $\mathcal{T}_{(N, A)}$. By inductive hypothesis, the
SLP-tree $T_{\sim B}$ is successful, and so the branch for the corresponding active leaf is
failed. Since the active leaf is arbitrary, the SLP-tree $T_{\sim A}$ is failed. □

4.3. Completeness

XOLDTNF resolution is not search space complete due to the local nature of its
computation rule. Consider the following program and goal:

$$p(X) \leftarrow \sim p(f(X)).$$

$$q \leftarrow p(a), r.$$

$$\leftarrow q.$$

In the clause for $q$, both body literals are positive. Suppose that $p(a)$ is selected
first. XOLDTNF resolution will start constructing an infinite number of XOLDTNF
trees (see Figure 5). On the other hand, a positivistic rule in global SLS resolution
is able to select $r$ immediately after $p(a)$, before the negative literals that are
introduced by $p(a)$. Nevertheless, completeness can still be achieved by XOLDTNF
resolution for quite a large class of programs and goals, including properly all
function-free programs.

Definition 4.2. A program is finitely negative if there is a function $f(n)$ and a
(computable) computation rule $R$ such that whenever a finite set $Q$ of atoms has
no atom whose argument sizes exceed $n$, the cardinality of $N$ for every
XOLDTNF tree $\mathcal{T}_{(N, A)}$ in the XOLDTNF forest $\mathcal{T}_Q$ is less than or equal to $f(n)$.

Lemma 4.4. Let $P$ be a finitely negative program and let $Q$ be a nonfloundering set of
atoms. Then every XOLDTNF tree in $\mathcal{T}_Q$ is either successful, failed, or undefined,
but not indeterminate.

![FIGURE 5. XOLDTNF forest for q.](image-url)
PROOF. Let the maximum size of arguments in $Q$ be $n$. Then $f(n)$ is an upper bound of $|N|$ for all XOLDTNF tree $\mathcal{T}_{(N, A)}$ for some atom $A$ in $\mathcal{T}_Q$. Let $k$ be the least upper bound of all the negative contexts in $\mathcal{T}_Q$.

Consider any negative context $N$ such that $|N| = k$. Let $\mathcal{E}$ be the set of all calls $(N, A)$ in $\mathcal{T}_Q$. $\mathcal{E}$ must be completed since every selected negative literal must be ground and in $N$. The cases for $N$, where $|N| < k$, follow by a similar analysis. □

We show that XOLDTNF resolution is ideally complete for any finitely negative and nonfloundering set of atoms. That is, XOLDTNF resolution computes all answers that can be derived in global SLS resolution for those atoms. The key difference is that XOLDTNF resolution cuts off some infinite negative branches by replacing selected ground negative literals with undefined $\mathfrak{u}$ if they are in the current negative context.

Theorem 4.5 (Completeness of XOLDTNF Resolution). Let $P$ be a program, let $Q$ be a finitely negative and nonfloundering set of atoms, and let $A$ be an atom.

- Suppose that the SLP-tree node $T_{\neg A}$ is successful with an answer substitution $\theta$. Recall that $\mathcal{Y}(T_{\neg A})$ is the foundation of $T_{\neg A}$. Then for every call $(N, A)$ in $\mathcal{T}_Q$, where $N \cap \mathcal{Y}(T_{\neg A}) = \emptyset$, the XOLDTNF tree $\mathcal{T}_{(N, A)}$ has an answer clause $(A', t)$ such that $A'$ and $A\theta$ are variants of each other.

- If the SLP-tree node $T_{\neg A}$ is failed, then for every call $(N, A)$ in $\mathcal{T}_Q$ such that $N \cap \mathcal{Y}(T_{\neg A}) = \emptyset$, the XOLDTNF tree $\mathcal{T}_{(N, A)}$ is also failed.

PROOF. The proof is based upon induction on the level of the SLP-tree $T_{\neg A}$. Since $T_{\neg A}$ is successful or failed only at a successor ordinal, the cases for limit ordinals are trivial.

