Tabled evaluation with delaying for general logic programs

Article *in* Journal of the ACM · January 1996

DOI: 10.1145/227595.227597 · Source: DBLP

Tabled Evaluation with Delaying for General Logic Programs [*]

Weidong Chen[†]
Computer Science and Engineering
Southern Methodist University
Dallas, TX 75275-0122

David S. Warren[‡]
Department of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794-4400

June 14, 1995

Abstract

SLD resolution with negation as finite failure (SLDNF) reflects the procedural interpretation of predicate calculus as a programming language and forms the computational basis for Prolog systems. Despite its advantages for stack-based memory management, SLDNF is often not appropriate for query evaluation for three reasons: a) it may not terminate due to infinite positive recursion; b) it may not terminate due to infinite recursion through negation; and c) it may repeatedly evaluate the same literal in a rule body, leading to unacceptable performance.

We address all three problems for goal-oriented query evaluation of general logic programs by presenting tabled evaluation with delaying, called *SLG resolution*. It has three distinctive features:

(i) SLG resolution is a partial deduction procedure, consisting of seven fundamental transformations. A query is transformed step by step into a set of answers. The use of transformations separates logical issues of query evaluation from procedural ones. SLG allows an arbitrary computation rule for selecting a literal from a rule body and an arbitrary control strategy for selecting transformations to apply.

(ii) SLG resolution is sound and search space complete with respect to the well-founded partial model for all non-floundering queries, and preserves all three-valued stable models. To evaluate a query under different three-valued stable models, SLG resolution can be enhanced by further processing of the answers of subgoals relevant to a query.

[*]A preliminary version of this paper appeared as Chen, W. and Warren D.S. Query Evaluation under the Well-Founded Semantics. In *Proceedings of the 12th Annual ACM Symposium on Principles of Database Systems.* ACM, Washington, D.C., 1993, pp. 168-179.

[†]Supported in part by the National Science Foundation under Grant No. IRI-9212074 and IRI-9314897.

[‡]Supported in part by the National Science Foundation under Grant No. CCR-9102159 and New York State Science and Technology Foundation under Grant No. RDG-90173.

(iii) SLG resolution avoids both positive and negative loops and always terminates for programs with the bounded-term-size property. It has a polynomial time data complexity for well-founded negation of function-free programs. Through a delaying mechanism for handling ground negative literals involved in loops, SLG resolution avoids the repetition of any of its derivation steps.

Restricted forms of SLG resolution are identified for definite, locally stratified, and modularly stratified programs, shedding light on the role each transformation plays.

SLG resolution makes many more rule specifications into effective programs. With simple (user or computer generated) annotations, both SLDNF resolution and SLG resolution can be used in a single application, allowing a smooth integration of Prolog computation and tabled evaluation of queries. Furthermore Prolog compiler technology has been adapted for two efficient implementations of SLG resolution. For all these reasons we believe that SLG resolution will provide the computational basis for the next generation of logic programming systems.

Contents

Symbols Used in the Paper

$\sim A$	a negative literal
$\mathcal{HU}$	the Herbrand universe used for all programs in a query evaluation
$\mathcal{LF}$	a language of function symbols for constructing $\mathcal{HU}$
$\mathcal{HB}_P$	the Herbrand base of a program P
f,**u**,**t**	truth values, where $\mathbf{f} < \mathbf{u} < \mathbf{t}$
$I \preceq J$	the truth ordering over interpretations
$I \subseteq J$	the information ordering over interpretations
$\mathcal{T}_P$	the transformation over interpretations for a program P
$\mathcal{T}_P^{\uparrow n}$	an ordinal power of $\mathcal{T}_P$
$LPM(P)$	the least three-valued model of a non-negative program
τ_P	a variation of $\mathcal{T}_P$
$\tau_P^{\uparrow n}$	an ordinal power of τ_P
$\frac{P}{I}$	the quotient of P modulo I
$\mathcal{ST}3(P)$	the set of all three-valued stable models of P
$\mathcal{WF}(P)$	the well-founded partial model of P
$\sim B^B$	a delayed ground negative literal
B_H^A	a delayed positive literal with control annotation
G	an X-rule — a rule with delayed literals in its body
H	the head atom of an X-rule
e	an X-element — an annotated X-rule
ρ	an X-sequence (of X-elements)
$\rho_1 \cdot \rho_2$	the concatenation of X-two sequences
$\rho \cdot e$	the concatenation of ρ with a singleton sequence containing e
$\rho_1 \subseteq \rho_2$	ρ_1 is a prefix of ρ_2
$\bigcup\{\rho_i : i < \beta\}$	the least upper bound of a chain of X-sequences
$\mathcal{S}$	a system
Λ	a set of subgoals
$\mathcal{S}_1 \sqsubseteq \mathcal{S}_2$	a partial order over systems
Π_P	the maximum number of literals in a rule body in P
$\mathcal{N}(n)$	the number of distinct atoms with argument sizes $\leq n$
$P(\mathcal{S})$	the program for X-rules that are not disposed in $\mathcal{S}$
G^A	a rule in $P(\mathcal{S})$ corresponding to an X-rule G of subgoal A
$I\|_P$	an interpretation restricted an interpretation of P
$P \cup P(\mathcal{S})$	the program that is the union of P and $P(\mathcal{S})$
$\|P\|$	the number of occurrences of rules in P

1 Introduction

The seminal work by Apt, Van Emden and Kowalski [2, 44] provided a foundation for both the declarative and the procedural semantics of logic programs. According to [44], a program rule can be viewed as a procedure declaration, and a literal in a rule body can be viewed as a procedure call. This operational interpretation is formalized in *SLD resolution* (Linear resolution with Selection function for Definite programs), which is sound and complete for positive queries with respect to the least model semantics of definite programs [2, 44].

Clark [15] extended SLD resolution to *SLDNF resolution* – SLD resolution with Negation as finite Failure. A ground negative literal succeeds if its positive counterpart finitely fails, and fails if its positive counterpart succeeds. SLDNF resolution serves well the purpose of an operational semantics for predicate logic as a programming language. It has the advantages of goal-oriented computation and efficient stack-based memory management and is the computation strategy used in Prolog systems.

Significant progress has been made in understanding default negation, leading to several variations of SLDNF resolution, including SLS resolution [31] and global SLS resolution [29, 36]. These are ideal procedures for computing the perfect model [31] and the well-founded partial model [46] of a logic program. Like SLD resolution, SLDNF and (global) SLS resolution use a top-down goal reduction search strategy. They may not terminate due to infinite recursion (possibly through negation) even for function-free programs. This prevents them from being used directly for query evaluation in data and knowledge bases. When they do terminate, repeated computation of identical subgoals may result in unacceptable performance.

Partial solutions have been proposed to improve the termination properties of top-down computation and to avoid redundant evaluation of subgoals. Several extensions of SLD resolution with memoing have been studied, including extension tables [16], OLDT resolution [43], and QSQR [48]. The main idea is to keep a global table of subgoals and their answers that have been computed. If a subgoal is identical to or subsumed by a previous one, instead of being solved using rules in a program, it is solved using answers computed for the previous subgoal. This avoids infinite branches and redundant computation due to repeated subgoals in the search space of SLD resolution. These techniques have been generalized to stratified programs [20, 40] and modularly stratified programs [37].

Non-termination may also occur due to infinite recursion through negation, which has to be treated differently from infinite recursion in definite programs. A positive loop, such as $p \leftarrow p$, is considered failed as can be seen in the well-founded partial model of $p \leftarrow p$ where p is false. In contrast, a negative loop, such as $p \leftarrow \sim p$, is considered indeterminate since p is undefined in the well-founded partial model of $p \leftarrow \sim p$.

Mechanisms for handling infinite recursion through negation have been studied in tabled evaluation of queries, including WELL! [6] and XOLDTNF resolution [14]. The key idea is to associate a set of ground negative literals, called the *negative context*, with each subgoal. The negative context for the initial subgoal is empty. When a subgoal A in a negative context N calls a ground negative literal $\sim B$, $\sim B$ is replaced with an undefined truth value $\mathbf{u}$ if $\sim B \in N$. Otherwise, the truth value of $\sim B$ is determined by evaluating B in a larger negative context, namely $N \cup \{\sim B\}$. For a function-free program, the Herbrand base is finite and so the size of a negative context is finite and infinite recursion through negation is avoided. The use of negative contexts, however, prevents the full sharing of answers of a subgoal across different negative contexts. In the worst case, a subgoal may be evaluated in a number of negative contexts that is exponential in the size of the Herbrand base of a function-free programs [11].

Techniques for effective set-at-a-time query evaluation have been studied in deductive databases, including magic sets [4, 5], magic templates [33] and Alexander templates [39]. The main idea is to simulate top-down SLD resolution to avoid generation of tuples irrelevant to the given goal. In fact, tuples of magic predicates correspond to subgoals maintained in OLDT resolution [43]. For definite programs, it has been shown [8, 39] that the top-down with memoing and the set-at-a-time approaches are essentially equivalent.

Methods of query processing have been investigated for stratified and modularly stratified programs [3, 34, 37]. With negation, the major issue becomes maintaining dependencies among magic tuples (or subgoals) so as to ensure that a positive subgoal be fully evaluated before its negative counterpart is solved. Kemp *et al.* [18] developed a technique that computes the well-founded partial model using a doubled program, one for deriving definitely true answers and the other for deriving potentially true answers. The doubled program technique may make too many magic facts true, which means that more subgoals are evaluated than necessary. Morishita [26] proposed an alternating fixpoint semantics tailored to magic sets computation, which generates fewer magic facts.

This paper presents a partial deduction framework for query evaluation, called *SLG resolution* (*L*inear resolution with *S*election function for *G*eneral logic programs). SLG resolution addresses the problems of non-termination and redundant computation of identical subgoals. Rather than detecting and handling infinite recursion through negation, we focus on the complementary problem of ensuring the complete evaluation of subgoals. We summarize the main results as follows.

First, seven fundamental transformations are identified that can be applied to transform a query step by step into a set of answers with respect to the well-founded partial model. Restricted forms of SLG resolution are identified for definite, locally stratified, and modularly stratified programs, shedding light on the role each transformation plays. These programs do not have to pay the overhead for transformations that are not needed.

Second, the separation of logical issues of query evaluation from procedural ones results in the maximum freedom in control strategies. SLG resolution allows a programmer or an implementer to choose an arbitrary computation rule for selecting a literal from a rule body and to choose an arbitrary strategy for selecting which transformation to apply.

Third, SLG resolution supports answer sharing of subgoals that are variants of each other. A subgoal is guaranteed to be evaluated only once. SLG resolution terminates for programs with the bounded-term-size property and has a polynomial time data complexity for well-founded negation of function-free programs.

Finally, SLG resolution *delays* ground negative literals that are involved in a loop and simplifies them away when their truth values become known to be true or false. The delaying mechanism is the key to the maximum freedom in control strategies and enables SLG resolution to avoid the repetition of any derivation step. More importantly, three-valued stable models, other than the well-founded partial model, can be computed by further processing the answers of subgoals relevant to a query, possibly with delayed literals, under the well-founded semantics. However, it has been shown by Marek and Truszczynski [24] that for propositional logic programs P, determining whether P has a (two-valued) stable model is NP-complete.

From a software engineering point of view, the major advantage of SLG resolution is its upward compatibility with existing Prolog systems. Although Prolog is notorious for its non-logical features, there is a real value in making SLG resolution available to Prolog applications. Several implementations of SLG resolutions have been carried out, including a Prolog meta interpreter [13], a partial implementation in a Prolog compiler called XSB [38], and a partial implementation using Prolog program transformation and Prolog-C interface [35]. Experimental results [35, 38] have demonstrated that Prolog compiler technology can be adapted for an efficient implementation of SLG resolution, providing impressive performance for in-memory query evaluation of deductive databases.

2 Three-Valued Stable Models

This section reviews the notion of three-valued stable models of logic programs [32]. The basic terminology of logic programs [22] is assumed.

An *atom* is of the form $p(t_1, ..., t_n)$, where p is an n-ary predicate symbol and $t_1, ..., t_n$ are terms. A *literal* L is either an atom A or its negation $\sim A$. The existential closure of a literal L is denoted by $\exists L$, and the universal closure of L is denoted by $\forall L$.

A *rule* C is of the form:

$$H \leftarrow L_1, ..., L_n$$

where the rule head H is an atom, and $L_1, ..., L_n (n \geq 0)$ in the rule body are literals. If $n = 0$, C is also called a *fact*. A *program* P is a (possibly infinite) multiset of rules. (Unique labeling or annotations will be used when there is a need to distinguish between different occurrences of a rule in P.) An expression, which can be an atom, a literal or a rule, is *ground* if it is variable-free.

This paper considers query evaluation as a process of partial deduction, which may involve multiple programs that are possibly infinite, even though the original program is normally finite. Allowing a program to be infinite makes it possible to consider the intermediate result of partial deduction as a program. To relate the semantics of one program to another, we use a single Herbrand universe for all programs.

Specifically, we assume a countable language $\mathcal{LF}$ of function symbols. $\mathcal{LF}$ contains all function symbols that occur in programs involved in the evaluation of a query, plus a unary function symbol f' and a zero-ary function symbol c' that do not occur in any of the programs being considered. The symbols f' and c' are needed to cope with the "universal query problem" [31], where the semantics of a program containing a single fact, $p(a)$, may imply $\forall X.p(X)$ if the Herbrand universe is $\{a\}$. But the empty answer substitution cannot be obtained for $p(X)$ by SLD resolution. The introduction of new symbols f' and c' eliminates such situations.

The *Herbrand universe* $\mathcal{HU}$ is the set of all ground terms that can be constructed using function symbols in $\mathcal{LF}$. An *instance* of an expression, which can be an atom, a literal, or a rule, is obtained by replacing every variable in the expression with a term constructed from function symbols in $\mathcal{LF}$ and variables.

Let P be a program. The *Herbrand instantiation* of P is the set of all the ground instances of rules in P. The *Herbrand base* of P, denoted by $\mathcal{HB}_P$, is the set of all ground atoms that are constructed using predicates in P and terms in $\mathcal{HU}$.

Let $\mathbf{f}, \mathbf{u}, \mathbf{t}$ be truth values ordered by $\mathbf{f} < \mathbf{u} < \mathbf{t}$. An *interpretation* I of a program P is a mapping from $\mathcal{HB}_P$ to $\{\mathbf{f}, \mathbf{u}, \mathbf{t}\}$. I can be represented by a partition of $\mathcal{HB}_P$, $Pos(I) \cup Und(I) \cup Neg(I)$, where $Pos(I)$ (respectively, $Und(I)$, $Neg(I)$) is the set of ground atoms A such that

$I(A)$ is **t** (respectively **u**, **f**). Any two of these sets will uniquely determine I. I can also be viewed as the set $Pos(I) \cup \{\sim B | B \in Neg(I)\}$ of ground literals.

Definition 2.1 Let P_1 and P_2 be programs such that $\mathcal{HB}_{P_1} \subseteq \mathcal{HB}_{P_2}$, and let I be an interpretation of P_2. Then the *restriction of I to* P_1, denoted by $I|_{P_1}$, is an interpretation of P_1 whose mapping is the restriction of the mapping I to $\mathcal{HB}_{P_1}$. □

Definition 2.2 [32] Let I be an interpretation of a program P.

- A ground atom A is *true* in I if $A \in Pos(I)$ and is *false* in I if $A \in Neg(I)$, and is *undefined* if $A \in Und(I)$;
- A ground negative literal $\sim A$ is *true* in I if $A \in Neg(I)$ and is *false* in I if $A \in Pos(I)$, and is *undefined* if $A \in Und(I)$;
- The existential closure $\exists L$ of a literal L is *true* in I, denoted by $I \models \exists L$, if some ground instance of L is true in I; and $\exists L$ is *false* in I if all ground instances of L are false in I;
- The universal closure $\forall L$ of a literal L is *true* in I, denoted by $I \models \forall L$, if every ground instance of L is true in I; and $\forall L$ is *false* in I if some ground instance of L is false in I.
- A ground rule, $H \leftarrow L_1, ..., L_n$, is *true* in I if
  - H is true in I when all L_i's in the rule body are true in I; and
  - at least one of the L_i's in the rule body is false in I when H is false in I.
- A rule G is *true* in I if every ground instance of G is true in I, and is *false* in I if some ground instance of G is false in I.

I is a *model* of P if every rule in P is true in I. □

Definition 2.3 [32] Let I and J be interpretations of a program P. There are two natural orderings between interpretations, namely the *truth ordering* $\preceq$ (also called *Fitting-ordering*) and the *information ordering* $\subseteq$, where

- $I \preceq J$ if $Pos(I) \subseteq Pos(J)$ and $Neg(I) \supseteq Neg(J)$;
- $I \subseteq J$ if $Pos(I) \subseteq Pos(J)$ and $Neg(I) \subseteq Neg(J)$.

Models that are least in the sense of the truth ordering $\preceq$ are called *least* models. Models that are smallest in the sense of information ordering $\subseteq$ are called *smallest* models. □

Associated with each program P there is a mapping $\mathcal{T}_P$ over interpretations. It is a generalization of the immediate consequence operator in [44].

Definition 2.4 [32] Let P be a program and I be an interpretation of P. Then $\mathcal{T}_P(I)$ is an interpretation of P such that for every $H \in \mathcal{HB}_P$,

- $H \in Pos(\mathcal{T}_P(I))$ if and only if there is a rule, $H \leftarrow L_1, ..., L_n$, in the Herbrand instantiation of P and every $L_i (1 \leq i \leq n)$ is true in I;

- $H \in Neg(\mathcal{T}_P(I))$ if and only if for every rule with H in the head, $H \leftarrow L_1, ..., L_n$, in the Herbrand instantiation of P, some $L_i (1 \leq i \leq n)$ is false in I.

Let $\emptyset$ be the least interpretation of P, in which all ground atoms in $\mathcal{HB}_P$ are false. The *powers* of $\mathcal{T}_P$ are defined as follows:

$$\begin{array}{lll} \mathcal{T}_P^{\uparrow 0} & = \emptyset & \\ \mathcal{T}_P^{\uparrow n} & = \mathcal{T}_P(\mathcal{T}_P^{\uparrow (n-1)}) & \text{if } n \text{ is a successor ordinal} \\ & = \sqcup\{\mathcal{T}_P^{\uparrow k} : k < n\} & \text{if } n \text{ is a limit ordinal} \end{array}$$

where $\sqcup$ is the least upper bound operation of interpretations of P with respect to the truth ordering $\preceq$. □

We assume that there is a special ground atom $\mathbf{u}$. Atom $\mathbf{u}$ is always undefined, i.e., $\mathbf{u} \in Und(I)$. It can appear only in rule bodies in a program. A *non-negative program* is a multiset of rules whose bodies do not contain any negative literals, but may contain atom $\mathbf{u}$.

Theorem 2.1 ([32]) *Let P be a non-negative program. Then P has a unique least three-valued model, denoted by $LPM(P)$. Furthermore, $\mathcal{T}_P$ has a least fixed point, which coincides with $\mathcal{T}_P^{\uparrow \omega}$ and $LPM(P)$.*

An interpretation I can also be determined by specifying $Pos(I)$ and $Und(I)$. Let P be a non-negative program and I be an interpretation of P. We define $\tau_P(I)$ such that for every $H \in \mathcal{HB}_P$,

- $H \in Pos(\tau_P(I))$ if and only if there is a rule, $H \leftarrow L_1, ..., L_n$, in the Herbrand instantiation of P and every $L_i (1 \leq i \leq n)$ is true in I;
- $H \in Und(\tau_P(I))$ if $H \notin Pos(\tau_P(I))$ and there is a rule, $H \leftarrow L_1, ..., L_n$, in the Herbrand instantiation of P and every $L_i (1 \leq i \leq n)$ is either true or undefined in I.

The powers of τ_P are defined as those of $\mathcal{T}_P$.

Lemma 2.2 *Let P be a non-negative program and I be an interpretation of P. Then $\mathcal{T}_P(I) = \tau_P(I)$. If P is a non-negative program, then τ_P has a least fixpoint, which coincides with $\tau_P^{\uparrow \omega}$ and $LPM(P)$.*

Proof: Notice that $Pos(\mathcal{T}_P(I)) = Pos(\tau_P(I))$, and $H \in Neg(\mathcal{T}_P(I))$ if and only if $H \notin Pos(\tau_P(I)) \cup Und(\tau_P(I))$. Thus $\mathcal{T}_P(I) = \tau_P(I)$. The rest of the lemma follows from Theorem 2.1. □

Definition 2.5 [32] Let P be a program and I be an interpretation of P. The *quotient of P modulo I*, denoted by $\frac{P}{I}$, is the non-negative program obtained from the Herbrand instantiation of P by:

- deleting every rule with a negative literal in the body that is false in I; and
- deleting every negative literal in a rule body that is true in I; and

- replacing with **u** every negative literal in a rule body that is undefined in I.

I is a *three-valued stable model* of P if $I = LPM(\frac{P}{I})$. The set of all three-valued stable models of P is denoted by $\mathcal{ST}3(P)$. □

The notion of three-valued stable models is a generalization of both the well-founded partial model [46] and the two-valued stable models [17].

Theorem 2.3 ([32]) *Let P be a program, and $\mathcal{WF}(P)$ be the well-founded partial model of P. Then $\mathcal{WF}(P)$ is the smallest three-valued stable model of P. Stable models as defined by Gelfond and Lifschitz coincide with two-valued stable models.*

3 Query Evaluation as Partial Deduction

Partial deduction is a program transformation technique that specializes a logic program with respect to a query to produce a more efficient or simpler program [21, 23]. The new program should be equivalent to the original one as far as the query is concerned. This paper uses partial deduction for query evaluation with respect to the well-founded partial model. The primary motivation for using partial deduction is to understand the fundamental transformations involved in query evaluation, and to separate the logical issues from the procedural ones.

Recall that unification can be viewed as a process of transforming a system of equations from a general form into a solved form [25]. Each transformation preserves all solutions or unifiers. We follow a similar approach to query evaluation, which is considered a process of transforming a system of subgoals with associated rules into one that contains only answers of subgoals. This section introduces by examples the representation of systems of subgoals and various transformations.

3.1 Partial Deduction of Definite Programs

Without loss of generality, we consider queries that are represented as atoms, and use "subgoal" as a synonym for "atom". For definite programs, the partial deduction of a subgoal corresponds to SLD resolution with tabling such as OLDT resolution [43].

Consider an atomic query $q(X)$ with respect to the following program:

(C1) $q(X) \leftarrow p(X).$
(C2) $q(a).$
(C3) $p(X) \leftarrow q(X).$

In all the examples, we will always label each rule in a program for convenience of reference, and assume a left-to-right computation rule.

A *system* is a set of pairs of the form $(A : \rho)$, where A is a subgoal and ρ is a sequence of annotated rules for A. No two pairs in a system have the same subgoal, where two subgoals are considered the same if they are renaming variants of each other. The sequence ρ of annotated rules represents the history of the addition and/or deletion of rules for subgoal A. The annotation of each rule in ρ indicates where the rule is derived from and whether some rule is disposed. Intuitively, a rule is disposed when it no longer has anything to contribute to the derivation. In general, the sequence ρ can be transfinite, which is necessary because the well-founded partial model of a program may be transfinite.

Initially, the system is empty. The query atom $q(X)$ is introduced as the first subgoal. The initial sequence of $q(X)$ is an arbitrary sequence of all rules obtained by resolving $q(X) \leftarrow q(X)$ on $q(X)$ in the body with rules in the program. The following is the system after $q(X)$ is introduced:

$$\left\{ q(X): \left\{ \begin{array}{lll} 0: & q(X) \leftarrow p(X) & \langle C1 \rangle \\ 1: & q(a) & \langle C2 \rangle \end{array} \right\} \right\}$$

An annotated rule of the form $G\langle C \rangle$, where C is (the label of) a rule in a program, means that G is obtained by resolution with rule C in the program. The sequence of a subgoal is written as a set of pairs $\alpha : AG$ where α is an ordinal and AG is an annotated rule. In general, the annotation in AG indicates where the rule in AG is derived from — a rule in a program or an earlier rule in the sequence, and in the latter case, whether the earlier rule in the sequence is disposed.

For each non-disposed rule in the sequence of a subgoal, the head captures variable bindings that have been accumulated, and the rule body contains literals that remain to be solved in order to derive an answer for the subgoal. A rule with the empty body is an *answer*, e.g., $q(a)$ in the above sequence of $q(X)$. For a rule with a non-empty body, a literal is selected from the rule body by a computation rule. The atom of the selected literal is added as a new subgoal if it is not in the current system. In the example above, $p(X)$ is selected from the body of $q(X) \leftarrow p(X)$ and is added as a new subgoal. Like $q(X)$, $p(X)$ has its own sequence of annotated rules obtained by resolution of $p(X) \leftarrow p(X)$ on $p(X)$ in the body with rules in a program. So the system becomes:

$$\left\{ \begin{array}{ll} q(X): & \left\{ \begin{array}{lll} 0: & q(X) \leftarrow p(X) & \langle C1 \rangle \\ 1: & q(a) & \langle C2 \rangle \end{array} \right\} \\ p(X): & \left\{ \begin{array}{lll} 0: & p(X) \leftarrow q(X) & \langle C3 \rangle \end{array} \right\} \end{array} \right\}$$

The selected atom of $p(X) \leftarrow q(X)$ is $q(X)$, which is a subgoal already in the current system. Thus it is solved using only answers of the subgoal $q(X)$, such as $q(a)$ in the sequence of $q(X)$. This avoids repeated evaluation of identical subgoals.

By solving the selected atom of $p(X) \leftarrow q(X)$ using the answer $q(a)$, the sequence of $p(X)$ is extended as follows:

$$p(X): \left\{ \begin{array}{lll} 0: & p(X) \leftarrow q(X) & \langle C3 \rangle \\ 1: & p(a) & \langle 0, q(a) \rangle \end{array} \right\}$$

The annotated rule $p(a)\langle 0, q(a) \rangle$ means that $p(a)$ is obtained from an earlier rule in the sequence of $p(X)$, namely $p(X) \leftarrow q(X)$ corresponding to ordinal 0, by solving the selected atom using an answer $q(a)$. Continuing in a similar way, $p(a)$ is an answer and is used to solve the selected atom $p(X)$ in $q(X) \leftarrow p(X)$. The sequence of $q(X)$ is extended by adding the resulting rule:

$$q(X): \left\{ \begin{array}{lll} 0: & q(X) \leftarrow p(X) & \langle C1 \rangle \\ 1: & q(a) & \langle C2 \rangle \\ 2: & q(a) & \langle 0, p(a) \rangle \end{array} \right\}$$

The rule $q(a)$ corresponding to ordinal 2 in the sequence of $q(X)$ is an answer. But it is a redundant answer and so is not used to solve the selected atom $q(X)$ in the rule $p(X) \leftarrow q(X)$ for $p(X)$.

At this point, all answers have been derived for subgoals relevant to a query. No new subgoal can be introduced and no annotated rule can be added to the sequence of any subgoal in the system. For definite programs, all annotated rules with a selected atom can be disposed at the end, leaving only answers of subgoals relevant to the original query. The final system of our example is as follows:

$$\left\{ \begin{array}{ll} q(X): & \left\{ \begin{array}{lll} 0: & q(X) \leftarrow p(X) & \langle C1 \rangle \\ 1: & q(a) & \langle C2 \rangle \\ 2: & q(a) & \langle 0, p(a) \rangle \\ 3: & \mathit{disposed} & \langle 0 \rangle \end{array} \right\} \\ p(X): & \left\{ \begin{array}{lll} 0: & p(X) \leftarrow q(X) & \langle C3 \rangle \\ 1: & p(a) & \langle 0, q(a) \rangle \\ 2: & \mathit{disposed} & \langle 0 \rangle \end{array} \right\} \end{array} \right\}$$

where $\mathit{disposed}\langle\alpha\rangle$ for some ordinal α indicates that an earlier rule corresponding to α in a sequence of annotated rules is simply disposed.

Given a finite program P, an atomic query A, and a system of subgoals that is constructed during the partial deduction of A with respect to P, the set of all rules that are not disposed in the system constitutes an intermediate program for subgoals relevant to A. When a final system is reached in which all rules that are not disposed are answers, partial deduction becomes query evaluation in the sense that the program for subgoals corresponding to the final system contains only answers.

3.2 Partial Deduction of Stratified Programs

A subgoal B is *completed* when all the rules in its sequence that are not disposed are answers. For stratified negation, a ground negative literal $\sim B$ succeeds only if B is completed and has no answers. However, when there is mutual recursion among subgoals, these subgoals may have rules in their sequences that have selected atoms but are not disposed, even though they cannot possibly have any answers. These rules with selected atoms have to be disposed so that the subgoals become completed and their negative counterparts can succeed.

Consider an atomic query m with respect to the following stratified program:

(C1)	$m \leftarrow \sim q(b)$.
(C2)	$q(X) \leftarrow p(X)$.
(C3)	$q(a)$.
(C4)	$p(X) \leftarrow q(X)$.

The first subgoal in the system is the initial atomic query m:

$$\left\{ m: \left\{ 0: \ m \leftarrow \sim q(b) \quad \langle C1 \rangle \right\} \right\}$$

The selection of $\sim q(b)$ in the rule body of $m \leftarrow \sim q(b)$ for m results in a new subgoal $q(b)$. The new subgoal $q(b)$ is processed as in definite programs, leading in two transformation steps to the following system:

$$\left\{ \begin{array}{ll} m: & \left\{ 0: \ m \leftarrow \sim q(b) \quad \langle C1 \rangle \right\} \\ q(b): & \left\{ 0: \ q(b) \leftarrow p(b) \quad \langle C2 \rangle \right\} \\ p(b): & \left\{ 0: \ p(b) \leftarrow q(b) \quad \langle C4 \rangle \right\} \end{array} \right\}$$

Notice that neither $q(b)$ nor $p(b)$ can have any answers. But they are not completed since they have rules in their sequences that have a selected atom and are not disposed. On the other hand, $\sim q(b)$ cannot succeed unless $q(b)$ is completed without any answers.

We introduce the COMPLETION transformation to handle this situation. It is applied to a non-empty set of subgoals. It checks rules in the associated sequences of annotated rules of the subgoals and, under certain conditions, disposes all rules that are not answers. For the set of subgoals, $\{q(b), p(b)\}$, two properties are satisfied, both of which are required by the COMPLETION transformation.

First, neither of them has a rule that has a selected negative literal but is not disposed. Second, all non-disposed rules that have a selected atom have been processed using all answers of the selected atom. The first condition ensures that the situation is similar to tabled evaluation of definite programs, and the second condition guarantees that all answers of a selected atom have been returned.

In the case of $\{q(b), p(b)\}$, neither $q(b)$ nor $p(b)$ has any answer. The COMPLETION transformation is applied to $\{q(b), p(b)\}$ and all rules of $q(b)$ and $p(b)$ that are not answers are disposed:

$$\left\{\begin{array}{ll} m: & \left\{\begin{array}{lll} 0: & m \leftarrow \sim q(b) & \langle C1\rangle \end{array}\right\} \\ q(b): & \left\{\begin{array}{lll} 0: & q(b) \leftarrow p(b) & \langle C2\rangle \\ 1: & \textit{disposed} & \langle 0\rangle \end{array}\right\} \\ p(b): & \left\{\begin{array}{lll} 0: & p(b) \leftarrow q(b) & \langle C4\rangle \\ 1: & \textit{disposed} & \langle 0\rangle \end{array}\right\} \end{array}\right\}$$

Both $q(b)$ and $p(b)$ are now completed without any answers. By negation as failure, $\sim q(b)$ can be removed so that the rule for m is disposed and an annotated answer m is added, leading to a final system of subgoals:

$$\left\{\begin{array}{ll} m: & \left\{\begin{array}{lll} 0: & m \leftarrow \sim q(b) & \langle C1\rangle \\ 1: & m & \langle 0\rangle \end{array}\right\} \\ q(b): & \left\{\begin{array}{lll} 0: & q(b) \leftarrow p(b) & \langle C2\rangle \\ 1: & \textit{disposed} & \langle 0\rangle \end{array}\right\} \\ p(b): & \left\{\begin{array}{lll} 0: & p(b) \leftarrow q(b) & \langle C4\rangle \\ 1: & \textit{disposed} & \langle 0\rangle \end{array}\right\} \end{array}\right\}$$

The annotated rule $m\langle 0\rangle$ in the sequence associated with m means that the rule corresponding to ordinal 0, namely $m \leftarrow \sim q(b)$, is disposed.

3.3 Handling Negative Loops with Delaying

With recursion through negation, a set of subgoals may be waiting for each other in a circular fashion through selected negative literals. Each of them may be waiting for others to become completed in order to solve the selected negative literals in its own rules, but none of them can be completed. We introduce the DELAYING transformation to skip selected negative literals that are ground so that computation can proceed.

Consider an atomic query s with respect to the following program:

(C1) $s \leftarrow \sim t.$
(C2) $t \leftarrow \sim s.$

The evaluation of subgoal s leads to a system as follows:

$$\left\{ \begin{array}{ll} s: & \left\{ 0: \ s \leftarrow {\sim}t \quad \langle C1 \rangle \right\} \\ t: & \left\{ 0: \ t \leftarrow {\sim}s \quad \langle C2 \rangle \right\} \end{array} \right\}$$

In the sequences associated with subgoals, rules that have a selected literal can be viewed as directed edges in a dependency graph of subgoals. Each edge is labeled positive or negative according to whether the selected literal is an atom or the negation of an atom. In the system above, the rule $s \leftarrow {\sim}t$ in the sequence for subgoal s can be viewed as a negative edge from s to t, and the rule $t \leftarrow {\sim}s$ in the sequence of subgoal t can be viewed as a negative edge from t to s. There is a loop among s and t in the dependency graph involving negative edges. Such a loop is called a *negative loop*. Each of s and t is waiting for the other to be completed in order to solve the selected negative literal in the body of its rule, which keeps either from ever proceeding.

Our approach is to *delay* selected negative ground literals so that computation can proceed. Delayed literals will not be selected by a computation rule, even though they may be simplified away later if their truth values become known to be true or false. Each delayed literal is annotated by the corresponding subgoal, e.g., ${\sim}s^s$. The notion of rules is extended to allow delayed literals in rule bodies. When the selected ground negative literal of a rule is delayed, the rule is disposed and replaced with a new rule that has a delayed literal in the body. Suppose that ${\sim}t$ is delayed in the rule for s. We derive the following system:

$$\left\{ \begin{array}{ll} s: & \left\{ \begin{array}{lll} 0: & s \leftarrow {\sim}t & \langle C1 \rangle \\ 1: & s \leftarrow {\sim}t^t & \langle 0 \rangle \end{array} \right\} \\ t: & \left\{ \begin{array}{lll} 0: & t \leftarrow {\sim}s & \langle C2 \rangle \end{array} \right\} \end{array} \right\}$$

where $s \leftarrow {\sim}t^t\langle 0 \rangle$ in the sequence of s means that the rule corresponding to ordinal 0 in the sequence of s is disposed and replaced with $s \leftarrow {\sim}t^t$.

A rule with only delayed literals in its body is considered an answer. Subgoal s becomes completed since all its rules that are not disposed are answers. However, subgoal s neither succeeds with an answer that has an empty body, nor fails without any answers. Accordingly the selected negative literal ${\sim}s$ in the rule for t neither succeeds nor fails. Our approach is to delay ${\sim}s$ too, which results in a new system:

$$\left\{ \begin{array}{ll} s: & \left\{ \begin{array}{lll} 0: & s \leftarrow {\sim}t & \langle C1 \rangle \\ 1: & s \leftarrow {\sim}t^t & \langle 0 \rangle \end{array} \right\} \\ t: & \left\{ \begin{array}{lll} 0: & t \leftarrow {\sim}s & \langle C2 \rangle \\ 1: & t \leftarrow {\sim}s^s & \langle 0 \rangle \end{array} \right\} \end{array} \right\}$$

This is also the final system since no further processing can be done. Neither of the subgoals succeeds or fails. In fact, both s and t are undefined in the well-founded partial model of the given program.

3.4 Propagation of Delayed Literals

When an answer is used to solve the selected atom in a rule body, the variable bindings accumulated in the head of the answer are propagated by unification with the selected atom. But the answer may have delayed literals in its body and there is a question whether they should also be propagated. In our approach they are not. We next illustrate how answers with delayed literals are used to solve the selected atom of a rule. After that we give an example that shows that propagating delayed literals of answers may cause an exponential explosion.

Consider the evaluation of an atomic query $p(X)$ with respect to the following program:

(C1) $s \leftarrow \sim t.$

(C2) $t \leftarrow \sim s.$

(C3) $p(X) \leftarrow q(X,Y), r(Y).$

(C4) $q(a,Y) \leftarrow \sim s.$

(C5) $r(b).$

Following the procedures discussed in previous subsections (and assuming a left-to-right computation rule), the evaluation of the query $p(X)$ leads to the following system:

$$\left\{ \begin{array}{ll} p(X): & \left\{ \begin{array}{lll} 0: & p(X) \leftarrow q(X,Y), r(Y) & \langle C3 \rangle \end{array} \right\} \\ q(X,Y): & \left\{ \begin{array}{lll} 0: & q(a,Y) \leftarrow \sim s & \langle C4 \rangle \end{array} \right\} \\ s: & \left\{ \begin{array}{lll} 0: & s \leftarrow \sim t & \langle C1 \rangle \\ 1: & s \leftarrow \sim t^t & \langle 0 \rangle \end{array} \right\} \\ t: & \left\{ \begin{array}{lll} 0: & t \leftarrow \sim s & \langle C2 \rangle \\ 1: & t \leftarrow \sim s^s & \langle 0 \rangle \end{array} \right\} \end{array} \right\}$$

Subgoal s is completed with an answer that has a delayed literal $\sim t^t$. Since s neither succeeds nor fails, the negative literal $\sim s$ in the rule of $q(X,Y)$ is also delayed. The sequence associated with $q(X,Y)$ is extended as follows:

$$q(X,Y): \left\{ \begin{array}{lll} 0: & q(a,Y) \leftarrow \sim s & \langle C4 \rangle \\ 1: & q(a,Y) \leftarrow \sim s^s & \langle 0 \rangle \end{array} \right\}$$

The annotated rule $q(a,Y) \leftarrow \sim s^s \langle 0 \rangle$ means that the rule corresponding to 0, namely $q(a,Y) \leftarrow \sim s$, is disposed and replaced with $q(a,Y) \leftarrow \sim s^s$.

By definition, $q(a,Y) \leftarrow \sim s^s$ is an answer for subgoal $q(X,Y)$ as it contains only delayed literals in its body. This answer should be used to solve the selected atom $q(X,Y)$ in the rule for $p(X)$. Our approach does not propagate the delayed literal $\sim s^s$ in the body of the answer. Instead, we introduce a positive delayed literal. The sequence of annotated rules associated with $p(X)$ is extended as follows:

$$p(X): \left\{ \begin{array}{lll} 0: & p(X) \leftarrow q(X,Y), r(Y) & \langle C3 \rangle \\ 1: & p(a) \leftarrow q(a,Y)^{q(X,Y)}_{q(a,Y)}, r(Y) & \langle 0, q(a,Y) \rangle \end{array} \right\}$$

where $q(a,Y)^{q(X,Y)}_{q(a,Y)}$ represents a positive delayed literal whose truth value depends upon the truth value of some answer of subgoal $q(X,Y)$ with $q(a,Y)$ in the head. The annotation $\langle 0, q(a,Y) \rangle$ means that the corresponding rule is obtained by solving the selected atom of an earlier rule corresponding to 0 using some answer with $q(a,Y)$ in the head. Since variable bindings in the head of an answer have been propagated through unification, a delayed positive literal serves only as a place holder that can be simplified away if and when its truth value later becomes known. The annotation in a delayed positive literal provides control information for such simplification.

Recall that a computation rule cannot select a delayed literal. Thus $r(Y)$ is selected in the body of $p(a) \leftarrow q(a,Y)^{q(X,Y)}_{q(a,Y)}, r(Y)$. A new subgoal $r(Y)$ is added to the system; its initial sequence contains one rule obtained by resolution with rule labeled $C5$ in the program.

$$r(Y) : \left\{ \; 0: \;\; r(b) \;\; \langle C5 \rangle \; \right\}$$

The rule $r(b)$ is an answer. It is used to solve the selected atom $r(Y)$ in the rule corresponding to 1 in the sequence of $p(X)$. The sequence of $p(X)$ is extended to:

$$p(X) : \left\{ \begin{array}{lll} 0: & p(X) \leftarrow q(X,Y), r(Y) & \langle C3 \rangle \\ 1: & p(a) \leftarrow q(a,Y)^{q(X,Y)}_{q(a,Y)}, r(Y) & \langle 0, q(a,Y) \rangle \\ 2: & p(a) \leftarrow q(a,b)^{q(X,Y)}_{q(a,Y)} & \langle 1, r(b) \rangle \end{array} \right\}$$

Now the COMPLETION transformation can be applied to the set $\{p(X)\}$ of subgoals, which produces a final system in which all subgoals are completed:

$$\left\{ \begin{array}{ll} p(X): & \left\{ \begin{array}{lll} 0: & p(X) \leftarrow q(X,Y), r(Y) & \langle C3 \rangle \\ 1: & p(a) \leftarrow q(a,Y)^{q(X,Y)}_{q(a,Y)}, r(Y) & \langle 0, q(a,Y) \rangle \\ 2: & p(a) \leftarrow q(a,b)^{q(X,Y)}_{q(a,Y)} & \langle 1, r(b) \rangle \\ 3: & \mathit{disposed} & \langle 0 \rangle \\ 4: & \mathit{disposed} & \langle 1 \rangle \end{array} \right\} \\ q(X,Y): & \left\{ \begin{array}{lll} 0: & q(a,Y) \leftarrow \sim s & \langle C4 \rangle \\ 1: & q(a,Y) \leftarrow \sim s^s & \langle 0 \rangle \end{array} \right\} \\ s: & \left\{ \begin{array}{lll} 0: & s \leftarrow \sim t & \langle C1 \rangle \\ 1: & s \leftarrow \sim t^t & \langle 0 \rangle \end{array} \right\} \\ t: & \left\{ \begin{array}{lll} 0: & t \leftarrow \sim s & \langle C2 \rangle \\ 1: & t \leftarrow \sim s^s & \langle 0 \rangle \end{array} \right\} \\ r(Y): & \left\{ \; 0: \;\; r(b) \;\; \langle C5 \rangle \; \right\} \end{array} \right\}$$

There are two reasons that we do not propagate delayed literals in the body of an answer when the answer is used to solve the selected atom of a rule. First, it is not necessary to propagate delayed literals for well-founded negation. We are interested in only whether an atom is definitely true or false in the well-founded partial model, not in how many ways an atom may depend upon other literals. Second, for a subgoal A, there may be many answers for A that have the same atom in the head, but different delayed literals in the body. Propagating delayed literals may generate an exponential number of distinct answers for a subgoal, as the following example shows.

Example 3.1 Let n be an arbitrary positive integer. Consider the following program:

$max(n).$
$succ(0,1).\ succ(1,2).\ \ldots\ succ(n-1,n).$
$p(X) \leftarrow succ(X,Y), r(X), p(Y).$
$p(X) \leftarrow max(X), r(X).$
$r(X) \leftarrow \sim q(X,a).$
$r(X) \leftarrow \sim q(X,b).$
$q(X,a) \leftarrow r(X).$
$q(X,b) \leftarrow r(X).$

Notice that each $r(i)$, where $0 \leq i \leq n$, has two answers. One is $r(i) \leftarrow {\sim}q(i,a)^{q(i,a)}$, and the other is $r(i) \leftarrow {\sim}q(i,b)^{q(i,b)}$. For the evaluation of subgoal $p(0)$, our approach generates a polynomial number (in n) of answers for relevant subgoals. In contrast, propagating delayed literals of answers of each $r(i)$ would have led to an exponential number (in n) of answers for $p(0)$. □

3.5 Simplification of Delayed Literals

Not all delayed literals are undefined in the well-founded partial model of a program. Some of them may become known to be true or false later and should be simplified away. Simplification of delayed literals is necessary so that when a subgoal is completed, a ground instance of the subgoal is true in the well-founded partial model if and only if it is an instance of the head of an answer that has an empty body, and is false if and only if it is not an instance of the head of any answer of the subgoal.

Suppose that p is evaluated with respect to the following program:

(C1) $p \leftarrow p.$
(C2) $p \leftarrow {\sim}s.$
(C3) $s \leftarrow {\sim}r.$
(C4) $r \leftarrow {\sim}s, r.$

Assuming a left-to-right computation rule, the evaluation of p leads to the following system after the introduction of several subgoals:

$$\left\{\begin{array}{ll} p: & \left\{\begin{array}{lll} 0: & p \leftarrow p & \langle C1\rangle \\ 1: & p \leftarrow {\sim}s & \langle C2\rangle \end{array}\right\} \\ s: & \left\{\begin{array}{lll} 0: & s \leftarrow {\sim}r & \langle C3\rangle \end{array}\right\} \\ r: & \left\{\begin{array}{lll} 0: & r \leftarrow {\sim}s, r & \langle C4\rangle \end{array}\right\} \end{array}\right\}$$

There is a negative loop involving subgoals s and r. The DELAYING transformation is applied to each of the selected negative literals, namely ${\sim}s$ and ${\sim}r$. This results in a new system:

$$\left\{\begin{array}{ll} p: & \left\{\begin{array}{lll} 0: & p \leftarrow p & \langle C1\rangle \\ 1: & p \leftarrow {\sim}s & \langle C2\rangle \\ 2: & p \leftarrow {\sim}s^s & \langle 1\rangle \end{array}\right\} \\ s: & \left\{\begin{array}{lll} 0: & s \leftarrow {\sim}r & \langle C3\rangle \\ 1: & s \leftarrow {\sim}r^r & \langle 0\rangle \end{array}\right\} \\ r: & \left\{\begin{array}{lll} 0: & r \leftarrow {\sim}s, r & \langle C4\rangle \\ 1: & r \leftarrow {\sim}s^s, r & \langle 0\rangle \end{array}\right\} \end{array}\right\}$$

Subgoal p has an answer with delayed literals, namely $p \leftarrow {\sim}s^s$. The answer is returned to the selected atom p in the rule $p \leftarrow p$. The sequence of annotated rules of p is extended as follows:

$$p: \left\{\begin{array}{lll} 0: & p \leftarrow p & \langle C1\rangle \\ 1: & p \leftarrow {\sim}s & \langle C2\rangle \\ 2: & p \leftarrow {\sim}s^s & \langle 1\rangle \\ 3: & p \leftarrow p_p^p & \langle 0, p\rangle \end{array}\right\}$$

The singleton set $\{p\}$ of subgoals is completely evaluated by definition. All rules of p that are not answers are disposed by the transformation COMPLETION:

$$p : \left\{ \begin{array}{lll} 0: & p \leftarrow p & \langle C1 \rangle \\ 1: & p \leftarrow \sim s & \langle C2 \rangle \\ 2: & p \leftarrow \sim s^s & \langle 1 \rangle \\ 3: & p \leftarrow p_p^p & \langle 0, p \rangle \\ 4: & \mathit{disposed} & \langle 0 \rangle \end{array} \right\}$$

The singleton set $\{r\}$ of subgoals is completely evaluated by definition. All rules of subgoal r that are not answers are disposed by the transformation COMPLETION. The final sequence of annotated rules associated with r is as follows:

$$r : \left\{ \begin{array}{lll} 0: & r \leftarrow \sim s, r & \langle C4 \rangle \\ 1: & r \leftarrow \sim s^s, r & \langle 0 \rangle \\ 2: & \mathit{disposed} & \langle 1 \rangle \end{array} \right\}$$

Since subgoal r is completed without any answers, all occurrences of the delayed literal $\sim r^r$ can be deleted. In particular, the sequence of annotated rules of subgoal s is extended to the following:

$$s : \left\{ \begin{array}{lll} 0: & s \leftarrow \sim r & \langle C3 \rangle \\ 1: & s \leftarrow \sim r^r & \langle 0 \rangle \\ 2: & s & \langle 1 \rangle \end{array} \right\}$$

Subgoal s succeeds with an answer that has s in the head and an empty body. The rule $p \leftarrow \sim s^s$ of subgoal p can be disposed.

$$p : \left\{ \begin{array}{lll} 0: & p \leftarrow p & \langle C1 \rangle \\ 1: & p \leftarrow \sim s & \langle C2 \rangle \\ 2: & p \leftarrow \sim s^s & \langle 1 \rangle \\ 3: & p \leftarrow p_p^p & \langle 0, p \rangle \\ 4: & \mathit{disposed} & \langle 0 \rangle \\ 5: & \mathit{disposed} & \langle 2 \rangle \end{array} \right\}$$

The only answer for p is $p \leftarrow p_p^p$. In the well-founded semantics of the original program, p is false. Notice that the answer $p \leftarrow p_p^p$ was derived because of the answer with a delayed literal, $p \leftarrow \sim s^s$. Even though $p \leftarrow \sim s^s$ has been deleted, $p \leftarrow p_p^p$ remains. We introduce another transformation, called ANSWER COMPLETION, which deletes answers with some positive delayed literals under certain conditions (to be defined later). For the example above, the final sequence of annotated rules of p is as follows:

$$p : \left\{ \begin{array}{lll} 0: & p \leftarrow p & \langle C1 \rangle \\ 1: & p \leftarrow \sim s & \langle C2 \rangle \\ 2: & p \leftarrow \sim s^s & \langle 1 \rangle \\ 3: & p \leftarrow p_p^p & \langle 0, p \rangle \\ 4: & \mathit{disposed} & \langle 0 \rangle \\ 5: & \mathit{disposed} & \langle 2 \rangle \\ 6: & \mathit{disposed} & \langle 3 \rangle \end{array} \right\}$$

where the answer $p \leftarrow p_p^p$ has been disposed. Thus subgoal p is completed without any answers.

4 Transformations

This section presents the formal definitions of systems and transformations of systems.

4.1 Systems

Definition 4.1 A *subgoal* is an atom. Two subgoals are considered the same if they are renaming variants of each other.

A negative *delayed* literal is of the form $\sim B^B$, where B is a ground atom. A positive *delayed* literal is of the form B_H^A, where B, H and A are atoms such that B is an instance of H and H is an instance of A. If θ is a substitution, then $(B_H^A)\theta$ is defined as $(B\theta)_H^A$. □

As we have discussed in Section 3, a positive delayed literal of the form B_H^A is created when the selected atom of a rule is solved using an answer of a subgoal A that has H in its head and some delayed literals in its body, in which case H must be an instance of A. The annotations A and H in B_H^A provide control information for the later simplification of the delayed literal later.

Definition 4.2 An *X-rule* G is of the form:

$$H \leftarrow L_1, ..., L_n$$

where H is an atom, and each $L_i (1 \leq i \leq n)$ is an atom, the negation of an atom, or a delayed literal, and $n \geq 0$. If $n = 0$, G is called a *fact*. If every $L_i (1 \leq i \leq n, n \geq 0)$ is a delayed literal, G is called an *answer*.

A *computation rule* is an algorithm that selects from the body of an X-rule G a literal L that is an atom or the negation of an atom (if there is any). L is called the *selected literal* of G. If L is an atom, it is also called the *selected atom* of G. □

Notice that a computation rule never selects a delayed literal. However, it may select a negative literal that is not ground. We try to solve a non-ground negative literal by negation as failure if it is sound to do so.

Definition 4.3 Let P be a finite program, C be a rule in P, G be an X-rule, α be an ordinal, and H be an atom. Then an *X-element* is of the form $G\langle C\rangle$, $G\langle\alpha\rangle$, $G\langle\alpha, H\rangle$, or *disposed*$\langle\alpha\rangle$.

An *X-sequence* ρ is a mapping from all ordinals that are smaller than some ordinal α to the set of X-elements. The ordinal α is the *length* of ρ. □

Each subgoal in a system has an associated X-sequence. The X-sequence captures the history of the addition/disposal of X-rules for the subgoal during its partial deduction (as shown in the examples in Section 3). Each X-element in the X-sequence indicates from where an X-rule is derived — a rule in a program or an X-rule earlier in the X-sequence — and/or whether the X-rule earlier in the X-sequence is disposed.

More specifically, let ρ be the X-sequence of a subgoal A. Let β be an ordinal such that $\rho(\beta) = e$ for some X-element e, and α be an ordinal such that $\alpha < \beta$.

- If e is of the form $G\langle C\rangle$, where G is an X-rule and C is a rule in a program, then G is created by resolving $A \leftarrow A$ on A in the body with C;

- If e is of the form $G\langle\alpha\rangle$, where G is an X-rule, then the X-rule corresponding to α in ρ is disposed and replaced by G;
- If e is of the form $G\langle\alpha, H\rangle$, where G is an X-rule and H is an atom, then G is obtained by solving the selected atom of the X-rule corresponding to α in ρ using an answer with H in the head;
- If e is $\mathit{disposed}\langle\alpha\rangle$, then the X-rule corresponding to α in ρ is simply disposed.

We consider two main operations over X-sequences: concatenation and the least upper bound of an increasing chain of X-sequences. The former is used to extend the X-sequence of a subgoal in a system, and the latter is needed for the transfinite definition of the notion of SLG derivation given in Section 4.3 later.

Definition 4.4 Let ρ_1 and ρ_2 be X-sequences of length α_1 and α_2 respectively. The concatenation of ρ_1 with ρ_2, denoted by $\rho_1 \cdot \rho_2$, is an X-sequence of length $\alpha_1 + \alpha_2$ such that $(\rho_1 \cdot \rho_2)(i) = \rho_1(i)$ for every $i (0 \leq i < \alpha_1)$, and $(\rho_1 \cdot \rho_2)(\alpha_1 + j) = \rho_2(j)$ for every $j (0 \leq j < \alpha_2)$.

The X-sequence ρ_1 is said to be a *prefix* of $\rho_1 \cdot \rho_2$. If ρ_2 is a sequence of length 1 such that $\rho_2(0) = e$ for some X-element e, we also write $\rho_1 \cdot \rho_2$ as $\rho_1 \cdot e$.

□

There is a natural prefix partial order over X-sequences. Let ρ_1 and ρ_2 be X-sequences. Then $\rho_1 \subseteq \rho_2$ if ρ_1 is a prefix of ρ_2. An X-sequence ρ of length α can also be viewed as a set of pairs $\{\langle i, \rho(i)\rangle \mid 0 \leq i < \alpha\}$, in which case the prefix relation reduces to the subset relation.

Definition 4.5 Let β be an ordinal and $\rho_i (0 \leq i < \beta)$ be an increasing chain of X-sequences (with respect to $\subseteq$), and let α be an ordinal such that the length of each $\rho_i (0 \leq i < \beta)$ is less than α. Then the least upper bound of the chain $\rho_i (0 \leq i < \beta)$, denoted by $\bigcup\{\rho_i | 0 \leq i < \beta\}$, exists since the length of each $\rho_i (0 \leq i < \beta)$ is less than α. It is the X-sequence that is the union of all $\rho_i (0 \leq i < \beta)$ when an X-sequence is viewed as a set. □

The intermediate state of the partial deduction of a query is represented by a *system*.

Definition 4.6 Let P be a finite program, and R be a computation rule. A *system* $\mathcal{S}$ is a set of pairs of the form $(A : \rho)$, where A is a subgoal and ρ is its X-sequence, such that no two pairs in $\mathcal{S}$ have the same subgoal. A subgoal A is said to be *in* $\mathcal{S}$ if $(A : \rho) \in \mathcal{S}$ for some X-sequence ρ.

Let $(A : \rho) \in \mathcal{S}$, where A is a subgoal and ρ is its X-sequence. Let G be an X-rule and α be an ordinal. G is said to be *the X-rule of* A *corresponding to* α *in* $\mathcal{S}$ if $\rho(\alpha)$ is either $G\langle C\rangle$, $G\langle i\rangle$, or $G\langle i, H\rangle$, where C is a rule in P, $i < \alpha$, and H is an atom.

Let G be an X-rule of A corresponding to α in $\mathcal{S}$. Then

- G is *disposed* if for some $j > \alpha$, $\rho(j)$ is either $\mathit{disposed}\langle\alpha\rangle$ or $G'\langle\alpha\rangle$ for some X-rule G';
- G is an *answer of* A if G is not disposed and all literals in the body of G are delayed literals;
- G is an *active X-rule of* A if G is not disposed and has a selected literal.

A subgoal A in $\mathcal{S}$ is *completed* if all X-rules of A that are not disposed are answers.

□

4.2 Transformations of Systems of Subgoals

Starting with the empty system of subgoals, each transformation transforms one system into another. The initial X-sequence of a subgoal is obtained by resolution with rules in a program. Without loss of generality, we consider only *atomic queries* that contain a single atom.

Definition 4.7 [X-resolution] Let G be an X-rule, of the form $H \leftarrow L_1, ..., L_n$, and L_i be the selected atom of G for some $i (1 \leq i \leq n)$. Let C be a rule and C', of the form $H' \leftarrow L'_1, ..., L'_m$, be a variant of C with variables renamed so that G and C' have no variables in common. Then G is *X-resolvable* with C if L_i and H' are unifiable. The X-rule:

$$(H \leftarrow L_1, ..., L_{i-1}, L'_1, ..., L'_m, L_{i+1}, ..., L_n)\theta$$

is the *X-resolvent* of G with C, where θ is the most general unifier of L_i and H'.

□

The selected atom in the body of an X-rule can be solved by an answer that may or may not have delayed literals. If the answer does have delayed literals, *X-factoring* is used to propagate variable bindings captured in the head atom of the answer and create a positive delayed literal for propagating the truth values of the delayed literals in the body of the answer.

Definition 4.8 [X-factoring] Let $\mathcal{S}$ be a system. Let G, of the form $H \leftarrow L_1, ..., L_n$, be an active X-rule of a subgoal A in $\mathcal{S}$, and let L_i be the selected atom of G for some $i (1 \leq i \leq n)$. Suppose that L_i is a subgoal in $\mathcal{S}$ and C is an answer of L_i in $\mathcal{S}$, and C', of the form $H' \leftarrow L'_1, ..., L'_m$, is a variant of C with variables renamed so that G and C' have no variables in common, and $m > 0$. Then the X-rule:

$$H\theta \leftarrow L_1\theta, ..., L_{i-1}\theta, (L_i\theta)^{L_i}_{H'}, L_{i+1}\theta, ..., L_n\theta$$

is the *X-factor* of G with C, where θ is the most general unifier of L_i and H'. □

The selected negative literal, possibly containing variables, in the body of an X-rule can be solved if the positive counterpart succeeds (without binding any variables) or fails. A delayed literal can be simplified if it is successful or failed.

Definition 4.9 Let $\mathcal{S}$ be a system.

- A subgoal A in $\mathcal{S}$ *succeeds* if A has an answer that has A in the head and an empty body, and *fails* if A is completed in $\mathcal{S}$ without any answers;
- A ground negative delayed literal $\sim B^B$ is *successful* if subgoal B fails, and is *failed* if subgoal B succeeds;
- A positive delayed literal B^A_H is *successful* if subgoal A has an answer that has H in the head and an empty body, and is *failed* if subgoal A is completed and does not have any answer with H in the head.

□

The transformation COMPLETION requires the notion of a set of subgoals that are completely evaluated so that it can dispose their active X-rules that are not answers.

Definition 4.10 Let $\mathcal{S}$ be a system and Λ be a non-empty set of subgoals in $\mathcal{S}$, none of which is completed. Λ is said to be *completely evaluated* if for every subgoal $A \in \Lambda$, where $(A : \rho) \in \mathcal{S}$ for some X-sequence ρ, either A succeeds, or for every active X-rule G of A corresponding to some ordinal α in $\mathcal{S}$, there exists an atom A_1 such that:

- A_1 is the selected atom of G; and
- A_1 is a subgoal in $\mathcal{S}$ that is either completed or in Λ; and
- for every atom H that is the head of some answer of A_1 in $\mathcal{S}$, there exists an ordinal $i > \alpha$ and an X-rule G' such that $\rho(i) = G'\langle \alpha, H \rangle$.

□

The transformation ANSWER COMPLETION is needed to get rid of answers of subgoals that have positive delayed literals in their bodies, provided that they are not supported in the following sense.

Definition 4.11 Let $\mathcal{S}$ be a system, A be a subgoal in $\mathcal{S}$ and H be an atom that occurs in the head of some answer of A. Then H is *supported by A in $\mathcal{S}$* if

(i) either A is not completed; or

(ii) there exists an answer C of A that has H in the head such that for every positive delayed literal $(B_1)^{A_1}_{H_1}$ in the body C, H_1 is supported by A_1.

No atom is supported by A in $\mathcal{S}$ unless it follows from (i) and (ii). □

Notice that if a subgoal A in a system $\mathcal{S}$ has an answer C that has an atom H in the head and only negative delayed literals in the body, then H is supported by A in $\mathcal{S}$.

Let P be a finite program, R be an arbitrary but fixed computation rule, and Q be an atomic query. The following are all the transformations of systems, each of which transforms a current system $\mathcal{S}$ into a new one.

- $\boxed{\text{NEW SUBGOAL}}$ Let A be a subgoal that is not in $\mathcal{S}$ and that satisfies one of the following conditions:

  - A is the initial atomic query Q; or
  - there is an active X-rule of some subgoal in $\mathcal{S}$ whose selected literal is either A or $\sim A$.

  Let $C_0, C_1, ..., C_{k-1} (k \geq 0)$ be all rules in program P with which $A \leftarrow A$ is X-resolvable, and $G_i (0 \leq i < k)$ be the X-resolvent of $A \leftarrow A$ with C_i. Let ρ be the X-sequence of length k such that $\rho(i) = G_i\langle C_i \rangle (0 \leq i < k)$. Then

$$\frac{\mathcal{S}}{\mathcal{S} \cup \{(A : \rho)\}}$$

  Remark: Repeated subgoals are not solved by resolution with rules in a program.

- $\boxed{\text{POSITIVE RETURN}}$ Let $(A : \rho) \in \mathcal{S}$, where A is a subgoal and ρ is its X-sequence. Let G be an active X-rule of A corresponding to an ordinal α in $\mathcal{S}$. Let A_1 be the selected atom of G, and let C be an answer of subgoal A_1 in $\mathcal{S}$ that has H in its head. If there is no ordinal $i > \alpha$ such that $\rho(i) = G_1\langle\alpha, H\rangle$ for some X-rule G_1, then

$$\frac{\mathcal{S}}{\mathcal{S} - \{(A : \rho)\} \cup \{(A : \rho \cdot (G_2\langle\alpha, H\rangle)\}}$$

  where G_2 is the X-resolvent of G with C if C has an empty body and is the X-factor of G with C if C has some delayed literals in its body.

  Remark: A_1 may have multiple answers with the same atom H in the head. Only one of them is used by POSITIVE RETURN to solve the selected atom A_1 in G. So, in particular, redundant answers are not used.

- $\boxed{\text{NEGATIVE RETURN}}$ Let $(A : \rho) \in \mathcal{S}$, where A is a subgoal and ρ is its X-sequence. Let G be an active X-rule of a subgoal A corresponding to an ordinal α in $\mathcal{S}$, and let $\sim A_1$ be the selected literal of G, where A_1 either succeeds or fails. Then

$$\frac{\mathcal{S}}{\mathcal{S} - \{(A : \rho)\} \cup \{(A : \rho \cdot \mathit{disposed}\langle\alpha\rangle)\}} \quad \text{if } A_1 \text{ succeeds}$$

$$\frac{\mathcal{S}}{\mathcal{S} - \{(A : \rho)\} \cup \{(A : \rho \cdot G'\langle\alpha\rangle)\}} \quad \text{if } A_1 \text{ fails}$$

  where G' is G with the selected literal $\sim A_1$ deleted.

- $\boxed{\text{DELAYING}}$ Let $(A : \rho) \in \mathcal{S}$, where A is a subgoal and ρ is its X-sequence. Let G be an active X-rule of subgoal A corresponding to an ordinal α in $\mathcal{S}$, and let $\sim B$ be the selected literal of G such that $\sim B$ is ground. Then

$$\frac{\mathcal{S}}{\mathcal{S} - \{(A : \rho)\} \cup \{(A : \rho \cdot (G'\langle\alpha\rangle))\}}$$

  where G' is obtained from G by replacing $\sim B$ with $\sim B^B$.

  Remark: Non-ground negative literals are not delayed.

- $\boxed{\text{COMPLETION}}$ Let Λ be a non-empty set of subgoals in $\mathcal{S}$ that is completely evaluated. Then

$$\frac{\mathcal{S}}{\text{for every } A \in \Lambda, \text{ replace } (A, \rho) \in \mathcal{S} \text{ with } (A, \rho \cdot \rho')}$$

  where ρ' is an arbitrary sequence of all X-elements of the form $\mathit{disposed}\langle\alpha\rangle$, where α is an ordinal such that the X-rule of A corresponding to α in ρ is an active X-rule.

  Remark: COMPLETION does not depend upon any *a priori* stratification ordering of predicates or atoms. Instead, a set of subgoals is inspected and completed dynamically.

- $\boxed{\text{SIMPLIFICATION}}$ Let $(A : \rho) \in \mathcal{S}$, where A is a subgoal that is completed and ρ is its X-sequence. Let G be the X-rule of A corresponding to an ordinal α in $\mathcal{S}$ that is not disposed and that has a delayed literal L in its body which is either failed or successful. Then

$$\frac{\mathcal{S}}{\mathcal{S} - \{(A:\rho)\} \cup \{(A:\rho \cdot \mathit{disposed}\langle\alpha\rangle)\}} \quad \text{if } L \text{ is failed}$$

$$\frac{\mathcal{S}}{\mathcal{S} - \{(A:\rho)\} \cup \{(A:\rho \cdot G'\langle\alpha\rangle)\}} \quad \text{if } L \text{ is successful}$$

where G' is G with L deleted from its body.

Remark: simplifying a positive delayed literal does not generate variable bindings. The reason is that variable bindings have already been propagated by X-factoring in POSITIVE RETURN when a positive delayed literal is generated.

- $\boxed{\text{ANSWER COMPLETION}}$ Let $(A:\rho) \in \mathcal{S}$, where A is a subgoal that is completed and ρ is its X-sequence. Let H be the head atom of some answer of A in $\mathcal{S}$ such that H is not supported by A. Then

$$\frac{\mathcal{S}}{\mathcal{S} - \{(A:\rho)\} \cup \{(A:\rho \cdot \rho'\}}$$

where ρ' is an arbitrary sequence of all X-elements of the form $\mathit{disposed}\langle\alpha\rangle$, where α is an ordinal such that the X-rule of A corresponding to α in ρ is an answer that is not disposed and that has H in the head.

4.3 Derivation and SLG Resolution

Definition 4.12 Let P be a finite program, R be an arbitrary but fixed computation rule, and Q be an atomic query. An *SLG derivation* for Q is a sequence of systems $\mathcal{S}_0, \mathcal{S}_1, ..., \mathcal{S}_\alpha$ such that:

- $\mathcal{S}_0$ is the empty system $\{\}$;
- for each successor ordinal $\beta + 1 \leq \alpha$, $\mathcal{S}_{\beta+1}$ is obtained from $\mathcal{S}_\beta$ by an application of one of the transformations, namely, NEW SUBGOAL, POSITIVE RETURN, NEGATIVE RETURN, DELAYING, COMPLETION, SIMPLIFICATION, or ANSWER COMPLETION;
- for each limit ordinal $\beta \leq \alpha$, $\mathcal{S}_\beta$ is such that $(A:\rho) \in \mathcal{S}_\beta$ if A is a subgoal in $\mathcal{S}_i$ for some $i < \beta$ and $\rho = \bigcup\{\rho' | (A:\rho') \in \mathcal{S}_j \text{ and } j < \beta\}$.

If no transformation is applicable to $\mathcal{S}_\alpha$, $\mathcal{S}_\alpha$ is called a *final system* of Q.

SLG resolution is the process of constructing an SLG derivation for a query Q with respect to a finite program P under a computation rule R. □

To show that a final system always exists, we prove that each SLG derivation is a monotonically increasing sequence of systems with respect to some partial ordering and each system in an SLG derivation is bounded in size by some ordinal.

Definition 4.13 Let P be a finite program, and $\mathcal{S}_1$ and $\mathcal{S}_2$ be systems. Then $\mathcal{S}_1 \sqsubseteq \mathcal{S}_2$ if for every $(A:\rho_1) \in \mathcal{S}_1$, there exists $(A:\rho_2) \in \mathcal{S}_2$ such that $\rho_1 \subseteq \rho_2$, i.e., ρ_1 is a prefix of ρ_2. □

Theorem 4.1 *Let P be a finite program, R be an arbitrary but fixed computation rule, and Q be an atomic query. Then* *a. there exists some ordinal λ such that for any system $\mathcal{S}$ in any SLG derivation for Q, the length of the X-sequence of each subgoal in $\mathcal{S}$ is bounded by λ;*

b. every SLG derivation of Q is a monotonically increasing sequence of systems with respect to $\sqsubseteq$; and

c. there exists some SLG derivation for Q, $\mathcal{S}_0, \mathcal{S}_1, ..., \mathcal{S}_\alpha$, for some ordinal α such that $\mathcal{S}_\alpha$ is a final system.

Proof: Let Π_P be the maximum number of literals in the body of a rule in P.

(a) Let $\mathcal{S}$ be any system in an SLG derivation for Q, and $(A : \rho)$ be any pair in $\mathcal{S}$, where A is a subgoal and ρ is its X-sequence. The length of ρ is bounded based upon the following observations:

- When subgoal A is added to a system, its initial X-sequence contains the X-rules that are obtained by resolving $A \leftarrow A$ on A in the body with rules in P. The number of literals in the body of each X-rule is bounded by Π_P. Since P is finite, the initial X-sequence of A is finite.

- When a transformation extends the X-sequence of A, X-elements are appended to the end of the X-sequence. Let α be an ordinal and let G be the X-rule of a subgoal A corresponding to α. We discuss the possible forms of X-elements e that are appended.

  (i) If e is $disposed\langle\alpha\rangle$, then G is disposed and can be disposed at most once by the construction of an SLG derivation;

  (ii) If e is of the form $G'\langle\alpha\rangle$, where G' is an X-rule, then G is disposed and replaced by G' (in NEGATIVE RETURN, DELAYING, or SIMPLIFICATION).

  (iii) If e is $G'\langle\alpha, H\rangle$, where G' is an X-rule and H is an atom, then G' is obtained from G by solving the selected atom of G with an answer that has H in the head, and G' has one literal less than G that is not delayed. The number of such X-rules G' that can be obtained from G by POSITIVE RETURN is bounded by the number of distinct atoms (that are not variants of each other), which is countable.

Therefore for each X-rule G of a subgoal A corresponding to an ordinal α, the number of X-rules G' that can be obtained directly from G is bounded by the number of distinct atoms, which is countable. In both (ii) and (iii), either G' and G have the same number of literals in the body and G' has fewer literals that are not delayed, or G' and G have the same number of literals that are not delayed and G' has fewer literals that are delayed. For any chain of X-rules, $G_0, G_1, ..., G_l$, where each $G_{j+1} (0 \leq j < l)$ is obtained from G_j by some transformation, $l \leq 2\Pi_P$. Thus there exists some countable ordinal λ by which the length of the X-sequence ρ of A in $\mathcal{S}$ is bounded. The ordinal λ depends upon only the finite program P and the language $\mathcal{LF}$ (defined in Section 2) that is countable, and is applicable to the X-sequence ρ of any subgoal A in any system $\mathcal{S}$ in an arbitrary SLG derivation for Q.

For (b), let $\mathcal{S}_0, \mathcal{S}_1, ..., \mathcal{S}_\alpha$ be an SLG derivation for Q. By definition, $\mathcal{S}_0$ is the empty system $\{\}$. For each successor ordinal $i + 1 (0 < i + 1 \leq \alpha)$, $\mathcal{S}_i \sqsubseteq \mathcal{S}_{i+1}$ since each transformation either adds a new subgoal to $\mathcal{S}_i$ or extends the X-sequences of some subgoals in $\mathcal{S}_i$. For a limit ordinal $i(0 < i \leq \alpha)$, $(A : \rho) \in \mathcal{S}_i$ if and only if A is a subgoal in $\mathcal{S}_j$ for some $j < i$ and $\rho = \bigcup\{\rho' | (A : \rho') \in \mathcal{S}_j \text{ and } j < i\}$. By (a), there exists some ordinal λ by which the length of ρ' for $(A : \rho') \in \mathcal{S}_j$ for each $j < i$ is bounded. Thus $\bigcup\{\rho' | (A : \rho') \in \mathcal{S}_j \text{ and } j < i\}$ is well defined. Clearly $\mathcal{S}_j \sqsubseteq \mathcal{S}_i$ for every $j < i$. This concludes the inductive proof of (b).

For (c), the size of a system is bounded since the number of distinct subgoals (that are not variants of each other) is countable and the length of the X-sequence of a subgoal in a system is bounded. As each transformation increases the size of a system, and each SLG derivation for Q is a monotonically increasing sequence of systems, there must exist some SLG derivation for Q that ends with a final system. □

Theorem 4.1 shows that some final system can be derived for an atomic query Q, given a finite program P and an arbitrary but fixed computation rule R. It turns out that for any final system $\mathcal{S}$ for Q, either every subgoal in $\mathcal{S}$ is completed or some active X-rule of some subgoal in $\mathcal{S}$ has a selected negative literal that is not ground.

Definition 4.14 Let P be a finite program, and $\mathcal{S}$ be a system. $\mathcal{S}$ is *completed* if every subgoal in $\mathcal{S}$ is completed, and $\mathcal{S}$ is *floundered* if for some active X-rule G of some subgoal in $\mathcal{S}$, G has a selected negative literal that is not ground. □

Lemma 4.2 *Let P be a finite program, R be an arbitrary but fixed computation rule, and Q be an atomic query. Then for every final system $\mathcal{S}$ for Q, $\mathcal{S}$ is either completed or floundered.*

Proof: Follows from the definitions of transformations and floundered and completed systems. □

5 Soundness and Completeness

This section establishes the soundness and search space completeness of SLG resolution. First, SLG resolution is shown to preserve all three-valued stable models. Second, SLG resolution computes the well-founded semantics in the sense that in a final system that is completed, a ground instance of a subgoal is true if and only if it is an instance of the head of some answer of the subgoal that has an empty body, and is false if and only if it is not an instance of the head of any answer of the subgoal. Third, we establish the termination of SLG resolution for programs with the bounded-term-size property and the polynomial time data complexity of SLG resolution for function-free programs.

5.1 Relating Partial Answers of Subgoals to a Program

Let P be a finite program, R be an arbitrary but fixed computation rule, and $\mathcal{S}$ be a system in an SLG derivation for an atomic query. For each subgoal A in $\mathcal{S}$, the multiset of X-rules of A in $\mathcal{S}$ that are not disposed constitutes the set of partial answers for A. The head of each X-rule contains relevant variable bindings that have been accumulated; delayed literals in the rule body are partially solved and may be simplified away later; and the remaining literals in the rule body are yet to be solved with respect to the original program P.

To relate P to X-rules of subgoals in a system $\mathcal{S}$, we introduce some new predicates. But first, let us consider an example.

Example 5.1 Suppose that an atomic query $p(X)$ is evaluated with respect to the following program:

(C1)	$p(a).$
(C2)	$p(X) \leftarrow \sim s, p(a).$
(C3)	$s \leftarrow \sim s, s.$

Assuming a left-to-right computation rule, the following intermediate system $\mathcal{S}$ may be constructed:

$$\left\{\begin{array}{ll} p(X): & \left\{\begin{array}{lll} 0: & p(a) & \langle C1\rangle \\ 1: & p(X) \leftarrow \sim s, p(a) & \langle C2\rangle \\ 2: & p(X) \leftarrow \sim s^s, p(a) & \langle 1\rangle \\ 3: & p(X) \leftarrow \sim s^s & \langle 2, p(a)\rangle \end{array}\right\} \\ s: & \left\{\begin{array}{lll} 0: & s \leftarrow \sim s, s & \langle C3\rangle \\ 1: & s \leftarrow \sim s^s, s & \langle 0\rangle \end{array}\right\} \\ p(a): & \left\{\begin{array}{lll} 0: & p(a) & \langle C1\rangle \\ 1: & p(a) \leftarrow \sim s, p(a) & \langle C2\rangle \end{array}\right\} \end{array}\right\}$$

where $\sim s$ has been delayed in X-rules of subgoals $p(X)$ and s.

There are two observations on the treatment of X-rules of subgoals. First, X-rules of different subgoals are treated independently, even if they are subgoals of the same predicate. For instance, subgoal $p(a)$ may be completed before subgoal $p(X)$ since $p(a)$ already succeeds. Second, X-rules of a subgoal that have distinct head atoms are treated independently, especially in POSITIVE RETURN and SIMPLIFICATION and ANSWER COMPLETION. In POSITIVE RETURN, when the selected atom A in the body of an X-rule has multiple answers that have the same head atom, only one of them is used to solve A. However, if A has two answers with distinct head atoms, both will be used to solve A, even if the head atom of one answer subsumes the other. In both SIMPLIFICATION and ANSWER COMPLETION, answers with the same head atom are grouped together, in order to determine whether a delayed literal is successful or failed, or whether the head atom of an answer is supported. □

In relating a finite program P to X-rules of subgoals in a system $\mathcal{S}$, we introduce new predicates in a way that reflects the independent treatment of X-rules of different subgoals and the independent treatment of X-rules of the same subgoal that have different head atoms. Specifically, for every subgoal A in $\mathcal{S}$ and for every instance H of A, we introduce a new predicate whose arity is the number of distinct variables in H. Atoms of the new predicate will be written as B_H^A, where B is an instance of H. They are derived using X-rules of A in $\mathcal{S}$ that are not disposed and that have H in the head. There is a one-to-one correspondence between ground atoms of the new predicate and ground instances of H.

Definition 5.1 Let P be a finite program, R be an arbitrary but fixed computation rule, and Q be an atomic query. Let $\mathcal{S}$ be a system in an SLG derivation for Q.

Let G, of the form $H \leftarrow L_1, ..., L_n$, be an X-rule of a subgoal A in $\mathcal{S}$. Then we denote by G^A the rule of the form, $H_H^A \leftarrow L'_1, ..., L'_n$, where for each $i (1 \leq i \leq n)$,

- L'_i is L_i if L_i is not a delayed literal;
- L'_i is $\sim B_B^B$ if L_i is a negative delayed literal of the form $\sim B^B$;

- L_i' is B_H^A if L_i is a positive delayed literal of the form B_H^A.

We denote by $P(\mathcal{S})$ the program that is the multiset of all rules G^A, where G is an X-rule of a subgoal A in $\mathcal{S}$ that is not disposed. □

Example 5.2 For the system $\mathcal{S}$ in Example 5.1, the corresponding program $P(\mathcal{S})$ is as follows:

/* from non-disposed X-rules of $p(X)$ */
$p(a)_{p(a)}^{p(X)}$.
$p(X)_{p(X)}^{p(X)} \leftarrow \sim s_s^s, p(a)$.
$p(X)_{p(X)}^{p(X)} \leftarrow \sim s_s^s$.

/* from non-disposed X-rules of s */
$s_s^s \leftarrow \sim s_s^s, s$.

/* from non-disposed X-rules of $p(a)$ */
$p(a)_{p(a)}^{p(a)}$.
$p(a)_{p(a)}^{p(a)} \leftarrow \sim s, p(a)$.

In general, $P(\mathcal{S})$ depends upon the original program, unless $\mathcal{S}$ is completed. □

For technical reasons, we include in the Herbrand base $\mathcal{HB}_{P \cup P(\mathcal{S})}$ and $\mathcal{HB}_{P(\mathcal{S})}$ all ground atoms of the form B_H^A for every subgoal A in $\mathcal{S}$ and for every instance H of A and for every ground instance B of H. (An alternative is to introduce useless rules in $P(\mathcal{S})$ such that $H_H^A \leftarrow H_H^A$ for every subgoal A in $\mathcal{S}$ and every instance H of A.)

Definition 5.2 Let P be a finite program, R be an arbitrary but fixed computation rule, and Q be an atomic query. Let $\mathcal{S}$ be a system in an SLG derivation for Q. We associate with $\mathcal{S}$ a set of ground literals $I(\mathcal{S})$ of $P \cup P(\mathcal{S})$ as follows:

- if a subgoal A in $\mathcal{S}$ has an answer that has an atom H in the head and an empty body, then for every ground instance B of H, $B \in I(\mathcal{S})$ and $B_H^A \in I(\mathcal{S})$;
- if a subgoal A in $\mathcal{S}$ is completed and H is an instance of A and A has no answers with H in the head, then $\sim B_H^A \in I(\mathcal{S})$ for every ground instance B of H;
- if a subgoal A in $\mathcal{S}$ is completed and B is a ground instance of A such that B is not an instance of the head of any answer of A, then $\sim B \in I(\mathcal{S})$ and $\sim B_H^A \in I(\mathcal{S})$ for every instance H of A such that B is an instance of H.

□

The set $I(\mathcal{S})$ of ground literals captures subgoals that succeed or fail and delayed literals that are successful or failed. For example, if a subgoal A succeeds in $\mathcal{S}$, then every ground instance of A is in $I(\mathcal{S})$, and if a subgoal A fails, then $\sim B \in I(\mathcal{S})$ for every ground instance B of A. Similarly, if a positive delayed literal B_H^A is successful, then every ground instance of B_H^A is in $I(\mathcal{S})$, and if B_H^A is failed, then the negation of each ground instance of B_H^A is in $I(\mathcal{S})$. As we shall see later, for any system $\mathcal{S}$ in an SLG derivation for an atomic query, $I(\mathcal{S})$ is an interpretation of $P \cup P(\mathcal{S})$, i.e., $I(\mathcal{S})$ is consistent. At this point, we define $I(\mathcal{S})$ simply as a set of ground literals.

Lemma 5.1 *Let P be a finite program, R be an arbitrary but fixed computation rule, and Q be an atomic query. Let $\mathcal{S}_0, \mathcal{S}_1, ..., \mathcal{S}_\alpha$ be an arbitrary SLG derivation of Q, where α is an ordinal. Then $I(\mathcal{S}_0) \subseteq I(\mathcal{S}_1) \subseteq ... \subseteq I(\mathcal{S}_\alpha)$.*

Proof: The lemma follows from two observations. One is that an answer of a subgoal that has an empty body is never deleted by any transformation. The other is that when a subgoal A is completed, answers of A can be simplified, but no new answer can be added whose head atom is distinct from the head atom of any existing answer of A. $\Box$

Let P be a finite program and $\mathcal{S}$ be a system in an SLG derivation for an atomic query Q. To relate the semantics of $P(\mathcal{S})$ to that of P, we look at the least partial model $LPM(\frac{P \cup P(\mathcal{S})}{J})$, where J is an interpretation of $P \cup P(\mathcal{S})$ and $\frac{P \cup P(\mathcal{S})}{J}$ is the quotient of $P \cup P(\mathcal{S})$ modulo J. For every ground instance B of a subgoal A in $\mathcal{S}$, we compare the truth value of B with those of atoms of the form B^A_H in $LPM(\frac{P \cup P(\mathcal{S})}{J})$, where H is an instance of A and B is also an instance of H, provided that J satisfies certain conditions. The correctness of transformations is expressed in terms of some symmetry of $LPM(\frac{P \cup P(\mathcal{S})}{J})$, which is defined as follows.

Definition 5.3 Let P be a finite program, R be an arbitrary but fixed computation rule, and Q be an atomic query. Let $\mathcal{S}$ be a system in an SLG derivation for Q. Let J be an interpretation of $P \cup P(\mathcal{S})$.

J is *partially symmetric* if for every ground subgoal B in $\mathcal{S}$, $J(B) = J(B^B_B)$. J is *symmetric on a subgoal A in $\mathcal{S}$* if for every ground instance B of A,

- $J(B) = \mathbf{t}$ if and only if $J(B^A_H) = \mathbf{t}$ for some instance H of A; and
- $J(B) = \mathbf{f}$ if and only if $J(B^A_H) = \mathbf{f}$ for every instance H of A.

J is a *symmetric interpretation* of $P \cup P(\mathcal{S})$ if J is symmetric on every subgoal in $\mathcal{S}$. $\mathcal{S}$ is a *symmetric system* if for every interpretation J of $P \cup P(\mathcal{S})$ such that J is partially symmetric and $I(\mathcal{S}) \subseteq J$, $LPM(\frac{P \cup P(\mathcal{S})}{J})$ is symmetric. $\Box$

The correctness of $P(\mathcal{S})$ of a system $\mathcal{S}$ with respect to a program P is specified by the notions of symmetric systems and symmetric interpretations. In comparing $P(\mathcal{S})$ with P in the least partial model $LPM(\frac{P \cup P(\mathcal{S})}{J})$, where J is an interpretation of $P \cup P(\mathcal{S})$, J is required to satisfy two conditions. The condition $I(\mathcal{S}) \subseteq J$ originates from the observation that rules in $P(\mathcal{S})$ are essentially derived from rules in P by solving subgoals that succeed or fail and by simplifying delayed literals that are successful or failed. As mentioned previously, $I(\mathcal{S})$ represents subgoals that succeed or fail and delayed literals that are successful or failed. The other condition that J is partially symmetric is due to DELAYING, where a ground negative literal of the form $\sim B$ may be replaced by $\sim B^B$ in the body of a rule. Recall that a negative delayed literal $\sim B^B$ is viewed as $\sim B^B_B$ in $P(\mathcal{S})$. In $P(\mathcal{S})$, all negative literals are either of predicates in P or of the form $\sim B^B_B$, where B is a ground subgoal. In particular, when $\mathcal{S}$ is completed, all negative literals in $P(\mathcal{S})$, if any, are of the form $\sim B^B_B$, where B is a ground subgoal in $\mathcal{S}$.

Recall that $\frac{P \cup P(\mathcal{S})}{J}$ is a ground non-negative program that is obtained from the Herbrand instantiation of $P \cup P(\mathcal{S})$ by replacing every ground negative literal with its truth value in J. For each rule $B \leftarrow \psi$ in $\frac{P \cup P(\mathcal{S})}{J}$, ψ is a possibly empty conjunction of atoms and the special atom

u. Also notice that $\frac{P \cup P(\mathcal{S})}{J} = \frac{P}{J} \cup \frac{P(\mathcal{S})}{J}$. In the following, we establish several key properties relating $\frac{P}{J}$ to $\frac{P(\mathcal{S})}{J}$, which will be used later for proving the soundness and completeness of SLG resolution.

Lemma 5.2 *Let P be a finite program, R be an arbitrary but fixed computation rule, and Q be an atomic query. Let $\mathcal{S}_0, \mathcal{S}_1, ..., \mathcal{S}_\alpha$ be an arbitrary SLG derivation of Q, where α is an ordinal. Then for every $i (0 \leq i \leq \alpha)$, and for every partially symmetric interpretation J of $P \cup P(\mathcal{S}_i)$ such that $\cup_{0 \leq j < i} I(\mathcal{S}_j) \subseteq J$, and for every subgoal A that is not completed in $\mathcal{S}_i$ and for every ground instance B of A, if $B \leftarrow \psi$ is a rule in $\frac{P}{J}$, then $B_H^A \leftarrow \psi$ is a rule in $\frac{P(\mathcal{S}_i)}{J}$ for some instance H of A such that B is a ground instance of H.*

The intuition of Lemma 5.2 is as follows. Given a subgoal A that is not completed, its initial X-rules are obtained by X-resolution with rules in P. For each non-disposed X-rule G of A, if G has a selected atom, then G is never disposed except by COMPLETION, and if G has a selected negative literal, say L, then either L is solved by NEGATIVE RETURN, or L is delayed by DELAYING (if L is ground), or G remains a non-disposed X-rule of A. The assumption about J ensures that Lemma 5.2 continues to hold after an application of NEGATIVE RETURN or DELAYING. Notice that in $\cup_{0 \leq j < i} I(\mathcal{S}_j) \subseteq J$, if i is a successor ordinal of the form $\beta + 1$, then $\cup_{0 \leq j < i} I(\mathcal{S}_j) = I(\mathcal{S}_\beta)$ due to Lemma 5.1. Lemma 5.2 is used in the next subsection to establish one direction in the symmetry of $LPM(\frac{P \cup P(\mathcal{S}_i)}{J})$.

Proof of Lemma 5.2: The proof is based upon an induction on i. The basis case, $i = 0$, holds trivially since $\mathcal{S}_0$ is the empty system and has no subgoals.

Let i be a successor ordinal $\beta + 1$. Then $\mathcal{S}_i$ is obtained from $\mathcal{S}_\beta$ by one of the transformations. The cases of COMPLETION, SIMPLIFICATION, and ANSWER COMPLETION are trivial since they affect only X-rules of subgoals that are completed in $\mathcal{S}_i$. The case of NEW SUBGOAL follows from the use of the most general unifier in X-resolution in deriving the initial X-rules of a new subgoal. The case of POSITIVE RETURN follows from the inductive hypothesis since POSITIVE RETURN adds another X-rule to a subgoal that is not completed.

Let A be a subgoal in $\mathcal{S}_\beta$ and G be an active X-rule of A corresponding to an ordinal γ, and let $\sim A_1$ be the selected literal of G.

- NEGATIVE RETURN. If A_1 succeeds, then G is disposed in $\mathcal{S}_i$. By definition, A_1 must have an answer in $\mathcal{S}_\beta$ that has A_1 in the head and an empty body. Then for every ground instance B_1 of A_1, $B_1 \in I(\mathcal{S}_\beta)$ and $(B_1)_{A_1}^{A_1} \in I(\mathcal{S}_\beta)$. By assumption, $B_1 \in J$.

  If A_1 fails, then G is replaced with G' that is G with $\sim A_1$ deleted. By definition, A_1 is completed in $\mathcal{S}_\beta$ without any answers. Then for every ground instance B_1 of A_1, $\sim B_1 \in I(\mathcal{S}_\beta)$, and so $\sim B_1 \in J$ by assumption.

- DELAYING. If $\sim A_1$ is ground, then G is replaced with G' that is G except that $\sim A_1$ is replaced with $\sim (A_1)^{A_1}$. In $P(\mathcal{S}_i)$, a ground negative delayed literal $\sim (A_1)^{A_1}$ is viewed as $\sim (A_1)_{A_1}^{A_1}$. Since J is partially symmetric, $J(A_1) = J((A_1)_{A_1}^{A_1})$.

In each of the cases for NEGATIVE RETURN and DELAYING, $\frac{P(\mathcal{S}_i)}{J} = \frac{P(\mathcal{S}_\beta)}{J}$, and the lemma holds by inductive hypothesis.

Let i be a limit ordinal. If A is a subgoal in $\mathcal{S}_i$ that is not completed, then A is a subgoal in $\mathcal{S}_\beta$ for some $\beta < i$, and is not completed in every $\mathcal{S}_\beta$ such that $\beta < i$ and A is a subgoal in $\mathcal{S}_\beta$. For every ground instance B of A and for every rule $B \leftarrow \psi$ in $\frac{P}{J}$, let G be an X-rule with the fewest negative literals in its body that are not delayed such that for some $\beta < i$,

- G is an X-rule of A in $\mathcal{S}_\beta$, and
- there exists a rule $B_H^A \leftarrow \psi$ in $\frac{P(\mathcal{S}_\beta)}{J}$ that is obtained from a ground instance of G^A, where H is the head atom of G.

The existence of G is ensured by the inductive hypothesis.

There are two cases:

- if G has a selected atom or is an answer, then G is also an X-rule of A in $\mathcal{S}_i$ since G can be disposed only by COMPLETION or SIMPLIFICATION and A is not completed in $\mathcal{S}_i$. The lemma holds by inductive hypothesis;
- if G has a selected negative literal, then either G remains an X-rule of A in $\mathcal{S}_i$, in which case the lemma holds, or NEGATIVE RETURN or DELAYING has been applied to G. The latter case, however, contradicts the assumption that G has the fewest negative literals in its body that are not delayed.

$\square$

Lemma 5.3 *Let P be a finite program, R be an arbitrary but fixed computation rule, and Q be an atomic query. Let $\mathcal{S}_0, \mathcal{S}_1, ..., \mathcal{S}_\alpha$ be an arbitrary SLG derivation of Q, where α is an ordinal. For every $i (0 \le i \le \alpha)$ and for every partially symmetric interpretation J of $P \cup P(\mathcal{S}_i)$ such that $\cup_{0 \le j < i} I(\mathcal{S}_j) \subseteq J$, and for every subgoal A that is not completed in $\mathcal{S}_i$ and for every ground instance B of A, if $B_H^A \leftarrow \phi$ is a rule in $\frac{P(\mathcal{S}_i)}{J}$ for some instance H of A, then $B \leftarrow \psi$ is a rule in $\frac{P}{J}$ such that*

$$Undelay(\phi) \subseteq \psi \text{ and } \psi - Undelay(\phi) \subseteq \cup_{0 \le j < i} I(\mathcal{S}_j)$$

where $Undelay(\phi) = \{B_1 | B_1$ or $(B_1)_{H_1}^{A_1}$ occurs in ϕ for some subgoal A_1 and some instance H_1 of $A_1\}$, and ψ is viewed as a set of atoms.

Lemma 5.3 reflects how the selected atom A_1 in the body of an X-rule G of a subgoal A is processed. Suppose that C is an answer of A_1 with an atom H_1 in the head. An application of POSITIVE RETURN to G using C has two possibilities.

If C has an empty body, then every ground instance of H_1 is in $\cup_{0 \le j < i} I(\mathcal{S}_j)$. An X-rule G' is generated from G by X-resolution with C, where G' is an instance of G, of the form $G\theta$, with $A_1\theta$ deleted, and θ is the most general unifier of A_1 with (a new variant of) H_1. The condition $\psi - Undelay(\phi) \subseteq \cup_{0 \le j < i} I(\mathcal{S}_j)$ means that every atom in ψ that does not occur in ϕ must be (true) in $\cup_{0 \le j < i} I(\mathcal{S}_j)$.

If C has some delayed literals in its body, then an X-rule G' is generated from G by X-factoring with C, where G' is an instance of G, of the form $G\theta$, except that $A_1\theta$ is repalced with $(A_1\theta)_{H_1}^{A_1}$, and θ is the most general unifier of A_1 with (a new variant of) H_1. The condition $Undelay(\phi) \subseteq \psi$ means that every atom that is delayed or unsolved in ϕ must come from ψ.

Lemma 5.3 is used in the next subsection to establish one direction in the symmetry of $LPM(\frac{P \cup P(\mathcal{S}_i)}{J})$, especially in the case of POSITIVE RETURN which introduces a new rule in $P(\mathcal{S}_i)$.

Proof of Lemma 5.3: The proof is based upon an induction on i. The basis case, $i = 0$, is trivial since $\mathcal{S}_0$ is the empty system and has no subgoals.

Let i be a successor ordinal $\beta + 1$. Then $\mathcal{S}_i$ is obtained from $\mathcal{S}_\beta$ by one of the transformations. The cases of COMPLETION, SIMPLIFICATION, and ANSWER COMPLETION are trivial since they affect only X-rules of subgoals that are completed in $\mathcal{S}_i$. The case of NEW SUBGOAL follows from the use of the most general unifier in X-resolution in deriving the initial X-rules of a new subgoal. If $\mathcal{S}_i$ is obtained from $\mathcal{S}_\beta$ by NEGATIVE RETURN or DELAYING, then $\frac{P(\mathcal{S}_i)}{J} = \frac{P(\mathcal{S}_\beta)}{J}$ holds following the same argument in the proof of Lemma 5.2, based upon the assumption that J is partially symmetric and $\cup_{0 \leq j < i} I(\mathcal{S}_j) \subseteq J$. The lemma holds by inductive hypothesis.

For the case of POSITIVE RETURN, let A be a subgoal in $\mathcal{S}_\beta$ and G be an active X-rule of A corresponding to an ordinal γ, and let A_1 be the selected atom of G. Let C be an answer of subgoal A_1 in $\mathcal{S}_\beta$. If C has an empty body, let G' be the X-resolvent of G with C; if C has some delayed literals in its body, let G' be the X-factor of G with C. Let H be the head atom of G and H' be the head atom of G', and B be a ground instance of H'.

Suppose that $B^A_{H'} \leftarrow \phi'$ is a rule in $\frac{P(\mathcal{S}_i)}{J}$ that is obtained from a ground instance of $(G')^A$. Then by the definition of POSITIVE RETURN, there exists a rule, $B^A_H \leftarrow \phi$, in $\frac{P(\mathcal{S}_\beta)}{J}$ that is obtained from a ground instance of G^A such that $Undelay(\phi') \subseteq Undelay(\phi)$ and $\phi - Undelay(\phi') \subseteq I(\mathcal{S}_\beta)$. By inductive hypothesis, there exists a rule, $B \leftarrow \psi$, in $\frac{P}{J}$ such that $Undelay(\phi) \subseteq \psi$ and $\psi - Undelay(\phi) \subseteq \cup_{0 \leq j < \beta} I(\mathcal{S}_j)$. Therefore $Undelay(\phi') \subseteq \psi$ and $\psi - Undelay(\phi') \subseteq \cup_{0 \leq j \leq \beta} I(\mathcal{S}_j)$. The lemma holds.

Let i be a limit ordinal. If A is a subgoal in $\mathcal{S}_i$ that is not completed, then A is a subgoal in $\mathcal{S}_\beta$ for some $\beta < i$, and is not completed in every $\mathcal{S}_\beta$ such that $\beta < i$ and A is a subgoal in $\mathcal{S}_\beta$. By definition, every X-rule G of A that is not disposed in $\mathcal{S}_i$ must be an X-rule of A that is not disposed in $\mathcal{S}_\beta$ for some $\beta < i$. The lemma follows by inductive hypothesis. □

Lemma 5.4 *Let P be a finite program, R be an arbitrary but fixed computation rule, and Q be an atomic query. Let $\mathcal{S}_0, \mathcal{S}_1, ..., \mathcal{S}_\alpha$ be an arbitrary SLG derivation of Q, where α is an ordinal. Then for every $i (0 \leq i \leq \alpha)$ and for every partially symmetric interpretation J of $P \cup P(\mathcal{S}_i)$ such that $\cup_{0 \leq j < i} I(\mathcal{S}_j) \subseteq J$, and for subgoal A in $\mathcal{S}_i$ that is not completed, let $(A : \rho) \in \mathcal{S}_i$ and the following holds: For every X-rule G of A corresponding to some ordinal α in $\mathcal{S}_i$, of the form*

$$H \leftarrow Left, A_1, Right$$

where A_1 is the selected atom of G, and for every atom H_1 such that $G'\langle \alpha, H_1 \rangle$ is an X-element in ρ for some X-rule G', there exists some non-disposed X-rule G^* of A in $\mathcal{S}_i$ such that for every rule in $\frac{P(\mathcal{S}_i)}{J}$ of the form

$$B^A_H \leftarrow \phi_{left}, B_1, \phi_{right}$$

obtained from a ground instance of G^A such that B_1 is a ground instance of H_1 (and A_1), there exists a rule in $\frac{P(\mathcal{S}_i)}{J}$, obtained from a ground instance of $(G^*)^A$, of the form

$$B^A_{H'} \leftarrow \phi_{left}, \phi_{right}$$

or

$$B^{A}_{H'} \leftarrow \phi_{left}, (B_1)^{A_1}_{H_1}, \phi_{right}$$

where H' is the head atom of G^*.

Recall that in the sequence ρ of annotated X-rules associated with a subgoal A, $G'\langle\alpha, H_1\rangle$ means that POSITIVE RETURN has been applied to an X-rule of A corresponding to some ordinal α by using an answer with H_1 in the head. In Lemma 5.4, the X-rule G^* is either G' or some X-rule that is derived, directly or indirectly, from G' by solving or delaying some negative literals in the body of G'. Lemma 5.4 describes essentially the relationship between an active X-rule G that has a selected atom A_1 and X-rules that are derived from G by POSITIVE RETURN. It is used in the next subsection to prove the correctness of COMPLETION.

Proof of Lemma 5.4: The proof is based upon an induction on i. The basis case, $i = 0$, is trivial.

Let i be a successor ordinal $\beta + 1$. Then $\mathcal{S}_i$ is obtained from $\mathcal{S}_\beta$ by one of the transformations. For the case of NEW SUBGOAL, the initial X-rules of a new subgoal, say A, are derived by X-resolution with rules in the program P. Transformation POSITIVE RETURN has not been applied to any initial X-rule of A that has a selected atom. The lemma holds by inductive hypothesis. The cases of COMPLETION, SIMPLIFICATION, and ANSWER COMPLETION hold by inductive hypothesis, because they affect only X-rules of subgoals that are completed in $\mathcal{S}_i$. For the cases of NEGATIVE RETURN and DELAYING, since J is partially symmetric and $\cup_{0\le j\le\beta} I(\mathcal{S}_j) \subseteq J$, it can be verified that $\frac{P(\mathcal{S}_i)}{J} = \frac{P(\mathcal{S}_\beta)}{J}$. Thus the lemma holds by inductive hypothesis.

For POSITIVE RETURN, let G be an active X-rule of A in $\mathcal{S}_\beta$, of the form

$$H \leftarrow Left, A_1, Right$$

where A_1 is the selected atom. Let C be an answer of A_1 with an atom H_1 in the head in $\mathcal{S}_\beta$ such that POSITIVE RETURN is applied to G by using C when $\mathcal{S}_i$ is derived from $\mathcal{S}_\beta$. Let G' be the X-resolvent of G with C if C has an empty body, and be the X-factor of G with C if C has some delayed literals in its body. Then G' satisfies the properties of G^* as specified in the lemma, and the lemma holds by inductive hypothesis.

If i is a limit ordinal, let A be a subgoal in $\mathcal{S}_i$ that is not completed and let $(A : \rho) \in \mathcal{S}_i$. Let G be an active X-rule of A in $\mathcal{S}_i$ with a selected atom A_1, and H_1 be an instance of A_1 such that $G'\langle\alpha, H_1\rangle$ is an element of ρ for some X-rule G'. Then for some $\beta < i$, both G and G' are non-disposed X-rules of A in $\mathcal{S}_\beta$.

Let G^* be an X-rule with the fewest negative literals that are not delayed in its body such that for some $\beta < i$,

- G^* is a non-disposed X-rule of A in $\mathcal{S}_\beta$; and
- for every rule

  $$B^{A}_{H} \leftarrow \phi_{left}, B_1, \phi_{right}$$

  in $\frac{P(\mathcal{S}_\beta)}{J}$, obtained from a ground instance of G^A, there exists a rule

  $$B^{A}_{H'} \leftarrow \phi_{left}, \phi_{right}$$

or

$$B^A_{H'} \leftarrow \phi_{left}, (B_1)^{A_1}_{H_1}, \phi_{right}$$

in $\frac{P(\mathcal{S}_\beta)}{J}$, obtained from a ground instance of $(G^*)^A$, where H' is the head atom of G^*.

The existence of G^* is guaranteed by G'.

If G^* has a selected atom or is an answer, then G^* remains in $\mathcal{S}_i$ since A is not completed in $\mathcal{S}_i$, in which case the lemma holds by inductive hypothesis. Otherwise, G^* has a selected negative literal. Either G^* remains in $\mathcal{S}_i$, in which case the lemma holds by inductive hypothesis; or NEGATIVE RETURN or DELAYING has been applied to G^*. In the latter case, since J is partially symmetric and $\cup_{0 \leq j \leq \beta} I(\mathcal{S}_j) \subseteq J$, there exists some ordinal γ, where $\beta < \gamma < i$, and some X-rule in $\mathcal{S}_\gamma$ that satisfies all the properties of G^*, but has one less negative literal that is not delayed, a contradiction. □

5.2 Preservation of Three-Valued Stable Models

The following key theorem shows that every system in an SLG derivation for an atomic query Q is a symmetric system.

Theorem 5.5 *Let P be a finite program, R be an arbitrary but fixed computation rule, and Q be an atomic query. Let $\mathcal{S}_0, \mathcal{S}_1, ..., \mathcal{S}_\alpha$ be an arbitrary SLG derivation for Q, where α is an ordinal. Then for every $i (0 \leq i \leq \alpha)$, $I(\mathcal{S}_i) \subseteq \mathcal{WF}(P \cup P(\mathcal{S}_i))$ and $\mathcal{S}_i$ is a symmetric system.*

Proof: The proof is based upon an induction on i. For the basis case, $i = 0$, $\mathcal{S}_0$ is the empty system and $I(\mathcal{S}_0)$ is the empty set and $P(\mathcal{S}_0)$ is the empty program. The lemma holds trivially.

For the inductive case, we prove the following:

(a) $LPM(\frac{P \cup P(\mathcal{S}_i)}{J})$ is symmetric for every partially symmetric interpretation J of $P \cup P(\mathcal{S}_i)$ such that $\cup_{0 \leq j < i} I(\mathcal{S}_j) \subseteq J$;

(b) $I(\mathcal{S}_i) \subseteq \mathcal{WF}(P \cup P(\mathcal{S}_i))$.

Let J be an arbitrary partially symmetric interpretation of $P \cup P(\mathcal{S}_i)$ such that $I(\mathcal{S}_i) \subseteq J$. Then $\cup_{0 \leq j < i} I(\mathcal{S}_j) \subseteq J$ by Lemma 5.1. Thus (a) implies that $\mathcal{S}_i$ is a symmetric system. We show that (a) implies (b) and then prove (a).

(a) $\Rightarrow$ (b). By inductive hypothesis, $I(\mathcal{S}_j) \subseteq \mathcal{WF}(P \cup P(\mathcal{S}_j))$ for every $j (0 \leq j < i)$. Since P is independent of $P(\mathcal{S}_j)$ for every $j (0 \leq j \leq i)$, and $I(\mathcal{S}_j) \subseteq I(\mathcal{S}_i)$ by Lemma 5.1, it follows that

$$\cup_{0 \leq j < i} I(\mathcal{S}_j) \subseteq \mathcal{WF}(P \cup P(\mathcal{S}_i))$$

We construct a partially symmetric interpretation J of $P \cup P(\mathcal{S}_i)$ as follows:

- $J|_P = \mathcal{WF}(P)$;
- for every ground subgoal B in $\mathcal{S}_i$, $J(B) = J(B^B_B)$;
- for every subgoal A in $\mathcal{S}_i$ and every instance H of A and every ground instance B of H, if $B^A_H \in \cup_{0 \leq j < i} I(\mathcal{S}_j)$, then $B^A_H \in J$, and if $\sim B^A_H \in \cup_{0 \leq j < i} I(\mathcal{S}_j)$, then $\sim B^A_H \in J$.

Clearly $\cup_{0 \leq j < i} I(\mathcal{S}_j) \subseteq J$. The existence of J is ensured by the fact that $\cup_{0 \leq j < i} I(\mathcal{S}_j) \subseteq \mathcal{WF}(P \cup P(\mathcal{S}_i))$.

Let $M = LPM(\frac{P \cup P(\mathcal{S}_i)}{J})$. Since $J|_P = \mathcal{WF}(P)$ and P is independent of $P(\mathcal{S}_i)$, $M|_P = \mathcal{WF}(P)$ as $\mathcal{WF}(P)$ is a three-valued stable model of P. By (a), M is symmetric. By the definition of $I(\mathcal{S}_i)$, every literal of the form B_H^A or $\sim B_H^A$ in $I(\mathcal{S}_i)$ is in M and in $\mathcal{WF}(P \cup P(\mathcal{S}_i))$, where A is a subgoal in $\mathcal{S}_i$, H is an instance of A, and B is a ground instance of H. Since M is symmetric, $I(\mathcal{S}_i)|_P \subseteq M|_P = \mathcal{WF}(P)$. Thus $I(\mathcal{S}_i) \subseteq \mathcal{WF}(P \cup P(\mathcal{S}_i))$.

Now that we have established that (a) implies (b), we prove (a). Let i be a successor ordinal $\beta + 1$. Then $\mathcal{S}_i$ is obtained from $\mathcal{S}_\beta$ by one of the transformations. By Lemma 5.1, $I(\mathcal{S}_\beta) = \cup_{0 \leq j < i} I(\mathcal{S}_j)$. Let $M_1 = LPM(\frac{P \cup P(\mathcal{S}_\beta)}{J})$ and $M_2 = LPM(\frac{P \cup P(\mathcal{S}_i)}{J})$. By inductive hypothesis, M_1 is a symmetric interpretation of $P \cup P(\mathcal{S}_\beta)$. We prove that M_2 is symmetric by a case analysis of the transformations.

<u>NEW SUBGOAL</u>: Suppose that A is a new subgoal that is introduced and B is an arbitrary ground instance of A. Then the Herbrand instantiation of P contains a rule of the form $B \leftarrow \phi$ if and only if the Herbrand instantiation of $P(\mathcal{S}_i)$ contains a rule of the form $B_H^A \leftarrow \phi$ for some instance H of A. Therefore M_2 is symmetric on A. Subgoals in $\mathcal{S}_\beta$ are not affected, and (a) holds by inductive hypothesis.

<u>NEGATIVE RETURN</u> and <u>DELAYING</u>: Since J is a partially symmetric interpretation of $P \cup P(\mathcal{S}_i)$ and $\cup_{0 \leq j < i} I(\mathcal{S}_j) \subseteq J$, it can be verified that $\frac{P(\mathcal{S}_\beta)}{J} = \frac{P(\mathcal{S}_i)}{J}$. Thus $M_2 = M_1$ and (a) holds by inductive hypothesis.

<u>SIMPLIFICATION</u>: Let A be a subgoal that is completed in $\mathcal{S}_\beta$ and G be an answer of A that is not disposed. Let L be a delayed literal in the body of G. If L is a negative delayed literal, it can be verified that $\frac{P(\mathcal{S}_\beta)}{J} = \frac{P(\mathcal{S}_i)}{J}$, based upon the assumption on J. Thus $M_2 = M_1$ and (a) holds by inductive hypothesis.

If L is a positive delayed literal of the form $B_{H_1}^{A_1}$, where A_1 is a subgoal in $\mathcal{S}_\beta$, H_1 is an instance of A_1, and B is an instance of H_1, there are two cases. If L is successful, then A_1 has an answer C in $\mathcal{S}_\beta$ that has H_1 in the head and an empty body. Then L is deleted from the body of G. Clearly for every ground instance h of B, $h_{H_1}^{A_1}$ can always be derived using C^{A_1} in $P(\mathcal{S}_i)$. Thus $M_2 = M_1$ and (a) holds by inductive hypothesis. The case that L is failed is similar.

<u>ANSWER COMPLETION</u>: Let U be the set of all pairs (A, H) in $\mathcal{S}_\beta$ such that A is a subgoal and H is the head atom of some answer of A and H is not supported by A. Then $\mathcal{S}_i$ is obtained from $\mathcal{S}_\beta$ by deleting all the answers of A that have H in the head, for some $(A, H) \in U$.

By definition, for every pair $(A, H) \in U$, A is completed, and for every answer G of A that has H in the head, there exists a positive delayed literal in the body of G, of the form $(B_1)_{H_1}^{A_1}$, where H_1 is not supported by A_1. Then for every ground instance B of H, $\sim B_H^A \in M_1$ and $\sim B_H^A \in M_2$. Hence $M_2 = M_1$ and (a) holds by inductive hypothesis.

<u>POSITIVE RETURN</u>: First, POSITIVE RETURN does not affect subgoals that are completed in $\mathcal{S}_\beta$. In particular, for every completed subgoal A in $\mathcal{S}_\beta$ and every answer C of A that is not disposed in $\mathcal{S}_\beta$, and for every positive delayed literal in the body of C, of the form $B_{H_1}^{A_1}$, A_1 is also completed. The reason is that a positive delayed literal is created by POSITIVE RETURN from an active X-rule with a selected atom, and an active X-rule of a subgoal with a selected atom is disposed only by COMPLETION. By inductive hypothesis, M_2 remains symmetric on all completed subgoals in $\mathcal{S}_i$, which are precisely completed subgoals in $\mathcal{S}_\beta$.

Second, let A be an arbitrary subgoal in $\mathcal{S}_i$ that is not completed, and let B be an arbitrary ground instance of A. By Lemma 5.2, $B \in M_2$ implies $B^A_H \in M_2$ for some instance H of A, and if $\sim B^A_H \in M_2$ for every instance H of A, then $\sim B \in M_2$.

For the other direction, let $P' = \frac{P \cup P(\mathcal{S}_i)}{J}$. Recall that $M_2 = \tau^{\uparrow\omega}_{P'}$. We show by induction on k that for every $k \geq 0$, and for every subgoal A in $\mathcal{S}_i$ and every ground instance B of A, if $B^A_H \in Und(\tau^{\uparrow k}_{P'})$ for some instance H of A, then $B \in Und(M_2) \cup Pos(M_2)$, and if $B^A_H \in Pos(\tau^{\uparrow k}_{P'})$ for some instance H of A, then $B \in Pos(M_2)$.

The basis case, $k = 0$, is trivial since $\tau^{\uparrow 0}_{P'} = \emptyset$, in which every ground atom is false. For the inductive case, $k+1$, consider any rule of the form $B^A_H \leftarrow \phi$ in P' for some instance H of A. By Lemma 5.3, there is a rule $B \leftarrow \psi$ in P' such that $Undelay(\phi) \subseteq \psi$ and $\psi - Undelay(\phi) \subseteq \cup_{0 \leq j < i} I(\mathcal{S}_j) = I(\mathcal{S}_\beta)$. By the definition of $I(\mathcal{S}_\beta)$, every literal in $I(\mathcal{S}_\beta)$ of the form $(B_1)^{A_1}_{H_1}$ or $\sim(B_1)^{A_1}_{H_1}$ is in M_1, where A_1 is a subgoal in $\mathcal{S}_\beta$ and H_1 is an instance of A_1 and B_1 is a ground instance of B_1. Since M_1 is symmetric by inductive hypothesis, every ground atom in $\psi - Undelay(\phi)$ is in M_1. As $M_1|_P = M_2|_P$, every ground atom in $\psi - Undelay(\phi)$ is in M_2 too. If $B^A_H \in Und(\tau^{\uparrow k+1}_{P'})$ due to a rule $B^A_H \leftarrow \phi$, then B is in $Und(M_1) \cup Pos(M_2)$ due to the rule $B \leftarrow \psi$ by inductive hypothesis. Similarly, if $B^A_H \in Pos(\tau^{\uparrow k+1}_{P'})$, then $B \in Pos(M_2)$.

This concludes the induction on k. Thus for every subgoal A that is not completed in $\mathcal{S}_i$ and for every ground instance B of A, if $B^A_H \in M_2$ for some instance H of A, then $B \in M_2$, and if $\sim B \in M_2$, then $\sim B^A_H \in M_2$ for every instance H of A.

This concludes the proof that M_2 is symmetric, and so (a) holds.

COMPLETION: Following the same argument as in POSITIVE RETURN, COMPLETION does not affect subgoals that are completed in $\mathcal{S}_\beta$. In particular, M_2 and M_1 coincide on all literals of the form B^A_H or $\sim B^A_H$, where A is a subgoal that is completed in $\mathcal{S}_\beta$. In addition, $M_1|_P = M_2|_P$. By inductive hypothesis, M_2 remains symmetric on all subgoals that are completed in $\mathcal{S}_\beta$.

By definition, COMPLETION disposes all active X-rules of some subgoals (that are not answers), and so $P(\mathcal{S}_i)$ can be obtained from $P(\mathcal{S}_\beta)$ by deleting some rules. Therefore $M_2 \preceq M_1$ (with respect to the truth ordering). Lemma 5.2 together with $M_2 \preceq M_1$ implies by inductive hypothesis that M_2 is symmetric on all subgoals that are not completed in $\mathcal{S}_i$.

Let Λ be a non-empty set of subgoals that are completely evaluated in $\mathcal{S}_\beta$ such that all active X-rules of subgoals in Λ are disposed by COMPLETION. It remains to show that M_2 is symmetric on subgoals in Λ. Let $P' = \frac{P \cup P(\mathcal{S}_i)}{J}$. Since $M_2 \preceq M_1$ and $M_1|_P = M_2|_P$, it suffices to prove that for every $k \geq 0$, and for every subgoal $A \in \Lambda$ and every ground instance B of A,

(1) if $B \in Und(\tau^{\uparrow k}_{P'})$, then $B^A_H \in Und(\tau^{\uparrow k}_{P'}) \cup Pos(\tau^{\uparrow k}_{P'})$ for some instance H of A; and

(2) if $B \in Pos(\tau^{\uparrow k}_{P'})$, then $B^A_H \in Pos(\tau^{\uparrow k}_{P'})$ for some instance H of A.

The basis case, $k = 0$, is trivial since $\tau^{\uparrow 0}_{P'} = \emptyset$. For the inductive case, suppose that $B \in Und(\tau^{\uparrow k+1}_{P'}) \cup Pos(\tau^{\uparrow k+1}_{P'})$, and the derivation of B uses a rule of the form $B \leftarrow \phi \in P'$. However, $B \leftarrow \phi$ is also a rule in $\frac{P \cup P(\mathcal{S}_\beta)}{J}$. By Lemma 5.2, $B^A_H \leftarrow \phi$ is a rule in $\frac{P \cup P(\mathcal{S}_\beta)}{J}$.

If $B^A_H \leftarrow \phi$ is a rule in P', then (1) and (2) hold by inductive hypothesis. Otherwise, since A is in Λ, either A succeeds, in which case (1) and (2) hold, or A has an active X-rule G corresponding to some ordinal α in $\mathcal{S}_\beta$ of the form

$$H \leftarrow Left, A_1, Right$$

with a selected atom A_1, and $B_H^A \leftarrow \phi$ is obtained from a ground instance of G^A and is of the form

$$B_H^A \leftarrow \phi_{left}, B_1, \phi_{right}$$

where B_1 is a ground instance of A_1. By assumption on B, $B_1 \in Und(\tau_{P'}^{\uparrow k}) \cup Pos(\tau_{P'}^{\uparrow k})$. By inductive hypothesis on k, (1) and (2) hold for B_1 and $(B_1)_{H_1}^{A_1}$ for some instance H_1 of A_1. Since A is completed in $\mathcal{S}_i$, H_1 must be the head atom of some answer of A in $\mathcal{S}_i$ that is not disposed. By the definition of Λ, the X-sequence of A in $\mathcal{S}_\beta$ must contain an X-element of the form $G'\langle\alpha, H_1\rangle$ for some X-rule G'. By Lemma 5.4, there exists some instance H' of A such that

$$B_{H'}^A \leftarrow \phi_{left}, \phi_{right}$$

or

$$B_{H'}^A \leftarrow \phi_{left}, (B_1)_{H_1}^{A_1}, \phi_{right}$$

is a rule in $\frac{P(\mathcal{S}_\beta)}{J}$.

The number of positive literals in the body of an active X-rule is bounded by the maximum number of literals in the body of a rule in P, which is finite. By repeatedly applying the same argument for $B_H^A \leftarrow \phi$ to $B_{H'}^A \leftarrow \phi_{left}, (B_1)_{H_1}^{A_1}, \phi_{right}$, we will eventually obtain some $B_{H^*}^A$ for some instance H^* of A such that (1) and (2) hold for B and $B_{H^*}^A$. This concludes the induction on k.

This concludes the proof for the case of COMPLETION.

The other inductive case for i is that i is a limit ordinal other than 0. Let $M = LPM(\frac{P \cup P(\mathcal{S}_i)}{J})$.

Let A be a subgoal that is completed in $\mathcal{S}_i$. Then A must be completed in $\mathcal{S}_\beta$ for some $\beta < i$, and A remains completed in $\mathcal{S}_j$ for all $j(\beta \leq j < i)$. After A becomes completed in $\mathcal{S}_\beta$, the only transformations that can be applied to A are SIMPLIFICATION and ANSWER COMPLETION, which either delete some answers of A or deletes some successful delayed literals in the body of an answer of A. The following two properties hold:

- First, the answers of A in $\mathcal{S}_i$ that are not disposed can be obtained from those of A in $\mathcal{S}_\beta$ by repeatedly applying SIMPLIFICATION and ANSWER COMPLETION.

- Second, by the argument for the case of a successor ordinal, $LPM(\frac{P \cup P(\mathcal{S}_j)}{J})$ and $LPM(\frac{P \cup P(\mathcal{S}_{j+1})}{J})$ coincide on all ground literals of the form B_H^A or $\sim B_H^A$, where $\beta \leq j < i$ and A is a subgoal that is completed in $\mathcal{S}_\beta$.

Thus M coincides with $LPM(\frac{P \cup P(\mathcal{S}_\beta)}{J})$ on all literals of the form B_H^A or $\sim B_H^A$, where A is a subgoal in $\mathcal{S}_i$ that is completed, H is an instance of A and B is a ground instance of H.

Let A be a subgoal that is not completed in $\mathcal{S}_i$, and let B be an arbitrary ground instance of A. By Lemma 5.2, $B \in M_2$ implies $B_H^A \in M_2$ for some instance H of A, and if $\sim B_H^A \in M_2$ for every instance H of A, then $\sim B \in M_2$. For the other direction, let $P' = \frac{P \cup P(\mathcal{S}_i)}{J}$. The argument is the same as in the case of POSITIVE RETURN, by using Lemma 5.3 and an induction on k in $\tau_{P'}^{\uparrow k}$, with an additional observation. That is, for each B_H^A in $Und(\tau_{P'}^{\uparrow k}) \cup Pos(\tau_{P'}^{\uparrow k})$, there is a corresponding derivation represented as a finite sequence of rules $r_0, ..., r_l$ in P' such that

- the head of r_l is B_H^A; and

- for each $r_j (0 \leq j \leq l)$, every atom in the body of r_j is either $\mathbf{u}$ or is the head of $r_{j'}$ for some $j' < j$.

Since l is finite, there exists some $\beta < i$ such that every rule $r_j (0 \leq j \leq l)$ is a rule in $\frac{P \cup P(\mathcal{S}_\beta)}{J}$. Then B_H^A is in $Und(LPM(\frac{P \cup P(\mathcal{S}_\beta)}{J})) \cup Pos(LPM(\frac{P \cup P(\mathcal{S}_\beta)}{J}))$. The inductive step in the induction on k holds since $LPM(\frac{P \cup P(\mathcal{S}_\beta)}{J})$ is symmetric by the inductive hypothesis.

□

Theorem 4.1 shows that some final system $\mathcal{S}$ can be derived for an atomic query Q, given a finite program P and an arbitrary but fixed computation rule R. By Lemma 4.2, $\mathcal{S}$ is either completed, i.e., every subgoal in $\mathcal{S}$ is completed with only answers, or floundered, i.e., some subgoal in $\mathcal{S}$ has an active X-rule with a selected non-ground negative literal. The following theorem shows by using Theorem 5.5 that every three-valued stable model of P is preserved when a completed system is reached. We discuss later in Section 7.2 how floundering may be avoided by imposing certain conditions on a program and the computation rule.

Theorem 5.6 *Let P be a finite program, R be an arbitrary but fixed computation rule, and Q be an atomic query, and $\mathcal{S}$ be a final system for Q that is completed. Then*

(a) for every $I \in \mathcal{ST}3(P)$, there exists a symmetric interpretation M of $P \cup P(\mathcal{S})$ such that $M|_P = I$ and $M|_{P(\mathcal{S})} \in \mathcal{ST}3(P(\mathcal{S}))$;

(b) for every $I \in \mathcal{ST}3(P(\mathcal{S}))$, there exists a symmetric interpretation M of $P \cup P(\mathcal{S})$ such that $M|_{P(\mathcal{S})} = I$ and $M|_P \in \mathcal{ST}3(P)$.

Proof: Since $\mathcal{S}$ is a final system that is completed, all X-rules of subgoals in $\mathcal{S}$ that are not disposed are answers. Thus $P(\mathcal{S})$ and P are independent of each other. By Theorem 5.5, $\mathcal{S}$ is a symmetric system.

For (a), let $I \in \mathcal{ST}3(P)$. By Theorem 2.3, $\mathcal{WF}(P) \subseteq I$. By Theorem 5.5, $I(\mathcal{S}) \subseteq \mathcal{WF}(P \cup P(\mathcal{S}))$. Since P and $P(\mathcal{S})$ are independent of each other, $\mathcal{WF}(P \cup P(\mathcal{S})) = \mathcal{WF}(P) \cup \mathcal{WF}(P(\mathcal{S}))$. Thus $I(\mathcal{S})|_P \subseteq \mathcal{WF}(P) \subseteq I$. We construct a partially symmetric interpretation J of $P \cup P(\mathcal{S})$ as follows:

- $J|_P = I$; and
- for every ground subgoal B in $\mathcal{S}$, $J(B) = J(B_B^B)$; and
- $I(\mathcal{S})|_{P(\mathcal{S})} \subseteq J$.

The existence of J is ensured by the fact that $I(\mathcal{S}) \subseteq \mathcal{WF}(P \cup P(\mathcal{S}))$ and $\mathcal{WF}(P) \subseteq I$.

Notice that $I(\mathcal{S}) \subseteq J$. Let $M = LPM(\frac{P \cup P(\mathcal{S})}{J})$. By Theorem 5.5, $\mathcal{S}$ is a symmetric system and so M is symmetric. Since P and $P(\mathcal{S})$ are independent of each other, $M|_P = LPM(\frac{P}{J|_P})$ and $M|_{P(\mathcal{S})} = LPM(\frac{P(\mathcal{S})}{J|_{P(\mathcal{S})}})$. As $J|_P = I \in \mathcal{ST}3(P)$, $M|_P = I$.

Both M and J are partially symmetric and $M|_P = J|_P = I$. Thus for every ground subgoal B in $\mathcal{S}$, $M(B_B^B) = J(B_B^B) = I(B)$. Since $\mathcal{S}$ is completed, all negative literals occurring in $P(\mathcal{S})$ are of the form $\sim B_B^B$. Thus $\frac{P(\mathcal{S})}{J|_{P(\mathcal{S})}} = \frac{P(\mathcal{S})}{M|_{P(\mathcal{S})}}$. Hence $M|_{P(\mathcal{S})} = LPM(\frac{P(\mathcal{S})}{J|_{P(\mathcal{S})}}) = LPM(\frac{P(\mathcal{S})}{M|_{P(\mathcal{S})}})$, and so $M|_{P(\mathcal{S})} \in \mathcal{ST}3(P(\mathcal{S}))$.

This concludes the proof for Part (a) of the lemma, and we now show that Part (b) of the lemma holds. Let $I \in \mathcal{ST}3(P(\mathcal{S}))$. First, we show that there exists a symmetric interpretation M_0 of $P \cup P(\mathcal{S})$ such that $M|_{P(\mathcal{S})} = I$.

By Theorem 2.3, $\mathcal{WF}(P(\mathcal{S})) \subseteq I$. By Theorem 5.5, $I(\mathcal{S}) \subseteq \mathcal{WF}(P \cup P(\mathcal{S}))$. Since P and $P(\mathcal{S})$ are independent of each other, $\mathcal{WF}(P \cup P(\mathcal{S})) = \mathcal{WF}(P) \cup \mathcal{WF}(P(\mathcal{S}))$. Thus $I(\mathcal{S})|_{P(\mathcal{S})} \subseteq \mathcal{WF}(P(\mathcal{S})) \subseteq I$. Let J be an interpretation of $P \cup P(\mathcal{S})$ such that

- $Pos(J) = Pos(I(\mathcal{S})) \cup \{B, B_B^B | B_B^B \in Pos(I)$ for some ground subgoal B in $\mathcal{S}\}$; and
- $Neg(J) = Neg(I(\mathcal{S})) \cup \{B, B_B^B | B_B^B \in Neg(I)$ for some ground subgoal B in $\mathcal{S}\}$.

The existence of J is ensured by the fact that $I(\mathcal{S}) \subseteq \mathcal{WF}(P \cup P(\mathcal{S}))$. Also notice that J is partially symmetric and $I(\mathcal{S}) \subseteq J$.

Let $M_0 = LPM(\frac{P \cup P(\mathcal{S})}{J})$. By Theorem 5.5, M_0 is symmetric. Since P and $P(\mathcal{S})$ are independent of each other, $M_0|_{P(\mathcal{S})} = LPM(\frac{P(\mathcal{S})}{J|_{P(\mathcal{S})}})$. Notice that for every ground subgoal B in $\mathcal{S}$, $J(B_B^B) = I(B_B^B)$. Therefore $\frac{P(\mathcal{S})}{J|_{P(\mathcal{S})}} = \frac{P(\mathcal{S})}{I}$ and so $M_0|_{P(\mathcal{S})} = LPM(\frac{P(\mathcal{S})}{I})$. Since $I \in \mathcal{ST}3(P(\mathcal{S}))$, $M_0|_{P(\mathcal{S})} = I$.

We partition the Herbrand universe $\mathcal{HB}_P$ into $\mathcal{H}_1 \cup \mathcal{H}_2$, where $\mathcal{H}_1$ is the set of all atoms $B \in \mathcal{HB}_P$ such that B is a ground instance of a subgoal in $\mathcal{S}$, and $\mathcal{H}_2 = \mathcal{HB}_P - \mathcal{H}_1$. We construct a symmetric interpretation M of $P \cup P(\mathcal{S})$ as follows:

- $M|_{P(\mathcal{S})} = I$; and
- $M(B) = M_0(B)$ for every $B \in \mathcal{H}_1$, where B is a ground instance of a subgoal in $\mathcal{S}$.

For atoms in $\mathcal{H}_2$, their truth values in M are chosen as follows. We construct a program P_{simpl} from the Herbrand instantiation of P by

- deleting every rule whose head is an atom in $\mathcal{H}_1$;
- deleting every rule whose body contains a positive literal B such that $B \in \mathcal{H}_1$ and $M(B) = \mathbf{f}$;
- deleting every rule whose body contains a negative literal $\sim B$ such that $B \in \mathcal{H}_1$ and $M(B) = \mathbf{t}$;
- replacing each positive literal B in the body of a rule with $\mathbf{u}$ if $B \in \mathcal{H}_1$ and $M(B) = \mathbf{u}$;
- replacing each negative literal $\sim B$ in the body of a rule with $\mathbf{u}$ if $B \in \mathcal{H}_1$ and $M(B) = \mathbf{u}$.

Consider each ground atom in P_{simpl} as a new propositional symbol, and let M_1 be an arbitrary three-valued stable model of P_{simpl} viewed as a propositional program. Then for every atom $B \in \mathcal{H}_2$,

- if B occurs in P_{simpl}, $M(B) = M_1(B)$;
- otherwise, $M(B) = \mathbf{f}$.

Notice that M is a symmetric interpretation of $P \cup P(\mathcal{S})$ such that $M|_{P(\mathcal{S})} = I$ and $I(\mathcal{S}) \subseteq M$. Let $M' = LPM(\frac{P \cup P(\mathcal{S})}{M})$. By Theorem 5.5, M' is symmetric. We show that $M = M'$. First, since $M|_{P(\mathcal{S})} = I$ and $I \in \mathcal{ST}3(P(\mathcal{S}))$ and P and $P(\mathcal{S})$ are independent of each other, $M'|_{P(\mathcal{S})} = I = M|_{P(\mathcal{S})}$. Second, for every atom $B \in \mathcal{H}_1$, $M'(B) = M(B)$ as both M and M' are symmetric and $M'|_{P(\mathcal{S})} = M|_{P(\mathcal{S})}$. Third, for every atom $B \in \mathcal{H}_2$, $M'(B) = M(B)$, which can be verified by the construction of P_{simpl} and the usage of a three-valued stable model M_1 of P_{simpl} in the definition of M. Thus $M = M'$.

Since $M = M' = LPM(\frac{P \cup P(\mathcal{S})}{M})$ and P and $P(\mathcal{S})$ are independent of each other, $M|_P = LPM(\frac{P}{M|_P})$, which implies that $M|_P \in \mathcal{ST}3(P)$. Furthermore, $M|_{P(\mathcal{S})} = I$ and M is symmetric, and so (b) holds.

$\square$

5.3 Computation of the Well-Founded Partial Model

The primary purpose of SLG resolution is to compute answers of a query with respect to the well-founded partial model of a finite program. Let $\mathcal{S}$ be a final and completed system that is derived for an atomic query with respect to a finite program P. We show that $\mathcal{WF}(P)$ coincides with $\mathcal{WF}(P(\mathcal{S}))$ as far as ground instances of subgoals in $\mathcal{S}$ are concerned. Moreover, for every ground instance B of any subgoal A in $\mathcal{S}$, B is true in $\mathcal{WF}(P)$ if and only if B is an instance of the head of an answer of A that has an empty body, and B is false in $\mathcal{WF}(P)$ if and only if B is not an instance of the head of any answer of A. In other words, the truth values of ground instances of subgoals relevant to a query can be determined directly from the answers in $\mathcal{S}$, without any further derivation. Finally we show that SLG resolution satisfies the most general answer property in the sense that if for some instance H of an atom Q, $\forall H$ is true in the well-founded partial model, then there is an answer of Q whose head has H as an instance and whose body is empty.

Theorem 5.7 *Let P be a finite program, R be an arbitrary but fixed computation rule, and Q be an atomic query, and $\mathcal{S}$ be a final system for Q that is completed. Then there exists a symmetric interpretation J of $P \cup P(\mathcal{S})$ such that $J|_P = \mathcal{WF}(P)$ and $J|_{P(\mathcal{S})} = \mathcal{WF}(P(\mathcal{S}))$.*

Proof: By Theorem 2.3, $\mathcal{WF}(P) \in \mathcal{ST}3(P)$. By Theorem 5.6, there exists a symmetric interpretation M of $P \cup P(\mathcal{S})$ such that $M|_P = \mathcal{WF}(P)$ and $M|_{P(\mathcal{S})} \in \mathcal{ST}3(P(\mathcal{S}))$. By Theorem 2.3, $\mathcal{WF}(P(\mathcal{S})) \subseteq M|_{P(\mathcal{S})}$. Therefore for every subgoal A in $\mathcal{S}$ and for every ground instance B of A in $\mathcal{S}$,

- if $B_H^A \in \mathcal{WF}(P(\mathcal{S}))$ for some instance H of A, then $B_H^A \in M|_{P(\mathcal{S})}$. Since M is symmetric, $B \in M|_P = \mathcal{WF}(P)$; and
- if $\sim B_H^A \in \mathcal{WF}(P(\mathcal{S}))$ for every instance H of A, then $\sim B_H^A \in M|_{P(\mathcal{S})}$ for every instance H of A. Since M is symmetric, $\sim B \in M|_P = \mathcal{WF}(P)$.

For the other direction, $\mathcal{WF}(P(\mathcal{S})) \in \mathcal{ST}3(P(\mathcal{S}))$. By Theorem 5.6, there exists a symmetric interpretation M of $P \cup P(\mathcal{S})$ such that $M|_{P(\mathcal{S})} = \mathcal{WF}(P(\mathcal{S}))$ and $M|_P \in \mathcal{ST}3(P)$. By Theorem 2.3, $\mathcal{WF}(P) \subseteq M|_P$. Therefore for every subgoal A in $\mathcal{S}$ and for every ground instance B of A in $\mathcal{S}$,

- if $B \in \mathcal{WF}(P)$, then $B \in M|_P$. Since M is symmetric, there exists an instance H of A such that $B^A_H \in M|_{P(\mathcal{S})} = \mathcal{WF}(P(\mathcal{S}))$;
- if $\sim B \in \mathcal{WF}(P)$, then $\sim B \in M|_P$. Since M is symmetric, for every instance H of A, $\sim B^A_H \in M|_{P(\mathcal{S})} = \mathcal{WF}(P(\mathcal{S}))$.

Let J be an interpretation of $P \cup P(\mathcal{S})$ such that $J|_P = \mathcal{WF}(P)$ and $J|_{P(\mathcal{S})} = \mathcal{WF}(P(\mathcal{S}))$. Then J is a symmetric interpretation by the arguments above.

□

Theorem 5.7 says only that the set of answers in a final system $\mathcal{S}$ preserves the well-founded partial model as a whole as far as instances of subgoals relevant to a query are concerned. The following theorem establishes further that the truth values of ground instances of subgoals in the well-founded partial model of the original program can be determined by simply looking at the syntactic format of the set of answers in $\mathcal{S}$, without any further derivation.

Theorem 5.8 *Let P be a finite program, R be an arbitrary but fixed computation rule, and Q be an atomic query, and $\mathcal{S}$ be a final system for Q that is completed. Then for every subgoal A in $\mathcal{S}$ and every ground instance B of A,*

a. $B \in \mathcal{WF}(P)$ if and only if B is an instance of the head of an answer of A in $\mathcal{S}$ that has an empty body; and

b. $\sim B \in \mathcal{WF}(P)$ if and only if B is not an instance of the head of any answer of A in $\mathcal{S}$.

Proof: Let I be the interpretation of $P(\mathcal{S})$ such that $I = I(\mathcal{S})|_{P(\mathcal{S})}$. By Theorem 5.7, it suffices to prove that $I = \mathcal{WF}(P(\mathcal{S}))$. Clearly $I \subseteq \mathcal{WF}(P(\mathcal{S}))$ by the definition of $I(\mathcal{S})$ in Definition 5.2.

For the other direction, it suffices to show that $I \in \mathcal{ST}3(P(\mathcal{S}))$, i.e., $I = LPM(\frac{P(\mathcal{S})}{I})$. Since $\mathcal{S}$ is a final system, no transformation can be applied. For every negative literal $\sim B^B_B$ that occurs in $P(\mathcal{S})$, since $\sim B^B$ cannot be simplified using SIMPLIFICATION, $I(B^B_B) = \mathbf{u}$. Since ANSWER COMPLETION cannot be applied to $\mathcal{S}$, for every subgoal A in $\mathcal{S}$ and for every atom H that occurs in the head of some answer of A, H is supported by A. Let $J = LPM(\frac{P(\mathcal{S})}{I})$. By a structural induction over the definition of H being supported by A,

- $J(B^A_H) = \mathbf{t}$ for every ground instance B of H if and only if A has an answer that has H in the head and an empty body;
- $J(B^A_H) = \mathbf{u}$ for every ground instance B of H if and only if A has some answers that have H in the head and all answers of A that have H in the head have some delayed literals.

Therefore $I = J$ and $I \in \mathcal{ST}3(P(\mathcal{S}))$. By Theorem 2.3, $\mathcal{WF}(P(\mathcal{S})) \subseteq I$. □

SLG resolution produces answers of queries that may contain variables. The following theorem shows that if the universal closure, $\forall H$, of some instance H of a subgoal A is true in the well-founded partial model, then SLG resolution is able to derive an answer of A that is at least as general as H, provided that a completed final system can be constructed.

Theorem 5.9 *Let P be a finite program, R be an arbitrary but fixed computation rule, and Q be an atomic query, and $\mathcal{S}$ be a final system for Q that is completed. Then if for some instance H of a subgoal A in $\mathcal{S}$, $\forall H$ is true in $\mathcal{WF}(P)$, then there is an answer of A in $\mathcal{S}$ whose head has H as an instance and whose body is empty.*

Proof: Recall that the Herbrand universe $\mathcal{HU}$ is constructed from a language $\mathcal{LF}$ that contains all function symbols in P and Q. In addition, $\mathcal{LF}$ contains a unary function f' and a constant c' that do not occur in P or Q. Let $X_1, ..., X_n$ be all the distinct variables in H, and H^* be the ground atom obtained from H by replacing each variable X_i with the term $(f')^i(c')$. Then H^* is true in $\mathcal{WF}(P)$ since $\forall H$ is true in $\mathcal{WF}(P)$. By Theorem 5.8, there is an answer C of subgoal A in $\mathcal{S}$ whose head has H^* as an instance and whose body is empty. Since f' and c' never occur in any SLG derivation, the head of C must also have H as an instance. □

5.4 Termination and Data Complexity

SLG resolution terminates for all function-free programs, and more generally, all programs with the bounded-term-size property [45]. The following definition is adapted from [45], with the difference that every variable is treated as of size 1 due to variant checking of subgoals.

Definition 5.4 [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 $f(t_1, ..., t_n)$ is one plus the sum of the sizes of its arguments.

A finite program P has the *bounded-term-size property* if there is a function $f(n)$ and a (computable) computation rule R such that whenever an atomic query Q has no arguments whose sizes exceed n, no atom in any X-rule of any subgoal in $\mathcal{S}$ has an argument whose size exceeds $f(n)$, where $\mathcal{S}$ is any system in any SLG derivation for Q with respect to P. □

Definition 5.5 Let P be a finite program. Then $|P|$ denotes the number of rules in P, and Π_P denotes the maximum number of literals in the body of a rule in P. Let s be an arbitrary positive integer. Then $\mathcal{N}(s)$ denotes the number of atoms of predicates in P that are not variants of each other and whose arguments do not exceed s in size. □

Theorem 5.10 (Termination) *Let P be a finite program with the bounded-term-size property, R be an arbitrary but fixed computation rule, and Q be an atomic query. Then a final system for Q can be constructed in $O(\mathcal{N}(s) \times |P| \times \mathcal{N}(s)^{\Pi_P})$ transformation steps for some $s > 0$.*

Proof: Let n be the maximum size of arguments in Q. Let $\mathcal{S}$ be any system in an SLG derivation for Q. By definition, no atom in any X-rule of any subgoal in $\mathcal{S}$ has an argument whose size exceeds $f(n)$ for some function f. Let $s = f(n)$.

The number of distinct subgoals in $\mathcal{S}$ is bounded by $\mathcal{N}(s)$. For each subgoal A in $\mathcal{S}$, the length of the initial X-sequence for A introduced by NEW SUBGOAL is bounded by $|P|$. Let G be an X-rule G of a subgoal A in $\mathcal{S}$ that is not disposed. The number of literals in the body of G is bounded by Π_P. This is due to the fact that delayed literals of an answer are not propagated in POSITIVE RETURN. The number of X-rules that can be generated directly from G is

- at most $\mathcal{N}(s)$ if G has a selected atom, and
- at most 1 otherwise.

Each of the resulting X-rules that is generated directly from G either

- has the same number of delayed literals as G and has one literal less than G that is not delayed; or
- has the same number of literals that are not delayed as G and has one delayed literal less than G.

Finally each X-rule corresponding to an ordinal in the X-sequence of A in $\mathcal{S}$ can be disposed only once. Therefore the size of a system $\mathcal{S}$ is bounded by $O(\mathcal{N}(s) \times |P| \times \mathcal{N}(s)^{\Pi_P})$. As each transformation increases the size of a system, a final system can be constructed in $O(\mathcal{N}(s) \times |P| \times \mathcal{N}(s)^{\Pi_P})$ steps.

□

In the framework of deductive databases, a query can be represented by an intensional database (IDB), P_I, which can be any finite function-free program. The predicates that occur in the rule bodies in P_I, but not in the rule heads in P_I, are the extensional database (EDB) predicates. An EDB is represented as a finite set of ground atoms over the EDB predicates. Given an EDB P_E, we can form a program $P_I \cup P_E$. P_I can be viewed as a function that maps P_E to the well-founded partial model $\mathcal{WF}(P_I \cup P_E)$.

Van Gelder *et al.* [46] has shown that for function-free programs, computing the well-founded semantics has a polynomial time data complexity. The notion of *data complexity*, as defined by Vardi [47], is the complexity of evaluating a database query when the query is fixed and the database is regarded as input.

Definition 5.6 [46] The *data complexity* of an IDB is defined as the computational complexity of deciding the answer to a ground atomic query as a function of the size of the EDB; in the context of well-founded semantics, this means deciding whether the ground atom is positive in the well-founded partial model. □

Theorem 5.11 *Let P_I be an IDB that is an arbitrary finite function-free program, P_E be a finite EDB, and P be $P_I \cup P_E$. Let R be an arbitrary but fixed computation rule, and Q be a function-free ground atomic query. Then a final system $\mathcal{S}$ for Q can be constructed in polynomial time in the size of the EDB.*

Proof: For function-free programs and atomic queries, the size of each argument is 1. Then $\mathcal{N}(1)$ denotes the number of distinct function-free atoms that are not variants of each other. Since the number of predicates in P and their arities are fixed, $\mathcal{N}(1)$ is a polynomial in the size of P_E. By Theorem 5.10, a final system $\mathcal{S}$ for Q can be constructed within $O(\mathcal{N}(1) \times |P| \times \mathcal{N}(1)^{\Pi_P})$ steps. Let $k = \mathcal{N}(1) \times |P| \times \mathcal{N}(1)^{\Pi_P}$. We show that the total time for constructing $\mathcal{S}$ is polynomial in the size of P_E.

We assume that a global table of subgoals and their answers are maintained. Answers of the same subgoal that have the same atom in the head are grouped together since only one of them is used in POSITIVE RETURN. Thus the time for searching and inserting a subgoal is $O(log\mathcal{N}(1))$.

Similarly it takes time $O(log\mathcal{N}(1))$ to insert an answer and check whether the inserted answer has the same head atom as some previous answer already in the table.

Since the EDB P_E is a finite set of ground facts, all literals of EDB predicates can be solved directly and so subgoals of EDB predicates do not have to be maintained. In the following all subgoals refer to subgoals of IDB predicates.

Each application of NEW SUBGOAL takes time $O(log\mathcal{N}(1))$ for checking whether a subgoal is new and a constant amount of time to construct all initial X-rules of a subgoal (since P_I is fixed).

Each application of POSITIVE RETURN and NEGATIVE RETURN can be carried in a data-driven manner. When an X-rule G is generated that has a selected atom A, POSITIVE RETURN is performed on G with all existing answers of A. When an answer for A is derived whose head atom is distinct from that of all previous answers, POSITIVE RETURN can be performed on all active X-rules that have a selected atom A using the new answer. Thus the time for each application of POSITIVE RETURN or NEGATIVE RETURN is at most $O(log\mathcal{N}(1))$ if we include the time to check whether a newly generated answer of a subgoal has a head atom that is distinct from all previous answers. Similarly each application of DELAYING is at most $O(log\mathcal{N}(1))$.

By definition, the transformation SIMPLIFICATION is applied to only subgoals that are completed. It can be done in a data-driven manner based upon whether a delayed literal is successful or failed. The time for each application of SIMPLIFICATION is a constant.

We assume that COMPLETION is postponed until no other transformation can be applied to subgoals that are not completed. A linear traversal of all active X-rules of all subgoals that are not completed can determine subgoals that are not completely evaluated, i.e.,

- a subgoal that has an active X-rule with a selected negative literal;
- a subgoal that has an active X-rule whose selected atom A is a subgoal that is not completely evaluated.

The complement of the set of subgoals that are not completely evaluated, with respect to the set of all subgoals that are not completed, gives the largest possible set of subgoals that are completely evaluated. Thus the time of one application of COMPLETION is $O(k)$. The total number of applications of COMPLETION is $O(\mathcal{N}(1))$, and so the total time spent on COMPLETION is $O(\mathcal{N}(1) \times k)$.

We assume that ANSWER COMPLETION is postponed until no other transformation is applicable. A linear traversal of all answers of subgoals that are completed can determine all pairs (A, H), where H is the head of an answer of a subgoal A and H is supported by A. Then answers of a completed subgoal A whose heads are not supported by A are deleted by ANSWER COMPLETION. The time of one application of ANSWER COMPLETION is $O(k)$. The total number of applications of ANSWER COMPLETION is the total number $O(\mathcal{N}(1))$ of distinct subgoals times the total number $O(\mathcal{N}(1))$ of distinct head atoms in answers of a subgoal. Hence the total time spent on ANSWER COMPLETION is $O(\mathcal{N}(1)^2 \times k)$.

In summary, the time for constructing a final system $\mathcal{S}$ is $O(k \times log\mathcal{N}(1) + \mathcal{N}(1) \times k + \mathcal{N}(1)^2 \times k)$, where $k = \mathcal{N}(1) \times |P| \times \mathcal{N}(1)^{\Pi_P}$, and it is polynomial in the size of P_E. Notice that P_I in $P = P_I \cup P_E$ is fixed, and so $|P|$ is linear in the size of P_E and Π_P is a constant since every rule in P_E is a ground fact, with an empty body. In addition, $\mathcal{N}(1)$ is polynomial in the size of P_E. □

In practice, efficient incremental algorithms have been developed that detect subgoals that are completely evaluated or are possibly involved in loops through negation in a constant amount of time [11]. We believe that the freedom of choosing an arbitrary computation rule and choosing an arbitrary strategy of selecting transformations in SLG resolution offers the maximum flexibility for practical implementations.

6 Restricted SLG Resolution

SLG resolution provides a general framework for effective query evaluation of logic programs with respect to the well-founded partial model. In this section, we show that SLG resolution can be simplified for restricted classes of programs without compromising its soundness and search space completeness. In particular, we consider definite programs, locally stratified programs, and modularly stratified programs. For any program in these classes, the well-founded partial model is two-valued and is the only stable model of the program.

6.1 Definite Programs

Definite programs are programs without negation. The well-founded partial model [46] of a definite program coincides with the least Herbrand model [44]. Any transformation that deals with negative literals is no longer needed. Without negation, no delayed literals will be introduced. Thus all X-rules in a system are just rules.

The following transformations are needed and sufficient for definite programs:

- NEW SUBGOAL for introducing a new subgoal and resolution with rules in a program;
- POSITIVE RETURN for solving the selected atom of an X-rule using an answer (that has an empty body);
- COMPLETION for disposing active X-rules of subgoals that are completely evaluated.

The results in Section 5 can be specialized to definite programs and the corresponding least Herbrand model semantics.

Since COMPLETION does not affect answers of subgoals, all applications of COMPLETION for definite programs can be postponed until the last step. Even then, it is not necessary since all answers have already been generated. That is, only NEW SUBGOAL and POSITIVE RETURN are really necessary for definite programs. This restriction of SLG resolution to definite programs results in a query evaluation strategy that is equivalent to OLDT [43] and SLD-AL [49], modulo differences in variant checking or subsumption checking of subgoals.

In practice, if a subgoal is completed, there is no need for a choice point for potentially new answers of the subgoal. That is, active X-rules that have a selected atom A, where A is completed, can be disposed after all answers of A have been used in POSITIVE RETURN to solve the selected atom. The use of COMPLETION may allow early disposal of such active X-rules.

6.2 Locally Stratified Programs

Stratified programs are programs in which there is no negation through recursion [1]. Przymusinski [28] extended the class of stratified programs to a wider class, called locally stratified programs, and introduced the perfect model semantics. There is no infinite recursion through negation in locally stratified programs. The perfect Herbrand model of a locally stratified program coincides with the well-founded partial model.

Let P be a finite program. P is *locally stratified* [28, 37] if there is an assignment of ordinal levels to ground atoms such that whenever a ground atom appears negatively in the body of an instantiated rule, the head of the ground rule is of strictly higher level, and whenever a ground atom appears positively in the body of an instantiated rule, the atom in the head has at least its level.

Due to stratification, for every selected ground negative literal $\sim B$, subgoal B can be completed before $\sim B$ needs to be solved using NEGATIVE RETURN. Since the well-founded partial model of a stratified program is two-valued, B either succeeds or fails when it is completed. Thus DELAYING is not needed.

In summary, the following transformations are necessary and sufficient for locally stratified programs:

- NEW SUBGOAL for introducing a new subgoal and resolution with rules in a program;
- POSITIVE RETURN for solving the selected atom of an X-rule using an answer (that has an empty body);
- NEGATIVE RETURN for solving the selected ground negative literal of an X-rule when its positive counterpart either succeeds or fails;
- COMPLETION for disposing active X-rules of subgoals that are completely evaluated.

By an induction on the strata of ground subgoals, it can be shown that all ground subgoals can be completed and either succeed or fail before the corresponding negative literal is solved by NEGATIVE RETURN. Thus the results in Section 5 can be specialized to locally stratified programs and the corresponding perfect Herbrand model.

Extensions of OLDT [43] and SLD-AL [49] have been developed for stratified programs [20, 40]. Our restriction of SLG resolution to locally stratified programs differs in that a single system of subgoals is maintained, which guarantees that each subgoal be evaluated only once.

6.3 Modularly Stratified Programs

Ross [37] studied a more general class of programs that can be evaluated in a subgoal-at-a-time fashion, called *modularly stratified programs.* Consider the well known game-playing program [17]:

$$win(X) \leftarrow move(X,\ Y),\ \sim win(Y).$$

where X is a winning position if there is a move from X to Y that is not a winning position. The program is not locally stratified in general. However, if *move* is acyclic, the program is modularly stratified. For modularly stratified programs, the same transformations of locally stratified programs can be used, except that a certain computation rule must be assumed.

Let P be a program. We say that a predicate p *calls* a predicate q if there is a rule in P such that p occurs in the head and q occurs in the rule body. Let DG be the corresponding calling graph of P. The set of predicates in P can be partitioned into equivalence classes according to the strongly connected components of DG. A program P can be broken into complete components following the partition of predicates. There is a natural partial ordering $\prec$ over components, where $F_1 \prec F_2$ if some predicates in F_2 call directly or indirectly some predicates in F_1.

Definition 6.1 [Modular Stratification[37]] Let P be a finite program and $\prec$ be the partial ordering over complete components of P. P is *modularly stratified* if, for every component F of P,

- There is a total well-founded model M for the union of all components $F' \prec F$, and
- The quotient of F modulo M, $\frac{F}{M}$, is locally stratified.

□

For query evaluation of modularly stratified programs, we must ensure that literals whose predicates are defined in a lower component be solved first. Therefore, an arbitrary but fixed computation rule does not work. Consider a query p with respect to the following program:

$p \leftarrow \sim p, \sim q.$
$q.$

If the computation rule selects $\sim p$, there will be an infinite negative loop, in which case delaying transformation must be applied. To avoid delaying, we need to use a computation rule that selects literals of lower components first.

Ross [37] introduced the notion of *left-to-right* modularly stratified programs. Each modularly stratified program can be converted into a left-to-right modularly stratified program by putting in the body of each rule all literals of lower components before literals of the same component as the head of the rule.

Let P be a left-to-right modularly stratified program, and R be the left-to-right computation rule and Q be an atomic query. Then the same transformations for locally stratified programs are necessary and sufficient for constructing SLG derivations for Q with respect to P under R.

By an induction on the level of components and the levels of ground atoms in each component, it can be shown that all ground subgoals can be completed and either succeed or fail before the corresponding negative literal is solved by NEGATIVE RETURN. Thus the results in Section 5 can be specialized to left-to-right modularly stratified programs under a left-to-right computation rule.

All modularly stratified programs are also weakly stratified [27], but the converse is not true [37]. The major difference between modularly stratified and weakly stratified programs is in the notion of components. In modularly stratified programs, components are defined in terms of the dependency relationship among predicates, while in weakly stratified programs, components are defined in terms of the dependency relationship among ground atoms. As a result, when a query is evaluated with respect to a finite non-ground program, literals in rule bodies of a modularly stratified programs can be put in a sequential order so that literals of lower components are always solved first. Such a static ordering is not possible for weakly stratified programs because the level of a component to which a literal belongs depends upon the variable bindings at run time. Therefore the full SLG resolution is needed for query evaluation of (non-ground) weakly stratified programs.

7 Discussion

This section discusses two decisions that are made in the design of SLG resolution, namely variant checking of subgoals and answers and an arbitrary computation rule. We compare with related work and present experiences in three implementations of SLG resolution that have been developed.

7.1 Variant versus Subsumption Checking

To guarantee termination, SLG resolution checks for repeated subgoals and repeated answers. Repeated subgoals are solved using only answers from previous calls and repeated answers are not returned to solve the selected atom of an X-rule. In SLG resolution, variant checking is used to detect both repeated subgoals and repeated answers. Two subgoals are identical if they are variants of each other. If two answers of the same subgoal have head atoms that are variants of each other, only one of them is used in POSITIVE RETURN to solve the selected atom of an X-rule.

Another approach is to use subsumption checking for subgoals and answers. If a subgoal B is an instance of a previous subgoal A, then B is solved using answers of A. Similarly if an answer is subsumed by a previous one, only the more general answer is kept.

The choice of variant checking in SLG resolution is motivated by two advantages. One is that variant checking allows easier and efficient implementation of indexing of tables of subgoals and answers. By using a ground representation of variables, variant checking of atoms can be reduced to equality of ground atoms. An efficient table lookup operation is crucial to the efficiency of an implementation of SLG resolution. The other advantage of variant checking of subgoals is that Prolog-style meta programming using builtin predicates such as var/1 can be supported.

The main disadvantage of variant checking is repeated computation among subgoals that subsume each other.

Example 7.1 Consider the following simple program:

$$\begin{array}{l} edge(a,b). \quad edge(b,c). \quad edge(c,d). \quad edge(d,a). \\ path(X,Y) \leftarrow edge(X,Y). \\ path(X,Y) \leftarrow edge(X,Z), path(Z,Y). \end{array}$$

Suppose that a query $path(X,Y)$ is evaluated and a left-to-right computation rule is used. Then the following set of subgoals of $path/2$ will be evaluated:

$$\{path(X,Y), path(b,Y), path(c,Y), path(d,Y), path(a,Y)\}$$

Clearly answers of $path(X,Y)$ include those of the other subgoals of $path/2$. □

Even with subsumption checking of subgoals, repeated computation cannot be fully avoided in general when a more specific subgoal, say $p(X, X)$, is encountered before a more general one such as $p(X, Y)$.

7.2 Computation Rule and Search Strategies

SLG resolution allows an arbitrary computation rule for selecting a literal from a rule body. Given a finite program P, an arbitrary but fixed computation rule R, and an atomic query Q, it is possible that a non-ground negative literal $\sim B$ may be selected by R from a rule body during the evaluation of Q with respect to P. Such a situation may lead to *floundering*. SLG resolution is able to solve $\sim B$ only if either B has an answer that has B in the head and has an empty body, or subgoal B is completed without any answers. Without mechanisms such as constructive negation [9, 10, 30, 42], non-ground negative literals cannot be solved in general. In SLG resolution, a final system for Q with respect to P may not be completed and may contain active X-rules that have a selected non-ground negative literal.

One may reduce floundering by using computation rules that select positive literals before negative ones. One may also avoid floundering by imposing conditions on programs. A finite program is *range restricted* if for every rule in the program, all variables in the rule head must occur in the rule body, and every variable that occurs in a negative literal in the rule body must also occur in a positive literal in the rule body. By using a computation rule that always selects positive literals before negative ones, floundering can be avoided for range restricted programs.

SLG resolution avoids imposing any restrictions on the computation rule an implementation may use or on programs with respect to which queries may be evaluated. The main reason is that an implementation of SLG resolution is free to choose any computation rule, such as the left-to-right computation rule in Prolog systems. In left-to-right modularly stratified programs [37], for example, the left-to-right computation rule must be used to guarantee that literals of predicates from lower components be selected first. This may or may not be consistent with requirements that positive literals be selected before negative ones. Also programmers can use their knowledge of the computation rule in an implementation to control floundering and to write more efficient programs. Negative literals may be used as guard conditions to determine if some expensive computation in the rest of a rule body should be evaluated.

SLG resolution also allows an arbitrary strategy for selecting which transformation to apply when multiple transformations are applicable to a system. In other words, the definition of SLG resolution does not dictate any particular strategy that should be used. In several implementations [13, 35, 38], a greedy strategy is used in which

- whenever a new answer C of a subgoal A is created, POSITIVE RETURN is applied using C to every X-rule in the system that has a selected atom A; and
- whenever an active X-rule G of a subgoal is created that has a selected atom A, POSITIVE RETURN is applied to G using every existing answer of A; and
- whenever a new subgoal A is encountered, its initial X-rules are generated by NEW SUBGOAL and are transformed.

This greedy strategy is close to the top-down tuple-at-a-time computation.

A different strategy has been implemented in a version of XSB [38], which computes as many answers as possible for a subgoal before any of the answers is returned through POSITIVE RETURN. This strategy is close to the bottom-up set-at-a-time computation in *ordered_search* [34].

Different implementations may choose different search strategies, according to specific applications, and SLG resolution offers the flexibility of such choices.

7.3 Related Work

SLS resolution is the early work on an operational procedure for the well-founded semantics [29, 36]. It does not incorporate any tabling mechanism. Every selected atom is solved by resolution with program rules, and every selected ground negative literal is solved by computing the corresponding positive literal up to a fixpoint. Without tabling, it serves only as an ideal top-down procedural semantics since it may go into infinite loops even for function-free programs. Without tabling, it requires a *positive and negatively parallel* computation rule in order to guarantee search space completeness (for non-floundering queries). That is, positive literals are selected before negative ones, and when only negative literals remain in a rule body, all the negative literals are selected and evaluated. The latter is required because the evaluation of one ground negative literal may go into an infinite loop while the evaluation of another may fail.

WELL! [6] and XOLDTNF resolution [14] represent a simple modification of SLS resolution with tabling to handle loops through negation. By maintaining a negative context with each subgoal, both can detect loops through negation, treat the ground negative literal involved in such a loop as *undefined*, and avoid non-termination. An answer consists of both an atom and a truth value that can be either **t** or **u**. The use of negative contexts, however, prevents the full sharing of answers across different negative contexts. In the worst case, a subgoal may be evaluated in an exponential number of distinct negative contexts.

Two methods of query evaluation have been developed for left-to-right modularly stratified programs. One is an extension of SLS resolution, called *QSQR/SLS* by Ross [37], and the other is an extension of supplementary magic templates, called *ordered_search* [34]. Both maintain subgoal dependency information to check whether subgoals are completely evaluated. Ross showed [37] that QSQR/SLS procedure has the same complexity as supplementary magic rewriting.

An interesting aspect of bottom-up computation such as magic templates [33] is that its checking of repeated subgoals is neither variant nor subsumption checking. Subgoals that have the same binding patterns are treated the same. For the program and query in Example 7.1, subgoals $path(b, Y)$, $path(c, Y)$, $path(d, Y)$ and $path(a, Y)$ all have the same binding pattern, while $path(X, Y)$ has a different binding pattern. Subgoals of the same binding pattern will share the same table of answers. For programs with function symbols, an argument is considered *bounded* if one of the variables in the argument is bounded. Thus subgoals $q(f(g(Y)))$ and $q(f(g(a)))$ may be treated as calls of the same binding pattern when they are obtained from $q(f(X))$ by binding X to $g(Y)$ and $g(a)$ respectively, even though one subsumes the other.

For general programs, the magic-sets transformation does not always preserve the well-founded semantics [18]. The proposed solutions in [18, 19] use a doubled program, one for computing definitely true facts and the other for computing not definitely false facts. The separate computation of these two classes of facts may cause redundant inferences since the sets of not definitely false facts are decreasing, but are computed in an increasing manner. The doubled program method also tends to make too many magic facts true, which means that more subgoals are evaluated than necessary. An alternating fixpoint tailored to magic-sets in [26] alleviates this problem, but still generates many irrelevant magic facts in the initial stages of the fixpoint computation [11].

An extension of *ordered_search* to compute the well-founded semantics, called *well-founded ordered search*, was developed by Stuckey and Sudarshan [41]. Both SLG resolution and well-founded ordered search support goal-oriented query evaluation and allow an arbitrary computation rule. The main difference is in the treatment of negative literals possibly involved in loops through negation. Our implementation of SLG resolution checks only potential loops through negation and may delay ground negative literals more than necessary. But it avoids repeated computation by keeping delayed literals explicitly and simplifying them later. In contrast, well-founded ordered search maintains precise dependency information among subgoals, by essentially run time re-organization of a stack of subgoals, and is able to detect genuine loops through negation. For portions of programs that involve loops through negation, it uses alternating fixpoint computation such as [26], which may repeat certain steps of computation.

In [7], a method of top-down tabulated resolution for well-founded semantics was presented. Like SLG resolution, it uses several transformations to construct a search forest, which corresponds to our notion of a system of subgoals. There are some majors differences, though, between SLG resolution and tabulated resolution in [7].

In SLG resolution, let G be an X-rule of a subgoal and $\sim B$ be the selected negative literal of G. When $\sim B$ is solved in NEGATIVE RETURN or DELAYING, G is disposed and possibly replaced by another X-rule. In other words, G has at most one child due to the selection of $\sim B$. Even when $\sim B$ is delayed, it is never selected again by the computation rule. A delayed negative literal is only simplified later when its truth value becomes known.

In the tabulated resolution in [7], let G be a node in a search forest and $\sim B$ be the selected ground negative literal of G. Then G may have two child nodes. One is derived through *extension by u-assumption*, in which $\sim B$ is essentially replaced with the undefined truth value $\mathbf{u}$. The other is derived through *extension by negation as failure*, in which B is known to be successful or failed and $\sim B$ is solved by negation as failure. As a result, the conjunction of the remaining literals in the body of G may have to be evaluated twice.

Another difference between SLG resolution and the tabulated resolution in [7] is in the treatment of non-ground negative literals. SLG resolution selects a non-ground negative literal only once and tries to solve it when it can be solved by negation as failure. In contrast, the tabulated resolution in [7] may select a non-ground negative literal twice, once when it is initially selected and the other when the non-ground negative literal becomes ground later. In addition, an answer may contain non-ground negative literals in its body. These non-ground negative literals are propagated when the answer is used to solve the selected atom of a node. A small variation of Example 3.1 can be used to show that propagation of non-ground negative literals may cause an exponential number of distinct conditional answers for a subgoal. Thus the polynomial data complexity of the well-founded semantics for function-free programs is not preserved by the tabulated resolution in [7].

A unique feature of SLG resolution is the handling of ground negative literals that may be involved in loops by using DELAYING, SIMPLIFICATION and ANSWER COMPLETION. There are several advantages. First, DELAYING provides SLG resolution the freedom of an arbitrary computation rule without compromising the soundness and search space completeness of SLG resolution. Even when the truth value of a ground negative literal cannot be decided, e.g., in the case of loops through negation, DELAYING allows SLG resolution to proceed and solve remaining literals in a rule body. In a certain sense, it achieves the same effect of the positivistic and negatively parallel computation rule in (global) SLS resolution [29, 36], yet without imposing any condition on the computation rule. Second, by maintaining delayed literals explicitly and simplifying them later, no derivation steps in SLG resolution are repeated. Finally, SLG keeps delayed literals in answers of subgoals that are undefined in the well-founded semantics. This allows query evaluation with respect to other three-valued stable models by further processing of the answers under the well-founded semantics.

The decision in SLG resolution not to propagate negative delayed literals in the bodies of answers is necessary in order to guarantee the polynomial data complexity of SLG resolution for function-free programs. On the other hand, this leads to the creation of positive delayed literals in POSITIVE RETURN, and the need for ANSWER COMPLETION to delete, under certain conditions, answers that have positive delayed literals in their bodies. The transformation ANSWER COMPLETION is an expensive operation since it may require traversing the answers of subgoals. An interesting topic for future work is to investigate conditions under which the use of ANSWER COMPLETION can be avoided or minimized.

7.4 Implementations of SLG Resolution

Perhaps the most important aspect of SLG resolution is the availability of its implementations and their performance. All of them use the left-to-right computation rule.

The first implementation is a Prolog meta interpreter [13]. A major implementation issue is to detect subgoals that are completely evaluated for COMPLETION and potential loops through negation for DELAYING. The latter is needed so that DELAYING can be avoided when ground negative literals can be solved using NEGATIVE RETURN. Both require the dependency information about subgoals in a system. It turns out that with DELAYING and SIMPLIFICATION it is not necessary to compute precisely loops through negation in a system, which is likely to be very expensive at run time. Instead we have developed an efficient approximate algorithm for incremental maintenance of the dependency information of subgoals [11]. The top-down framework of SLG resolution leads to a stack of subgoals based upon the sequence in which they are encountered and lends it naturally to incremental maintenance of dependencies among subgoals. By inspecting the dependency information of a *single* subgoal A, it is possible to determine whether all subgoals from the top of the stack to A are completely evaluated or are *possibly* involved in loops through negation. The performance of the meta interpreter implementation is competitive with others that handle arbitrary negation.

The second implementation is a Prolog compiler, called XSB [38]. XSB modifies the Warren Abstract Machine (WAM) of Prolog to implement SLG resolution restricted to modularly stratified programs. By taking advantage of WAM technology and efficient indexing of tables of subgoals and answers at WAM level, XSB has demonstrated impressive performance for query evaluation of deductive databases [38].

The most recent implementation uses source program transformation and tabling primitives external to Prolog WAM [35]. A program P is transformed into another program P' by inserting tabling primitives, and Prolog execution of P' yields SLG resolution. The tabling primitives are independent of the underlying Prolog WAM. They maintain tables of subgoals and answers and implement the control strategy that is needed for SLG resolution. It is much more efficient than the meta interpreter and portable across different Prolog systems.

All three implementations support an integration of SLG resolution and Prolog computation. A distinction can be made between predicates that are solved using SLG resolution and those that are solved as in Prolog. Ordinary Prolog computation can be incorporated into SLG resolution in a simple manner, without any overhead. In the other direction, predicates solved by SLG resolution can also be called by Prolog predicates.

8 Conclusion

SLG resolution serves as both a foundation and a practical framework for computing the well-founded semantics of logic programs. Theoretically, a number of fundamental transformations are identified, cleanly separating logical issues from procedural information. Restricted versions of SLG resolution have been developed for programs with limited uses of negation, including definite, locally stratified, and modularly stratified programs. These programs do not have to pay for the overhead of transformations that are not needed. This sheds light on the role that each transformation plays. SLG resolution preserves all three-valued stable models, including the well-founded partial model as a special case. It terminates for all programs with the bounded-term-size property.

SLG resolution guarantees the polynomial time data complexity for well-founded negation of function-free programs. It can be enhanced by further processing of the answers of subgoals relevant to a query under the well-founded semantics to deliver answers that are specific to other three-valued stable models [12].

Practically, SLG resolution is upward compatible with existing Prolog systems. This facilitates the integration of SLG resolution with Prolog applications. More importantly, Prolog compiler technology can be adapted for an efficient implementation of SLG resolution.

We firmly believe that SLG resolution will have an important impact on the theory and practice of logic-based computational systems. Its termination properties on stratified function-free programs make it a good strategy for deductive database query processing; its ability to be integrated seamlessly with Prolog evaluation makes it a good logic programming strategy, and its polynomial data complexity for handling nonstratified programs makes it a good strategy for nonmonotonic reasoning in knowledge based systems.

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. Morgan Kaufmann Publishers, Los Altos, CA, 1988.

[2] K.R. Apt and M.H. Van Emden. Contributions to the theory of logic programming. *Journal of ACM*, 29(3):841–862, July 1982.

[3] I. Balbin, G.S. Port, K. Ramamohanarao, and K. Meenakshi. Efficient bottom-up computation of queries on stratified databases. *Journal of Logic Programming*, 11:295–344, 1991.

[4] F. Bancilhon, D. Maier, Y. Sagiv, and J. Ullman. Magic sets and other strange ways to implement logic programs. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 1–15, March 1986.

[5] C. Beeri and R. Ramakrishnan. On the power of magic. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 269–283, San Diego, CA, March 1987.

[6] N. Bidoit and P. Legay. WELL!: An evaluation procedure for all logic programs. In *Intl. Conference on Database Theory*, pages 335–348, 1990.

[7] R. Bol and L. Degerstedt. Tabulated resolution for well founded semantics. In *Intl. Logic Programming Symposium*, October 1993.

[8] F. Bry. Query evaluation in recursive databases: Bottom-up and top-down reconciled. In *Intl. Conference on Deductive and Object-Oriented Databases*, December 1989.

[9] D. Chan. Constructive negation based on the completed database. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Proc. 5th Int. Conf. and Symp. on Logic Programming*, pages 111–125, 1988.

[10] W. Chen and L. Adams. Constructive negation of general logic programs. Technical Report 94-CSE-16, Department of Computer Science and Engineering, Southern Methodist University, April 1994.

[11] W. Chen, T. Swift, and D.S. Warren. Efficient top-down computation of queries under the well-founded semantics. *Journal of Logic Programming*, 1994. to appear.

[12] W. Chen and D.S. Warren. Computation of stable models and its integration with logical query processing. *IEEE Transactions on Knowledge and Data Engineering*, 1994. to appear.

[13] W. Chen and D.S. Warren. *The SLG System*, August, 1993. available by anonymous FTP from seas.smu.edu or cs.sunysb.edu.

[14] W. Chen and D.S. Warren. A goal-oriented approach to computing well founded semantics. In *Joint Intl. Conference and Symposium on Logic Programming*, November 1992. also available as SMU Technical Report 92-CSE-9.

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

[16] S.W. Dietrich and D.S. Warren. Extension tables: Memo relations in logic programming. Technical Report 86/18, Department of Computer Science, SUNY at Stony Brook, 1986.

[17] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R.A. Kowalski and K.A. Bowen, editors, *Joint Intl. Conference and Symposium on Logic Programming*, pages 1070–1080, 1988.

[18] David B. Kemp, Peter J. Stuckey, and Divesh Srivastava. Magic sets and bottom-up evaluation of well-founded models. In *Intl. Logic Programming Symposium*, pages 337–351, 1991.

[19] David B. Kemp, Peter J. Stuckey, and Divesh Srivastava. Query restricted bottom-up evaluation of normal logic programs. In *Joint Intl. Conference and Symposium on Logic Programming*, pages 288–302, 1992.

[20] D.B. Kemp and R.W. Topor. Completeness of a top-down query evaluation procedure for stratified databases. In *Joint Intl. Conference and Symposium on Logic Programming*, pages 178–194, 1988.

[21] J. Komorowski. Towards a programming methodology founded on partial deduction. In *Proceedings of the European Conference on Artificial Intelligence*, 1990.

[22] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, New York, second edition, 1987.

[23] J.W. Lloyd and J.C. Shepherdson. Partial evaluation in logic programming. *Journal of Logic Programming*, 11:217–242, 1991.

[24] W. Marek and M. Truszczynski. Autoepistemic logic. *Journal of ACM*, 38(3):588–619, 1991.

[25] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, 1982.

[26] S. Morishita. An alternating fixpoint tailored to magic programs. In *Proceedings of the Deductive Database Workshop at Joint International Conference and Symposium on Logic Programming*, 1992.

[27] H. Przymusinska and T.C. Przymusinski. Weakly perfect model semantics for logic programs. In R.A. Kowalski and K.A. Bowen, editors, *Joint Intl. Conference and Symposium on Logic Programming*, pages 1106–1120, 1988.

[28] 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. Morgan Kaufmann Publishers, Los Altos, CA, 1988.

[29] T.C. Przymusinski. Every logic program has a natural stratification and an iterated least fixed point model. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 11–21, 1989.

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

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

[32] T.C. Przymusinski. The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae*, 13:445–463, 1990.

[33] R. Ramakrishnan. Magic templates: A spellbinding approach to logic programs. *Journal of Logic Programming*, 11:189–216, 1991.

[34] R. Ramakrishnan, D. Srivastava, and S. Sudarshan. Controlling the search in bottom-up evaluation. In *Joint Intl. Conference and Symposium on Logic Programming*, pages 273–287, 1992.

[35] R. Ramesh and W. Chen. A portable method of integrating SLG resolution into prolog systems. In *Intl. Logic Programming Symposium*, November 1994.

[36] K.A. Ross. A procedural semantics for well founded negation in logic programs. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 22–33, 1989.

[37] K.A. Ross. *The Semantics of Deductive Databases.* PhD thesis, Department of Computer Science, Stanford University, August 1991.

[38] K. Sagonas, T. Swift, and D.S. Warren. XSB as an efficient deductive database engine. In *ACM SIGMOD Conference on Management of Data*, pages 442–453, 1994.

[39] H. Seki. On the power of Alexander templates. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 150–159, March 1989.

[40] H. Seki and H. Itoh. A query evaluation method for stratified programs under the extended CWA. In *Joint Intl. Conference and Symposium on Logic Programming*, pages 195–211, 1988.

[41] P. Stuckey and S. Sudarshan. Well-founded ordered search. In *Proceedings of the 13th Conference on Foundations of Software Technology and Theoretical Computer Science*, 1993. LNCS 761.

[42] P.J. Stuckey. Constructive negation in constraint logic programming. In *Proceedings of the 6th IEEE Annual Symposium on Logic in Computer Science*, pages 328–339, 1991.

[43] H. Tamaki and T. Sato. OLD resolution with tabulation. In *Intl. Conference on Logic Programming*, pages 84–98, 1986.

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

[45] A. Van Gelder. Negations as failure using tight derivations for general logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 149–176. Morgan Kaufmann Publishers, Los Altos, CA, 1988.

[46] A. Van Gelder, K.A. Ross, and J.S. Schlipf. The well-founded semantics for general logic programs. *Journal of ACM*, 38(3), July 1991.

[47] M. Vardi. The complexity of relational query languages. In *ACM Symposium on Theory of Computing*, pages 137–146, May 1982.

[48] L. Vieille. A database-complete proof procedure based upon SLD-resolution. In *Intl. Conference on Logic Programming*, 1987.

[49] L. Vieille. Recursive query processing: The power of logic. *Theoretical Computer Science*, 69:1–53, 1989.