Suppose that $T_{\neg A}$ is successful with an answer substitution $\theta$ at level $\alpha$. Let $\leftarrow \sim A_1, \ldots, \sim A_m$ be the corresponding active leaf. Then each $A_i$ must be ground and the SLP-tree $T_{\neg A_i}$ is failed at level $\beta < \alpha$. By definition,

$$\mathcal{Y}(T_{\neg A}) = \{\sim A_1, \ldots, \sim A_m\} \cup \mathcal{Y}(T_{\neg A_1}) \cup \cdots \cup \mathcal{Y}(T_{\neg A_m}).$$

By Lemma 3.1, $\sim A_i \not\in \mathcal{Y}(T_{\neg A_i})$ for every $i (1 \leq i \leq m)$.

Let $\mathcal{T}_{(N, A)}$ be XOLDTNF tree in $\mathcal{T}_Q$ such that $N \cap \mathcal{Y}(T_{\neg A}) = \emptyset$. Then $(N \cup \{\sim A_i\}) \cap \mathcal{Y}(T_{\neg A}) = \emptyset$. By inductive hypothesis, the XOLDTNF tree $\mathcal{T}_{(N \cup \{\sim A_i\}, A_i)}$ is failed for every $i (1 \leq i \leq m)$. By Lemma 4.1 and Lemma 4.2, $\mathcal{T}_{(N, A)}$ has an answer clause $(A', t)$ such that $A'$ and $A\theta$ are variants of each other.

Suppose that $T_{\neg A}$ is failed at level $\alpha$. Then every child $J$ of $T_{\neg A}$ is a failed negation node. Let $\leftarrow \sim A_1, \ldots, \sim A_m$ be the corresponding active leaf of $J$. Then for some $A_i$, the tree node $T_{\neg A_i}$, as a child of $J$, is successful at level $\beta < \alpha$. Let $T_{\neg A_i}$ be such a node with a minimum $\beta$. Then $\mathcal{Y}(J) = \{\sim A_i\} \cup \mathcal{Y}(T_{\neg A_i}) \subset \mathcal{Y}(T_{\neg A}).$

Consider an XOLDTNF tree $\mathcal{T}_{(N, A)}$ in $\mathcal{T}_Q$ such that $N \cap \mathcal{Y}(T_{\neg A}) = \emptyset$. Then $(N \cup \{\sim A_i\}) \cap \mathcal{Y}(T_{\neg A}) = \emptyset$. By inductive hypothesis, $\mathcal{T}_{(N \cup \{\sim A_i\}, A_i)}$ is successful. Since $Q$ is finitely negative and nonfloundering, $\mathcal{T}_{(N, A)}$ must be successful, failed, or undefined.

Suppose that $\mathcal{T}_{(N, A)}$ is not failed. Then it is either successful or undefined and has an answer clause $(A', t)$. By Lemma 4.2, the SLP-tree $T_{\neg A}$ has an active leaf
with a computed substitution \( \theta \) such that \( A \theta \) and \( A' \) are variants of each other. Let the active leaf be \( \sim A_1, \ldots, \sim A_m \). By the assumption of answer clause \((A', v), \mathcal{F}_{N \cup \{ \sim A_j \}, A_i} \) is either undefined or failed for every \( j \) \((1 \leq j \leq m)\), a contradiction with the fact that \( \mathcal{F}_{N \cup \{ \sim A_i \}, A_i} \) is successful for some \( i \). \( \Box \)

4.4. Termination

XOLDTNF resolution terminates for all function-free programs or, more generally, all programs with the bounded-term-size property [27]. The following definition is adopted from [27].

**Definition 4.3 (Bounded-Term-Size Property).** The size of a term is defined recursively as follows:

- The size of a variable or a constant is 1.
- The size of a compound term \( f(t_1, \ldots, t_n) \) is 1 plus the sum of the sizes of its arguments.

A program has the *bounded-term-size property* if there is a function \( f(n) \) and a (computable) computation rule \( R \) such that whenever a finite set \( Q \) of atoms has no atom whose argument sizes exceed \( n \), no atom in \( \mathcal{F}_Q \) has an argument whose size exceeds \( f(n) \).

**Lemma 4.4 (Termination).** Let \( P \) be a program with the bounded-term-size property, and let \( Q \) be a finite set of atoms. Then \( \mathcal{F}_Q \) can be constructed in a finite number of steps.

**Proof.** Let \( n \) be the maximum size of arguments of atoms in \( Q \). By definition, no atom in \( \mathcal{F}_Q \) has arguments whose sizes exceed \( f(n) \). Therefore the number of distinct negative contexts, the number of distinct atoms (that are not variants of each other), and the number of answer clauses are all finite. Thus there are a finite number of XOLDTNF trees in \( \mathcal{F}_Q \). Each XOLDTNF tree is finite since the height of a tree is bounded by the maximum number of literals in a clause in \( P \) and each node has a finite number of children. \( \Box \)

5. DISCUSSION

We have presented a more effective variant of global SLS resolution, called XOLDTNF resolution. It incorporates simple mechanisms for both positive and negative loop detection and handling. XOLDTNF resolution is sound and search space complete for all finitely negative and nonfloundering queries. XOLDTNF resolution always terminates for programs with the bounded-term-size property.

For definite programs, XOLDTNF resolution reduces to OLDT resolution [25], which is essentially equivalent to magic sets computation [2]. For stratified programs or even modularly stratified programs [22], negative contexts are not necessary. Then XOLDTNF resolution reduces to OLDTNF resolution [24].

An implementation of XOLDTNF resolution has been carried out as a Prolog metainterpreter [32]. Two calls that are renaming variants of each other are
considered identical. The implementation uses the left-most computation rule and traverses XOLDTNF forests in a depth-first manner. Efficient loop checking techniques have been investigated in [5] and [26]. We use a simple method in which each call is checked against all previous calls that have been made.

Negative subgoals are solved in a straightforward manner. If a selected ground negative subgoal \( \sim B \) is not in the current negative context, an XOLDTNF tree for the corresponding positive subgoal \( B \) is started, but with a fresh new table. The XOLDTNF tree for \( B \) is fully explored (up to a fixpoint) so that \( B \) and all its relevant calls are completely evaluated. If the new table contains any true answer for \( B \), \( \sim B \) is failed; if the new table contains no answer for \( B \), \( \sim B \) succeeds; if the new table contains only undefined answers for \( B \), \( \sim B \) is replaced with an undefined truth value.

The mechanism of completely evaluating \( B \) by computing up to a fixpoint is analogous to the handling of negation in Prolog. It is possible, however, that the same atom \( A \) may be evaluated multiple times in different negative contexts. Our implementation provides sharing of definite answers of calls that have been completely evaluated. All calls in the new table after \( B \) is evaluated up to a fixpoint are known to be completely evaluated. A single global table is maintained that keeps calls that are completely evaluated and that do not have any undefined answers. Answers of these calls can be reused in any negative context.

Evaluating a positive subgoal up to a fixpoint is a simple way of ensuring that the positive subgoal is completely evaluated. As mentioned above, it could lead to redundant evaluation of the same atom in different negative contexts. A different approach is to maintain the dependency information explicitly and to detect completely evaluated calls dynamically according to the dependency information.

Ross [22] developed a method called the QSQR/SLS procedure that computes and checks dependency information explicitly. The QSQR/SLS procedure handles programs without negative loops or infinite negation and is shown to have the same complexity as a bottom-up method called supplementary magic rewriting [22]. Recently Ramakrishnan et al. [19] investigated an extension of supplementary magic templates rewriting, which is a hybrid between a pure breadth-first and pure depth-first search. Their technique, called Ordered_Search, maintains subgoal dependency information and handles programs with left-to-right modularly stratified negation.

Perhaps the most notable feature of the implementation of XOLDTNF resolution is its simplicity and its relatively smooth integration with Prolog computation. Predicates that are evaluated using XOLDTNF resolution can call Prolog predicates, and vice versa. This has an important practical advantage for applications that may require both traditional Prolog computation and termination properties of XOLDTNF resolution. The XOLDTNF system is available by anonymous FTP from cs.sunysb.edu. An interesting topic for future work is to retain the advantages of XOLDTNF resolution and to avoid redundant evaluation of the same atom in different negative contexts.

The authors are indebted to Kenneth Ross for his work on global SLS resolution. The proofs of XOLDTNF resolution have been simplified using results of global SLS resolution. Detailed comments by the referees were very helpful in improving the content and the presentation of the paper.
REFERENCES


20. Ross, K. A., A Procedural Semantics for Well Founded Negation in Logic Programs, in:


