A New Formulation of Tabled Resolution with Delay

Terrance Swift

Computer Science Department, SUNY Stony Brook, Stony Brook NY, USA 11794
Email: tswift@cs.sunysb.edu

Abstract. Tabling has become important to logic programming in part because it opens new application areas, such as model checking, to logic programming techniques. However, the development of new extensions of tabling logic programming is becoming restricted by the formalisms that underly it. Formalisms for tabled evaluations, such as SLG [3], are generally developed with a view to a specific set of allowable operations that can be performed in an evaluation. In the case of SLG, tabling operations are based on a variance relation between atoms. While the set of SLG tabling operations has proven useful for a number of applications, other types of operations, such as those based on a subsumption relation between atoms, can have practical uses. In this paper, SLG is reformulated in two ways: so that it can be parameterized using different sets of operations; and so that a forest of trees paradigm is used. Equivalence to SLG of the new formulation, Extended SLG or SLG_X, is shown when the new formalism is parameterized by variant-based operations. In addition, SLG_X is also parameterized by subsumption-based operations and shown correct for queries to the well-founded model. Finally, the usefulness of the forest of trees paradigm for motivating tabling optimizations is shown by formalizing the concept of relevance within a tabled evaluation.

Keywords: Logic Programming, Non-Monotonic Reasoning

1 Introduction

The ability to compute queries according to the Well-Founded Semantics [19] (WFS) and its extensions has proven useful for a number of applications, including model checking [13] and diagnosis [5]. Within the logic programming community, evaluation of the WFS is commonly done using tabled evaluation methods [3,2,4]. Of these, SLG resolution [3] (Linear resolution with Selection function for General logic programs) is of interest because it can be used with non-ground programs and because it has formed the foundation of extended implementation efforts (such as the XSB system [7]). Extensions to SLG address constructive negation [11], and a property called delay minimality [17].

Understood in their proper context, the ideas that differentiate SLG from other tabling methods are relatively simple. SLG breaks down tabled evaluation into a set of primitive operations (or, equivalently transformations). To handle
possible loops through negation, SLG provides a delaying operation to dynamically adjust a computation rule, along with a simplification operation to resolve away delayed literals when their truth value becomes known. SLG also ensures polynomial data complexity through controlling the manner in which delayed literals are propagated and by marking as completed subgoals that will not benefit from further resolution.

SLG was originally presented [3] using a notation in which sets of objects, called X-elements, are associated with ordinals, upon which transfinite induction is performed. Despite its power, the formulation can be somewhat difficult to learn and use, particularly when exploring operational aspects of SLG. Indeed, a stated goal of [20] is to derive a calculus for computing the well-founded semantics that is simpler to understand than SLG. In addition many of the definitions in the original formulation of SLG assume a variance relation between terms to determine when a new subcomputation should arise, when an answer should be used for resolution against a selected literal or when a delayed literal should be simplified. However, to derive new computation strategies for the well-founded semantics, or for extensions of the well-founded semantics, new operations are needed beyond these variant operations.

This paper defines a reformulation of SLG called Extended SLG ($SLG_X$) that reformulates SLG in two ways. First, because trees may naturally be formulated as sets (see e.g., [6]) SLG may be modeled using forests of trees rather than explicit sets of X-elements. We believe the resulting framework is clearer than the original formulation of SLG; however it does not lose any of the power of SLG to formalize transfinite computations. Indeed, an informal forest of trees model of SLG has been used to derive scheduling properties of tabled evaluations [8] and to motivate the design of an abstract machine for SLG [15,16].

Second, definitions in $SLG_X$ are geared so that underlying tabling operations are parameterizable. The first result of this paper is to prove full equivalence to SLG of $SLG_X$ parameterized with variant-style operations (called in this paper $SLG_{\text{variance}}$). The second main result is to parameterize $SLG_X$ with subsumption-based tabling operations to form a method called $SLG_{\text{subsumption}}$, and to prove the soundness and completeness of $SLG_{\text{subsumption}}$ for the well-founded semantics. While the use of subsumption is new to tabled evaluations of definite programs (cf. [18]), the formulation of a fully subsumption-based tabling method for the well-founded semantics is novel, to our knowledge.

The practical usefulness of $SLG_X$ is also demonstrated through an optimization called relevance. SLG was defined so that a computation terminates only when all answers have been derived for an initial query $Q$ along with all subgoals that arise in solving $Q$. However, it can sometimes be useful to ignore a subcomputation when it can be determined that the subcomputation will produce no further answers for the initial query. Section 4.1, formalizes these notions.

Before proceeding further, we informally introduce concepts of $SLG_X$, as parameterized by variant-style operations, through the following example.

Example 1. Consider a possible $SLG_{\text{variance}}$ evaluation $E_1$ of the query $?- p(X)$ against the program $P_1$
p(b).
p(c) :- not p(a).
p(X) :- t(X,Y,Z), not p(Y), not p(Z).

t(a,b,a).
t(a,a,b).

An SLG\textsubscript{variance} evaluation, like any SLG\textsubscript{X} evaluation, consists of a sequence of forests of SLG trees, each of which either have the form

\textit{Answer-template} :: \text{Delay-set}\{\text{Goal-List}\}

or \textit{fail}. In the first form, the \textit{Answer-template} is used to represent bindings made to a tabled subgoal in the course of resolution along a computation path; the \textit{Delay-set} contains a set of literals that have been selected by a fixed-order literal selection strategy but whose evaluation has been delayed; and the \textit{Goal-List} is a sequence of unresolved literals, SLG\textsubscript{X} requires a fixed literal selection strategy. In this paper we assume, without loss of generality, that the literal selection strategy is left-to-right within the \textit{Goal-List}. The evaluation of the query `\text{?- p(X) against P}` begins with a tree for \textit{p(X)} with root node \textit{p(X)} :: \text{p(X)}. In a root node, the \textit{Answer-template} reflects no bindings to the subgoal, the \textit{Delay-set} is empty, and the \textit{Goal-List} is identical to the original subgoal. Children of root nodes are produced through the \textit{Program Clause Resolution} operation, and the children of \textit{p(X)} :: \text{p(X)} are depicted in Figure 1. In this figure, numbers associated with nodes depict their order of creation in \(E_1\). Of these children, node \(p(b) \ :- \) \textit{is an answer node, defined as a leaf node whose \textit{Goal-List} is empty}. The selected literals of the other two children of \textit{p(X)} :: \text{p(X)} produce new tabled subgoals: \(t(X,Y,Z)\), and \textit{p(a)} through the \textit{SLG\textsubscript{variance}} operation \textit{New Subgoal} (note that in \textit{SLG\textsubscript{variance}} a new tree is created for \textit{p(a)} even though \textit{p(a)} is subsumed by \textit{p(X)}). The children of non-root nodes whose selected literals are positive (such as node 4) are produced through the \textit{Positive Return} operation, while the children of nodes with negative selected literals (such as node 2) may be produced through the \textit{Negative Return} operation. Because an answer for \textit{p(b)} is contained in node 9 in the forest of Figure 1, a failure node (node 10) is produced as a child for \textit{p(a)} :: \text{not p(b), not p(a)}. A failure node is subject to no further \textit{SLG\textsubscript{variance}} operations, and indicates that the computation path leading up to it has failed.

In Figure 1, all possible \textit{New Subgoal}, \textit{Program Clause Resolution}, \textit{Positive Return} and \textit{Negative Return} operations have been performed. Despite this, the truth value of \textit{p(a)} cannot be determined since the tree for \textit{p(a)} depends on node 19 (\textit{p(a)} :: \text{not p(a), not p(b)}), in which the literal \textit{not p(a)} is selected. SLG (and SLG\textsubscript{X}) overcomes this problem of self-dependency through negation by using a \textit{Delaying} operation which in this case, creates a new node \textit{p(a) :: not p(a), not p(b) (node 20)} in which \textit{not p(a)} has been moved from the \textit{Goal-List} into the \textit{Delay-set}. In Figure 2, \textit{Delaying} operations produce nodes 21 and 22 in addition to node 20. Delaying a negative literal allows other literals to be selected whose failure may break the self-dependency through negation.
(e.g. not p(b) is selected in node 22). Figure 2 includes relevant portions of a final forest reached after restarting all subcomputations enabled after delaying negative literals. In node 20, for example, the literal not p(b) is selected and a negative RETURN causes the computation path leading up to that node to fail. Delaying not p(a) also creates node 22, p(c) :- not p(a), which is termed a conditional answer, because it has a non-empty Delay-Set. After node 23 is produced, all computation paths stemming from p(a) are failed. The result is that the literal not p(a) should in fact succeed, and the conditional answer p(c) :- not p(a) should become unconditional — not p(a) should be removed from its Delay-set. SLGvariance uses a SIMPLIFICATION operation to produce the unconditional answer in node 25, p(a) :- l. Figure 2 shows the state of the SLGvariance evaluation after all possible SLGvariance operations have been applied. Note that each tree has been marked with the token complete denoting that it has been completely evaluated and can produce no more answers.

We now turn to formalizing the concepts presented in Example 1.

2 A Framework for Parameterizing Tabling Operations

Terminology and assumptions: We assume the standard terminology of logic programming (see, e.g., [12]). We assume that any program is defined over a countable language $\mathcal{L}$ of predicates and function symbols. If $L$ is a literal, then
vars(\(L\)) denotes the set of variables in \(L\). The Herbrand base \(H_P\) of a program \(P\) is the set of all ground atoms formed from \(L\). By a 3-valued interpretation \(I\) of a program \(P\) we mean a set of literals defined over \(H_P\). For \(A \in H_P\), if \(A \in I\), \(A\) is true in \(I\), and if \(notA \in I\), \(A\) is false in \(I\). When \(I\) is an interpretation and \(A\) is an atom, \(I|_A\) refers to

\[
\{ L | L \in I \text{ and } (L = G \text{ or } L = not G) \text{ and } G \text{ is in the ground instantiation of } A \}
\]

In the following sections, we use the terms goal, subgoal, and atom interchangeably. Variant terms are considered to be identical. \(SLG_X\) evaluations allow arbitrary, but fixed literal selection strategies. For simplicity, throughout this paper we assume that literals are selected in a left-to-right order.

We now provide definitions for concepts introduced in Example 1.

**Definition 1 (\(SLG_X\) Trees and Forest).** An SLG forest consists of a forest of SLG trees. Nodes of SLG trees have the form:

\[
\text{Answer Template} :- Delayset|GoalList
\]
or

\[ \text{fail} \]

In the first form, the Answer_Template is an atom, the Delay_Set is a set of delayed literals (see Definition 2) and Goal_List is a sequence of literals. The second form is called a failure node. The root node of an SLG tree may be marked with the term complete.

We call a node \( N \) an answer when it is a leaf node for which Goal_List is empty. If the Delay_Set of an answer is empty it is termed an unconditional answer, otherwise, it is a conditional answer.

Definition 2 specifies the exact formulation of delay literals. Definitions 8 and 9 will ensure that the root node of a given SLG tree, \( T \), has the form \( S \vdash \top \), where \( S \) is a subgoal. If \( T \) is an SLG tree in a forest \( F \) whose root node is \( S \vdash \top \) (possibly marked as complete), then we use the following terminology. \( S \) is the root node for \( T \) or that \( T \) is the tree for \( S \), and \( S \) is in \( F \).

**Definition 2 (Delay Literals).** A negative delay literal in the Delay_Set of a node \( N \) has the form \( \neg A \), where \( A \) is a ground atom. Positive delay literals have the form \( D_{\text{Answer}}^\text{Call} \), where \( A \) is an atom whose truth value depends on the truth value of some answer Answer for the subgoal Call. If \( \theta \) is a substitution, then \( (D_{\text{Answer}}^\text{Call})^\theta = (D)^\theta \).

Positive delay literals contain information so that they may be simplified when a particular answer to a particular call becomes unconditionally true or false. It is useful to define answer resolution so that it takes into account the form of delay literals.

**Definition 3 (Answer Resolution).** Let \( N \) be a node \( A \vdash D[L_1, \ldots, L_n] \), where \( n > 0 \). Let \( \text{Ans} = A' \vdash D' \) be an answer whose variables have been standardize apart from \( N \). \( N \) is SLG resolvable with \( \text{Ans} \) if \( \exists i \), \( 1 \leq i \leq n \), such that \( L_i \) and \( A' \) are unifiable with an mgu \( \theta \). The SLG resolvent of \( N \) and \( \text{Ans} \) on \( L_i \) has the form

\[ (A \vdash D[L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n])^\theta \]

if \( D' \) is empty, and

\[ (A \vdash D[L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n])^\theta \]

otherwise, where \( D = L_i \) if \( L_i \) is negative, and \( D = L_i^{L_i} \) otherwise.

A set of subgoals is completely evaluated when it can produce no more answers. Formally,

**Definition 4 (Completely Evaluated).** A set \( S \) of subgoals in a forest \( F \) is completely evaluated if at least one of the conditions holds for each \( S \in S \)

1. The tree for \( S \) contains an answer \( S \vdash \top \); or
2. For each node \( N \) in the tree for \( S \):
   (a) The selected literal \( L_S \) of \( N \) is completed or in \( S \); or
(b) There are no applicable New Subgoal, Program Clause Resolution, Positive Return, Delaying, or Negative Return operations (Definition 9) for \( N \).

Once a set of subgoals is determined to be completely evaluated, the completion operation marks the root node of the trees for each subgoal (Definition 1).

According to Definition 3, if a conditional answer is resolved against the selected literal in the Goal List of a node, the information about the delayed literals in the answer need not be propagated. In [3] it is shown that the propagation of delay elements as specified in Definition 3 is necessary to ensure polynomial data complexity. However, in certain cases, the propagation of delayed answers can lead to a set of unsupported answers as shown in the example below.

**Example 2.** Consider the program \( P_2 \):

\[
\begin{align*}
\text{p} & \leftarrow \text{not q}. \\
\text{p} & \leftarrow \text{p}. \\
\text{q} & \leftarrow \text{not p}. \\
\text{q}. 
\end{align*}
\]

and query \( \text{?- p} \). In the well-founded model for \( P_2 \), \( p \) is false and \( q \) is true. A forest for a possible \( SLG_{\text{variance}} \) (or \( SLG_{\text{subsumption}} \)) evaluation is shown in Figure 3, whose nodes are numbered by the order of their creation. Consider the sub-forest of Figure 3 induced by all nodes numbered 6 or less. In nodes 4 and 6 the literals not \( p \) and not \( q \) have been delayed creating conditional answers for nodes 4 and 5. The conditional answer for \( p \) (node 5) is returned to node 6 creating a second conditional answer, node 6. Subsequently, in node 7, an unconditional answer is found for \( q \), causing \( q \) to be successful (Definition 7), and a Simplification operation to be performed that creates the failure node, node 8. However, \( p \), although it is completely evaluated, cannot be determined to be false, because it has a conditional answer that depends positively on itself, or is unsupported (see Definition 5). Unsupported answers are handled through the Answer Completion operation.

**Definition 5 (Supported Answer).** Let \( F \) be a SLG forest, \( S \) a subgoal in \( F \), and \( Answer \) be an atom that occurs in the head of some answer of \( S \). Then Template is supported by \( S \) in \( F \) if and only if:

1. \( S \) is not completely evaluated; or
2. there exists an answer node \( \text{Answer} \leftarrow \text{Delay_Set} \) of \( S \) such that for every positive delay literal \( D_{\text{call}} \) of \( \text{Ans} \), \( \text{Ans} \) is supported by \( \text{Call} \).

As an aside, we note that unsupported answers appear to be uncommon in practical evaluations which minimize the use of delay such as [17].

An \( SLG_X \) evaluation consists of a (possibly transfinite) sequence of SLG forests. In order to define the behavior of an evaluation at a limit ordinal, we define a notion of a least upper bound for a set of SLG trees. If a global ordering on literals is assumed, then the elements in the Delay_Set of a node can be
uniformly ordered, and under this ordering a node of a tree can be taken as a term to which the usual definitions of term variance and term subsumption apply (See Definition 13 for details). In particular, nodes of SLG trees are treated as identical when they are variant.

A rooted tree can be viewed as a partially ordered set in which each node \( N \) is represented as \( \{ N, P \} \) in which \( P \) is a tuple representing the path from \( N \) to the root of the tree [6]. When represented in this manner, it is easily seen that when \( T_1 \) and \( T_2 \) are rooted trees, \( T_1 \subseteq T_2 \) iff \( T_1 \) is a subtree of \( T_2 \), and furthermore, that if \( T_1 \) and \( T_2 \) have the same root, their union can be defined as their set union, for \( T_1 \) and \( T_2 \) taken as sets.

**Definition 6 (Tabled Evaluation).** Given a program \( P \), an atomic query \( Q \) and a set of tabling operations (from either Definition 8 or Definition 9), a tabled evaluation \( E \) is a sequence of SLG forests \( F_0, F_1, \ldots, F_\beta \), such that:

- \( F_0 \) is the forest containing a single tree \( Q \) \( \trianglerighteq \) \( Q \).
- For each successor ordinal, \( n + 1 \leq \beta \), \( F_n \) is obtained from \( F_n \) by an application of a tabling operation.
- For each limit ordinal \( \alpha \leq \beta \), \( F_\alpha \) is defined as the set of trees \( T \) such that
  - The root of \( T_i \), \( S \trianglerighteq \{ S \} \) is the root of some tree \( F_i \), \( i < \alpha \); and
  - \( T = \cup \{ T_i \, | \, i \in F_i, i < \alpha \text{ and } T_i \text{ has root } S \trianglerighteq \{ S \} \} \).

If no operation is applicable to \( F_\alpha \), \( F_\alpha \) is called a final forest of \( E \). If \( F_\beta \) contains a leaf node with a non-ground selected negative literal, it is floundered.

SLG forests are related to interpretations in the following manner.

**Definition 7.** Let \( \mathcal{F} \) be a forest. Then the interpretation induced by \( \mathcal{F} \), \( I_\mathcal{F} \), has the following properties:

- A (ground) atom \( A \in I_\mathcal{F} \) iff \( A \) is in the ground instantiation of some unconditional answer \( \text{Ans} \trianglerighteq \{ \} \) in \( \mathcal{F} \).
- A (ground) atom not \( A \in I_\mathcal{F} \) iff \( A \) is in the ground instantiation of a completely evaluated subgoal in \( \mathcal{F} \), and \( A \) is not in the ground instantiation of any answer in \( \mathcal{F} \).
An atom \( S \) is successful in \( F \) if the tree for \( S \) has an unconditional answer \( S \). \( S \) is failed in \( F \) if \( S \) is completely evaluated in \( F \) and the tree for \( S \) contains no answers. An atom \( S \) is successful (failed) in \( I_F \) if \( S' \) (not \( S' \)) is in \( I_F \) for every \( S' \) in the ground instantiation of \( S \). A negative delay literal not \( D \) is successful (failed) in a forest \( F \) forest if \( D \) is (failed) successful in \( F \). Similarly, a positive delay literal \( D' \) is successful (failed) in a \( F \) if \( Call \) has an unconditional answer \( Ans \overset{\perp}{\leftarrow} \) in \( F \).

3  \( SLG_\text{variance} \) Variant Tabling with Delay

\( SLG_\text{variance} \) uses a variant relation on terms to determine when to add a new SLG tree to the SLG forest in the NEW SUBGOAL operation; to determine whether an answer or program clause may be used for resolution in the PROGRAM CLAUSE RESOLUTION and POSITIVE RETURN operations; and in removing a delay literal or failing a conditional answer in the SIMPLIFICATION instruction. These operations are as follows.

**Definition 8 (\( SLG_\text{variance} \) Operations).** Given a forest \( F_n \) of a \( SLG_\text{variance} \) evaluation of program \( P \) and query \( Q \), where \( n \) is a non-limit ordinal, \( F_{n+1} \) may be produced by one of the following operations.

1. **NEW SUBGOAL:** Let \( F_n \) contain a non-root node

   \[
   N = Ans \overset{\perp}{\leftarrow} \text{DelaySet} | G, \text{Goal_List}
   \]

   where \( G \) is the selected literal \( S \) or not \( S \). Assume \( F_n \) contain no tree with root subgoal \( S \). Then add the tree \( S : | S \) to \( F_n \).

2. **Program Clause Resolution:** Let \( F_n \) contain a root node \( N = S : | S \) and \( C \) be a program clause \( \text{Head} : \text{Body} \) such that Head unifies with \( S \) with \( \mu\theta \). Assume that in \( F_n \), \( N \) does not have a child \( N_{\text{child}} = (S : | \text{Body})\theta \). Then add \( N_{\text{child}} \) as a child of \( N \).

3. **Positive Return:** Let \( F_n \) contain a non-root node \( N \) whose selected literal \( S \) is positive. Let \( Ans \) be an answer node for \( S \) in \( F_n \) and \( N_{\text{child}} \) be the SLG resolvent of \( N \) and \( Ans \) on \( S \). Assume that in \( F_n \), \( N \) does not have a child \( N_{\text{child}} \). Then add \( N_{\text{child}} \) as a child of \( N \).

4. **Negative Return:** Let \( F_n \) contain a leaf node

   \[
   N = Ans \overset{\perp}{\leftarrow} \text{DelaySet} | \text{not} S, \text{Goal_List}.
   \]

   whose selected literal not \( S \) is ground.

   (a) **Negation Success:** If \( S \) is failed in \( F \), then create a child for \( N \) of the form \( Ans \overset{\perp}{\leftarrow} \text{DelaySet} | \text{Goal_List} \).

   (b) **Negation Failure:** If \( S \) succeeds in \( F \), then create a child for \( N \) of the form fail.

5. **Delaying:** Let \( F_n \) contain a leaf node \( N = Ans \overset{\perp}{\leftarrow} \text{DelaySet} | \text{not} S, \text{Goal_List} \), such that \( S \) is grounded in \( F_n \), but \( S \) is neither successful nor failed in \( F_n \). Then create a child for \( N \) of the form \( Ans \overset{\perp}{\leftarrow} \text{DelaySet} | \text{not} S | \text{Goal_List} \).

6. **Simplification:** Let \( F_n \) contain a leaf node \( N = Ans \overset{\perp}{\leftarrow} \text{DelaySet} \), and let \( L \in \text{Delayset} \).
(a) If $L$ is failed in $F$ then create a child fail for $N$.
(b) If $L$ is successful in $F$, then create a child $\text{Ans} \leftarrow \text{DelaySet}'$ for $N$, where $\text{DelaySet}' = \text{DelaySet} - L$.

7. **Completion:** Given a completely evaluated set $S$ of subgoals (Definition 4), mark the trees for all subgoals in $S$ as completed.

8. **Answer Completion:** Given a set of unsupported answers $UA$, create a failure node as a child for each answer $\text{Ans} \in UA$.

An interpretation induced by a forest (Definition 7) has its counterpart for SLG, (Definition 5.2 of [3]). Using these concepts, we can relate SLG to $SLG_{\text{variance}}$ evaluations.

**Theorem 1.** Let $P$ be a finite program and $Q$ an atomic query. Then there exists an $SLG_{\text{variance}}$ evaluation $E = S_0, ..., S_k$ of $P$ and $Q$ if and only if there exists $SLG_{\text{variance}}$ evaluation $E' = F_0, ..., F_k$ such that

$$I_{S_k} = I_{F_k}$$

**Proof.** The proof is provided in Appendix B.1.

4 **Subsumption-Based Tabling with Delay**

The variance relation on atoms is used implicitly in several $SLG_{\text{variance}}$ operations; by replacing these uses by a subsumption relation on atoms $SLG_{\text{subsumption}}$ is obtained. Specifically, in $SLG_{\text{subsumption}}$, for a New Subgoal operation to be applicable in a forest, the new subgoal must not be subsumed by any subgoal in the forest. Thus, a $SLG_{\text{variance}}$ evaluation may perform New Subgoal operations that are not necessary in $SLG_{\text{subsumption}}$. Similarly, the $SLG_{\text{subsumption}}$ Program Clause Resolution and Positive Return operations will produce a child of a node $N$ only if that child is not subsumed by any other child of $N$. Other operations are also affected. A Simplification operation may be applicable to a delay literal $D_i$ if $D \in I_F$, rather than through the conditions on delay literals specified in Definition 7. Finally, a subgoal may become completely evaluated if it is subsumed by a subgoal that is also completely evaluated. This last condition is reflected in the $SLG_{\text{subsumption}}$ Completion instruction rather than by formulating a new definition of completely evaluated.

**Definition 9 (SLGsubsumption Operations).** Given a state $F_n$ of an SSLG evaluation of program $P$ and query $Q$, where $n$ is a non-limit ordinal, $F_{n+1}$ may be produced by one of the following operations.

1. **New Subgoal:** Let $F_n$ contain a non-root node

$$N = \text{Ans} \leftarrow \text{DelaySet} \{G, \text{GoalList}\}$$

where $G$ is the selected literal $S$ or not $S$. Assume $F_n$ contain no tree with root subgoal $S'$ such that $S'$ subsumes $S$. Then add the tree $S \succ |S|$ to $F_n$.

---

1 Thus the form of positive delay literals do not need the annotations described in Definition 2, but they are maintained here for sake of a uniform representation.
2. Program Clause Resolution: Let \( F_n \) contain a root node \( N = S : \varphi \) and \( C \) be a program clause Head : Body such that Head unifies with \( S \) with mgui \( \cdot \). Assume that in \( F_n \), \( N \) does not have a child that subsumes \( N_{\text{child}} = (S : \varphi)_{\theta} \). Then add \( N_{\text{child}} \) as a child of \( N \).

3. Positive Return: Let \( F_n \) contain a non-root node \( N \) whose selected literal \( S \) is positive. Let \( \text{Ans} \) be an answer node in \( F_n \) and \( N_{\text{child}} \) be the SLG resolvent of \( N \) and \( \text{Ans} \) on \( S \). Assume that in \( F_n \), \( N \) does not have a child which subsumes \( N_{\text{child}} \). Then add \( N_{\text{child}} \) as a child of \( N \).

4. Negative Return: Let \( F_n \) contain a leaf node:

\[
N = \text{Ans} : \text{DelaySet}\lnot S, \text{GoalList}
\]

whose selected literal not \( S \) is ground.

(a) Negation Success: If not \( S \in I_T \), then create a child for \( N \) of the form: \( \text{Ans} : \text{DelaySet}\lnot \text{GoalList} \).

(b) Negation Failure: If \( S \in I_T \), then create a child for \( N \) of the form fail.

5. Delaying: Let \( F_n \) contain a leaf node \( N = \text{Ans} : \text{DelaySet}\lnot S, \text{GoalList} \), such that \( S \) is ground, in \( F_n \), but \( S \) is neither successful nor failed in \( I_T \). Then create a child for \( N \) of the form \( \text{Ans} : \text{DelaySet}\lnot S, \text{GoalList} \).

6. Simplification: Let \( F_n \) contain a leaf node: \( N = \text{Ans} : \text{DelaySet}\lnot S, \text{GoalList} \), such that \( L \in \text{DelaySet} \).

(a) If \( L \) is failed in \( I_{F_n} \), then create a child fail for \( N \).

(b) If \( L \) is successful in \( I_{F_n} \), then create a child: \( \text{Ans} : \text{DelaySet} L \) for \( N \), where \( \text{DelaySet}^L = \text{DelaySet} - L \).

7. Answer Completion: Given a set of unsupported answers \( U_A \), create a failed child for some answer \( \text{Ans} \in U_A \).

8. Completion: Let \( S \) be a set of subgoals in \( F_n \) such that for each \( S \in S \),

(a) \( S \) is completely evaluated; or

(b) \( S \) is subsumed by some subgoal \( S' \) such that the tree for \( S' \) exists in \( F_n \) and is marked as complete.

Then mark as complete the tree for each \( S \in S \).

**Theorem 2.** Let \( P \) be a program and \( Q \) a query. Then there exists an SLG\text{\textsubscript{variance}} evaluation \( \mathcal{E}^V \) of \( Q \) against \( P \) with final state \( F^V \) if and only if there exists an SLG\text{\textsubscript{subsumption}} evaluation \( \mathcal{E}^S \) of \( Q \) against \( P \) with final state \( F^S \) such that

\[
I_{F^V} = I_{F^S}
\]

SLG\text{\textsubscript{variance}} and SLG\text{\textsubscript{subsumption}} can be considered as two extreme types of tabled evaluation. From an implementational perspective, it may be useful to mix the operations from different evaluation methods. For instance, by replacing the SLG\text{\textsubscript{variance}} NEW \text{\textsc{SUBGOAL}} operation with that of SLG\text{\textsubscript{subsumption}}, an evaluation may be said to use \textit{call subsumption}. By replacing the SLG\text{\textsubscript{variance}} \text{\textsc{POSITIVE RETURN}} operation with that of SLG\text{\textsubscript{subsumption}}, an evaluation may be said to use \textit{answer subsumption}. Special cases of call and answer subsumption have been discussed in the tabled literature for definite programs (see e.g., [18]).
4.1 Relevant Subgoals: an Example of a SLG Optimization

The previous theorems hold for any SLG\textit{variance} or SLG\textit{subsumption} evaluations: that is, for any ordering of applicable operations. However, in order to apply either method to practical programs, it can be useful to restrict evaluations to have certain properties. For instance, the notion of incremental completion, of applying a completion operation as soon as possible so that space for completed trees can be reclaimed, was described in [15]. Here, we describe another optimization, based on the notion of relevance. An SLG tree that is not relevant to an initial query need have no operations performed on it and can in principle be disposed of, reclaiming space. We begin by restating the definition of the subgoal dependency graph (SDG) which provides a useful abstraction of an SLG forest.

**Definition 10 (Subgoal Dependency Graph).** Let \( F \) be a forest in a SLG\( _X \) evaluation. We say that a tabled subgoal \( S_1 \) directly depends on a tabled subgoal \( S_2 \) in \( F \) iff neither the tree for \( S_1 \) nor that for \( S_2 \) is marked as complete and \( S_2 \) is the selected literal of some node in the tree for \( S_1 \). The Subgoal Dependency Graph of \( F \), SDG(\( F \)), is a directed graph \( V,E \) in which \( V \) is the set of root goals for trees in \( F \) and \((S_i, S_j) \in E \) iff \( S_i \) directly depends on \( S_j \).

**Example 3.** Figure 4 represents the SDG for the forest of Figure 1.

![Fig. 4. SDG for Forest of Figure 1](image)

Since the dependency relation is non-symmetric, the SDG is a directed graph and can be partitioned into strongly connected components, or SCCs. In particular an independent SCC is one which depends on no other SCC.

**Example 4.** Consider the program \( P_{rel} \):

\[
p(Y) := q(Y), r(X), s(Y).
\]

\[
q(a), \quad q(b).
\]

\[
r(c), \quad s(b).
\]

The tree for \( p \) in the SLG\textit{variance} evaluation for the query \( ?- p \) is shown in Figure 5. Note that once an answer for the subgoal \( r(X) \) is obtained, further answers become irrelevant to evaluation of \( p(Y) \). However, in the case of the subgoal \( q(Y) \), answers beyond the first \( (q(a)) \) are in fact relevant for solving the goal \( p \), even though \( q(Y) \) shares no variables with \( p \).
Definition 11 captures the notion of when one query is relevant to another.

Definition 11. Let $\mathcal{F}$ be an SLG forest, $N = \text{Ans} \leftarrow \text{Delay} \_ \text{Set} \_ \text{Goal} \_ \text{List}$ be a node $\mathcal{F}$, and $S$ the root subgoal of $N$. A selected literal $L$ in Goal\_List is relevant in $N$ if

1. $\text{vars}(L) \cap \text{vars}(S) \neq \emptyset$; or
2. $\text{vars}(L) \cap \text{L'} \neq \emptyset$, where $\text{L'}$ is a non-selected literal in Goal\_List.

Next, let $(S_1, S_2)$ be an edge of SDG($\mathcal{F}$). $(S_1, S_2)$ is relevant of SDG($\mathcal{F}$) if

- $S_2$ is the selected literal of a leaf node in $S_1$; or
- $S_2$ is the selected literal of a non-leaf node $N$ in $S_1$ and $S_2$ is relevant in $N$.

A relevant path from a node $S_1$ to a node $S_2$ exists in SDG($\mathcal{F}$) if there is a path of relevant edges from $S_1$ to $S_2$. If such a path exists, we say that $S_2$ is relevant to $S_1$ in $\mathcal{F}$.

A relevant evaluation can be defined by constraining operations to act within trees that are relevant to the original query.

Definition 12 (Relevance). A VSLG evaluation, $\mathcal{F}_0^Y, \mathcal{F}_1^Y, \ldots, \mathcal{F}_X^Y$ of a query $Q$ against a program $P$ is relevant if every NEW SUBGOAL, PROGRAM CLAUSE RESOLUTION, POSITIVE RETURN, DELAYING, NEGATIVE RETURN, and SIMPLIFICATION operation $O$ creating $\mathcal{F}_n$ from $\mathcal{F}_{n-1}$ is applied to a node $N$ whose root subgoal $S$ is relevant to $Q$ in SDG($\mathcal{F}_{n-1}$).

Note that relevance is defined as a property of forests, and is thus well defined for each forest in a possibly transfinite evaluation.
Theorem 3. Let $\mathcal{F}_0^R, \mathcal{F}_1^R, \ldots, \mathcal{F}_\lambda^R$ be a relevant evaluation of a query $Q$ against a program $P$, and let $\mathcal{M}^P$ represent the well-founded model of $P$. Then

- $\mathcal{F}_\lambda^Q \models \mathcal{M}^P | Q$
- For any VSLG evaluation, $\mathcal{F}_0^V, \mathcal{F}_1^V, \ldots, \mathcal{F}_\mu^V$, of $Q$ against $P$, $\lambda \leq \mu$.

Programs and queries can be easily constructed in which relevant evaluations terminate finitely and non-relevant evaluations do not. Relevance in a tabled evaluation thus captures aspects of the Prolog cut, as well as existential query optimization for deductive databases [14]. Aspects of relevance are being explored to allow tabling logic programs to implement partial-order model checking. Relevance differs from the cut, however, in that irrelevant trees are not necessarily removed from forests; rather operations on these trees can be postponed. If this strategy of keeping uncompleted irrelevant trees is adopted, it can be shown that relevant SLG evaluations thus maintain the polynomial data-complexity properties of general SLG. How to implement relevant evaluations is still an open question, particularly the determination of whether relevance should be done dynamically as part of an engine, or should be informed by analysis.

5 Discussion

As originally formulated, SLG cannot be learned and used without a relatively large amount of intellectual commitment; the forest of trees model that underlies SLG$_X$ may well reduce the commitment needed to learn SLG. In addition, the formulation of SLG$_X$ makes it easier to formulate alternate sets of operations as has been demonstrated by the creation of SLG$_{subsumption}$. Other extensions are presently being formulated and implemented by using new sets of SLG$_X$ operations; for abduction over the well-founded semantics [1], for generalized annotated programs [10], and to formalize algorithms for distributed tabling [9]. Also, as tabling becomes used for serious implementation of powerful non-monotonic logics, program optimizations become increasingly important, along with the necessary to prove these optimizations correct. It is hoped that the reformulation described in this paper will make such efforts easier than they have been to date.

6 Acknowledgements

The author would like to thank David S. Warren and Juliana Freire for several useful discussions on the forest of trees model for SLG, and David S. Warren for suggesting the exploration of the idea of relevance.

References


The following definitions and proofs are provided so that the referees can judge the technical validity of the results; and are not intended to be part of the final paper.

A Auxiliary Definitions

**Definition 13.** Let $\mathcal{L}$ be a language, and $O_L$ be a total ordering of literals formed from predicates, function symbols, and variables in $\mathcal{L}$, as well as from the connective not. Then given a node $N$ of an SLG tree, we define $\text{Ord}_{O_L}(N)$ as

1. if $N$ is of the form

   $$\text{Answer Template} \leftarrow \text{Delay set} \lvert \text{Goal List}$$

   then $\text{Ord}(N) =$

   $$\leftarrow (\text{Answer Template}, \lvert (\text{Delay set}', \text{Goal List}))$$

   in which $\text{Delay set}'$ is the reserved atom null if $\text{Delay set}$ is empty, and is a list of the elements of $\text{Delay set}$ sorted according to $O_L$ otherwise.

2. if $N$ is of the form fail, then $\text{Ord}_{O_L}(N) = N$.

Given two nodes within an SLG forest $N_1$ and $N_2$, $N_1$ is a variant of $N_2$ (denoted $N_1 \equiv_v N_2$) if the term $\text{Ord}_{O_L}(N_1)$ is a variant of $\text{Ord}_{O_L}(N_2)$. $N_1$ is a subsumed by $N_2$ (denoted $N_1 \subseteq_s N_2$) if the term $\text{Ord}_{O_L}(N_1)$ is subsumed by $\text{Ord}_{O_L}(N_2)$.

B Proofs of Main Results

B.1 Proof of Theorem 1

**Terminology used for this proof** In order to prove that $\text{SLG}_{\text{variance}}$ forests are equivalent to SLG sequences, we relate the notation of this paper to that of the original definition of [3]. An X-rule, ([3], Definition 4.2), has the form

$$H \Rightarrow L_1 \ldots L_n$$

where each $L_i$ is a literal or a delayed literal. We say that an X-rule $N$ corresponds to an $\text{SLG}_{\text{variance}}$ node $\text{Answer Template} \leftarrow \text{Delay set} \lvert \text{Goal List}$ if the following conditions hold.

- $H = \text{Answer Template};$
- $D \in \text{Delay set}$ iff $D = L_i$ for some $1 \leq i \leq n$.
- Let $\mathcal{L}$ be the subsequence of $L_1 \ldots L_n$ from which all delay literals have been removed: $\mathcal{L} = \text{Goal List}$. 
Let $P$ be a finite program, $C$ a clause in $P$, $\alpha$ an ordinal, and $H$ an atom. Then an X-element ([3], Definition 4.3) of the form $X < C >$, $X < \alpha >$, or $X < \alpha, H >$ corresponds to a $SLG_{\variance}$ node $N$ if $X$ corresponds to $N$.

We begin by adapting Theorem 4.1 of [3] and its proof to indicate properties of $SLG_{\variance}$ at limit ordinals. Throughout this section we assume knowledge of terminology of [3].

**Definition 14.** Let $F_1$ and $F_2$ be $SLG$ forests, $F_1 \sqsubseteq_F F_2$ if

$$\forall T_1 \in F_1, \exists T_2 \in F_2, \text{ such that } T_1 \subseteq T_2.$$ 

**Lemma 1.** Let $P$ be a finite program and $Q$ an atomic query. Then

- there exists an ordinal $\alpha$ such that for any forest $F$ in a $SLG_{\variance}$ derivation for $Q$, number of nodes of any $SLG$ tree in $F$ is bounded by $\alpha$,
- every $SLG_{\variance}$ derivation of $Q$ produces a monotonically increasing sequence of forests with respect to $\sqsubseteq_F$,
- there exists some $SLG_{\variance}$ derivation for $Q$ against $P$ $F_0, \ldots, F_\beta$ such that $F_\beta$ is a final system.

**Proof.** (Sketch) We let $\Pi_P$ be the maximum number of literals in the body of a rule for $P$.

1. Let $F$ be an arbitrary $SLG_{\variance}$ forest, and $F$ a tree in $F$ with root $R$. Then the number of nodes in $F$ is bounded by the following observations.
   (a) The immediate children of $R$ are created via $\text{PROGRAM CAUSE RESOLUTION}$. Since $P$ is finite, there will be a finite number of immediate children of $R$.
   (b) The operation $\text{DELAYING}$ applied to a node produces a single child for that node. Furthermore, a given literal in a $\text{Goal-List}$ can be delayed only once, so that there are at most $\Pi_P$ nodes created by $\text{DELAYING}$ on any path from $R$.
   (c) The operations $\text{NEGATIVE RETURN}$, $\text{Simplification}$, and $\text{ANSWER COMPLETION}$ applied to a node $N$ each produce a single child for $N$, and ensure that no other operations will be applicable to $N$. In the case of $\text{ANSWER COMPLETION}$ this child will be a failure node, to which no operations are applicable. In the case of $\text{NEGATIVE RETURN}$ the child will have one less literal in its $\text{Goal-List}$ than $N$, while in the case of $\text{Simplification}$ the child will have one less literal in its $\text{Delay-set}$ than $N$.
   (d) The operation $\text{Positive Return}$ will produce at most a countable number of children for $N$, each of which will have one fewer literal in their $\text{Goal-List}$ than $N$.

These considerations indicate that each path from $R$ to a leaf of $F$ is bounded by $2\Pi_P + 2$. To see this, consider that each such path contains the root itself along with a child formed by $\text{PROGRAM CAUSE RESOLUTION}$. In addition, note that the other operations must either reduce the size of the $\text{Goal-List}$
or move a literal from the GoalList to the DelaySet. Because the depth of $F$ is finite, and the arboricity of $F$ is bounded by $L$, which is countable, there are a countable number of nodes in $F$.

2. Let $F_0, \ldots, F_\alpha$ be an SLG-variance evaluation. Consider a successor ordinal $i + 1$, for $0 < i + 1 \leq \alpha$. In this case, consider the SLG-variance operation that produced $F_{i+1}$. If the operation was NEW SUBGOAL then each tree in $F_i$ is unchanged and a new tree is added. In the case of all other operations, a tree or tree in $F_i$ is extended by adding a new node or nodes. In either case, $F_i \subseteq F_{i+1}$ when $i$ is a successor ordinal. When $i (0 < i \leq \alpha)$ is a limit ordinal, $F_i$ is defined using set union, so the property also holds.

3. Note that the number of nodes in any SLG forest for a query $Q$ against $P$ is bounded since there are a countable number of subgoals (up to variance) for $P$ and each tree for each subgoal has a countable number of nodes. Since each transformation increases the number of nodes in a forest, and since, by b, each SLG-variance derivation contains a monotonically increasing sequence of forests, there must exist a final forest for some SLG-variance derivation of $Q$ against $P$.

**Theorem 1.** Let $P$ be a finite program and $Q$ an atomic query. Then there exists an SLG evaluation $F = S_0, \ldots, S_\mu$ of $P$ and $Q$ if and only if there exists SLG-variance evaluation $F^V = F_0, \ldots, F_\nu$ such that

$$I_{S_\mu} = I_{F_\nu}$$

**Proof.** Let $F$ be a SLG-variance forest. For the purposes of this proof, we say that a node $N$ is active in $F$ iff $N$ occurs in a tree that is not completely evaluated and is either a leaf node with a non-empty goal list, or a non-leaf node with a positive selected literal.

Also, for simplicity of presentation, we assume a fixed left-to-right literal selection strategy for both $F$ and $F^V$. It is straightforward to extend the proof to other search strategies.

We begin by proving that a subgoal or answer is in $S_\mu$ only if it is in $F_\nu$ for some $\nu$. To show this, we prove by induction a slightly stronger property than stated in the theorem: that there exists a monotonic function $map$ such that for $S_i$, $1 \leq i \leq \mu$, there exists a $F_{map(i)}$ such that

1. a subgoal $S$ is in $S_i$ iff $S$ is in $F_{map(i)}$;
2. for each subgoal $S$ in $S_i$,
   (a) $E$ is an answer for $S$ in $S_i$ iff $N_E$ is an answer for $S$ in $F_{map(i)}$ and $E$ corresponds to $N_E$.
   (b) $E$ is an active element for $S$ in $S_i$ iff $N_E$ is an active node for $S$ in $F_{map(i)}$ and $E$ corresponds to $N_E$.
3. a subgoal $S$ is completely evaluated in $S_i$ (i.e., according to Definition 4.10 of [3]) only if $S$ is completely evaluated in $F_{map(i)}$ (i.e., according to Definition 4).

Note that if the induction hypothesis holds for a system $S_i$ and forest $F_{map(i)}$, then $I_{S_i} = I_{F_{map(i)}}$. 


For the base case ($i = 1$), we note that since $S_0$ is empty, only the SLG 
NEW SUBGOAL transformation for the initial query $Q$ is applicable. Then there are $k$
program clauses $C_i$ that resolve against $Q$, the SLG 
NEW SUBGOAL transformation creates $k$ new X-elements $G < C_i >$ such that $G$ is an X-rule of
the form $H \leftarrow L_1, \ldots, L_n$. Because $F_0$ contains a tree with root $Q \triangleright Q$, $F_{\text{map}(1)}$ is
constructed by performing PROGRAM CLAUSE RESOLUTION for each program clause whose head unifies with $S$. Since SLG 
X-resolution operates on X-rules in the same way that the SLG \textit{variance} PROGRAM CLAUSE RESOLUTION operates
on nodes, it is straightforward to see that properties 1-3 will continue to hold when $i = 1$.

Next, we assume that the statement holds for all $m < n$ to show that the
statement is maintained at $n$.

If $n$ is a successor ordinal, we consider the cases of SLG transformations of
a system $S_{n-1}$

- NEW SUBGOAL. Suppose $S$ is the subgoal for which the NEW SUBGOAL trans-
formation is applied. Note that NEW SUBGOAL affects only the pair $(S : \rho)$
in $S_{n-1}$ and by definition, $S$ is not in $S_{n-1}$. If there are $k$ program clauses $C_i$
that resolve against $S$, the SLG NEW SUBGOAL transformation creates $k$ new
X-elements $G_i < C_i >$ such that $G$ is an X-rule of the form $H \leftarrow L_1, \ldots, L_n$.
Next consider $F_{n-1}$. By the induction hypothesis, $F_{\text{map}(n-1)}$ will not contain
$S$ as the root of any tree, and $F_{\text{map}(n)}$ is constructed by first performing a
SLG \textit{variance} NEW SUBGOAL operation to create a tree for $S$ to preserve equiva-
ence of subgoals. Afterwards, PROGRAM CLAUSE RESOLUTION is performed at
most $k$ times to ensure that the sets of answers and active rules/nodes are
identical up to correspondence.

We verify that this action preserves the three properties of the induction
hypothesis.
1. That a subgoal $S$ is in $S_n$ iff $S$ is in $F_{\text{map}(n)}$ is straightforward from
the induction hypothesis and the construction outlined above.
2. That an active element or answer $E$ is in $S_n$ iff $E$ is in $F_{\text{map}(n)}$ is also
straightforward.
3. No subgoal will become SLG completely evaluated in $S_n$ unless $S$ be-
comes completely evaluated in $S_n$. We first consider the cases where $S$ will
become completely evaluated. For this to happen, for each $G_i < C_i >$
produced by the SLG NEW SUBGOAL transformation, either $G_i$ does not
have a selected literal, or the selected literal is SLG completely eval-
uated with no answers. Because PROGRAM CLAUSE RESOLUTION acts
upon SLG \textit{variance} nodes in the same way that X-resolution acts upon
X-rules, there are at most $k$ children $N_i$ such that $N_j$ corresponds to
one of the above $G_i$. Furthermore, if $G < C_i >$ has a selected literal
$L_i$ which is SLG \textit{variance} completely evaluated and has no answers, then
there will be a unique child $N_j$ of $S$ in $F_{\text{map}(n)}$ such that $N_j$ corresponds
to $G_i$ with selected literal $L_i$. By the induction assumption, $L_i$ will be
completely evaluated in $F_{\text{map}(n)}$ also, and the induction statement will
continue to hold. This shows that if $S$ is completely evaluated in $S_n$, it
will be completely evaluated in $F_n$, also.
In order to show that this property holds for subgoals in \( S_n \) that have
\( S \) as a selected literal, we first consider that, if our induction hypothesis
holds, then given an active X-element \( X \) in \( S_{n-1} \) and a corresponding
active node \( N \) in \( F_{n-1} \), then a SLG transformation is applicable to \( X 
\) only if a similar SLG\textsubscript{variance} operation is applicable to \( N \). This can be
seen in a straightforward manner by comparing Definition 8 to the trans-
formations of [3]. Now, suppose a given subgoal \( S' \) not equal to \( S \) became
completely evaluated in \( S_n \). By part 1 of the induction hypothesis, \( S' \) is
in \( F_n \). By part 2 of the induction hypothesis, every node for \( S' \) in \( F_n 
\) corresponds to an X-element for \( S' \) in \( S_n \). By the considerations above,
no operation is applicable to a node in \( F_n \) unless a similar operation is
applicable to the corresponding node in \( S_n \). Thus, a contradiction is
derived, and \( S' \) must be completely evaluated in \( F_n \). We note that the fact
that \( S' \) is completely evaluated in \( S_n \) only if \( S' \) is completely evaluated
in \( F_n \) does not depend on the the NEW SUBGOAL operation. It will be
used in other cases of the proof, where it will be referred to as Property
1.1.

- **Positive Return** Suppose the SLG positive return transformation adds
an X-element \( G < a, H > \) to the X-sequence \( (S : p) \), and that \( G \) is created
by X-resolution or X-factoring of \( H \) against the selected literal \( L_i \) of an X-
rule \( X = S' : L_1, \ldots, L_m \). We note that positive return affects only the
pair \( (S : p) \).

1. It is trivial to see that the induction hypothesis preserves equivalence of
subgoals.

2. By the induction assumption, an active node \( N \) corresponding to \( X \) is
in \( F_{\text{map}(n-1)} \), and \( H \) is an answer for the subgoal of the selected
literal of \( N \). Let \( N_{\text{child}} \) designate the SLG\textsubscript{variance} answer resolvent of \( N \) and
\( A \) on \( L_i \). A new child \( N_{\text{child}} \) is created for \( N \) in \( F_{\text{map}(n)} \) iff \( N_{\text{child}} \)
is not a variant of another child of \( N \), so that \( \text{map}(n) = \text{map}(n - 1) \)
if there is such a variant and \( \text{map}(n) = \text{map}(n - 1) + 1 \) if \( N_{\text{child}} \)
is added as a child. Furthermore, it is straightforward from the definition of
SLG\textsubscript{variance} Answer Resolution that it captures X-resolution and X-
factoring (Definitions 4.7 and 4.9 of [3]) so that \( N_{\text{child}} \) will correspond to
the X-element produced in \( S_n \). It is clear to see that whether or not a
child is produced for \( N \), property 2 is preserved.

3. For reasons similar to part 3 for the case of NEW SUBGOAL, we may
restrict our attention to \( S \) becoming completely evaluated in \( S_n \). If \( S 
\) becomes completely evaluated in \( S_n \), we need to show that \( S \) is completely
evaluated in \( F_{\text{map}(n)} \). As in the previous paragraph, we consider
two cases. The case in which \( \text{map}(n) = \text{map}(n - 1) \) follows trivially from
the induction hypothesis. If \( \text{map}(n) = \text{map}(n - 1) + 1 \), because then
the newly created X-element must either have no selected literal, or have
a completely evaluated selected literal with no answers in \( S_n \). Because
SLG\textsubscript{variance} Answer Resolution captures X-resolution and X-factoring,
and because of the induction hypothesis, both part 3 will continue to hold
in $F_n$ for the subgoal $S$ in both of these cases. For all other subgoals, Property 1.1, introduced above, applies.

- **Negative Return.** In this case, suppose the SLG negative return transformation is applied to an X-rule $X = H \leftarrow L_1, \ldots, L_n$ associated with some subgoal $S$, such that $X$ has a selected ground literal not $A_i$.

1. Since **Negative Return** affects only the pair $(S : \rho)$, it is trivial to see that the induction hypothesis preserves equivalence of subgoals.

2. By parts 1 and 2 of the induction assumption, the subgoal $A_i$ exists in $F_{map(n-1)}$, and an active node $N$ corresponding to $X$ is in $F_{map(n-1)}$. There are two sub-cases of **Negative Return** depending on whether $A_i$ is successful or failed in $S_{n-1}$. If $A_i$ is successful in $S_{n-1}$, it will be successful in $F_{map(n-1)}$ by part 2 of the induction hypothesis. Similarly, if $A_i$ is failed in $S_{n-1}$ it must be completely evaluated and contain no answers. That $S$ is completely evaluated in $F_{map(n-1)}$ follows from part 3 of the induction hypothesis, and that it contains no answers follows from part 2 of the induction hypothesis.

   If $A_i$ is failed, then an X-rule $X = H \leftarrow L_1, \ldots, L_n$ will be created in $S_n$, and a corresponding child of $N$ will be created in $F_{map(n)}$. The creation of a child for $N$ will ensure the continued correspondence of active X-rules and active nodes, along with the correspondence of answers. Likewise, if $A_i$ is successful, then $X$ will be marked as disposed in $S_n$, and $N$ will have a child that is a failure node in $F_{map(n)}$. Neither the node nor the X-rule will remain active, so that part 2 will continue to hold.

3. Suppose $S$ becomes completely evaluated in $S_n$. There are two cases. In the first case, $A_i$ is failed in $S_{n-1}$ and $F_{map(n-1)}$. A new child is produced for $N$ and the statement will hold in for $n$ by an argument similar to the **New Subgoal** case. If $A_i$ is successful, a failure node will be produced as a child for $N$ in $F_{map(n)}$. No further operations are applicable to either $N$ or the child, and this fact, together with the induction assumption shows that $S$ will be completely evaluated in $F_{map(n)}$. For all other subgoals, Property 1.1, introduced above, applies.

- **Delaying Verification** that each part of the induction hypothesis will continue to hold at $F_{map(n)}$ is essentially similar to previous cases.

- **Simplification.** In this case, suppose the SLG **Simplification** operation is performed on a delayed literal $L$ of an X-element $G$ for a completed subgoal $S$ in $S_{n-1}$. We first note that $S$ can be completed in $S_{n-1}$ only if $S$ is completely evaluated; by part 3 of the induction hypothesis $S$ will also be completely evaluated in $F_{map(n-1)}$. Part 2 ensures that a node $N_G$ corresponding to $G$ will also exist in $F_{map(n-1)}$ and also ensures that $L$ will be successful in $S_{n-1}$ if it is successful in $F_{map(n-1)}$. Parts 2 and 3 ensure that $L$ will be failed in $S_{n-1}$ only if $L$ is failed in $F_{map(n-1)}$.

Also, **Simplification** does add any subgoals to $S_{n-1}$ or $F_{map(n-1)}$ so that part 1 will continue to hold. Also, note that no subgoal will become completely evaluated in $S_n$ that was not completely evaluated in $S_{n-1}$, so that part 3 of the induction hypothesis trivially holds.

There are then several sub-cases to consider for part 2.
• $L$ is positive and successful in $S_{n-1}$. In this case, $L$ is also successful in $F_{n-1}$. Furthermore, Definition 8 ensures that the new node produced in $F_{\text{map}(n)}$ will correspond to that produced for $S_n$.

• $L$ is positive and failed in $S_{n-1}$. $L$ is also failed in $F_{\text{map}(n-1)}$. $G$ will be disposed, and $N_G$ will have a failure node as a child. In either case, the correspondence of answers will be preserved.

• If $L$ is negative, the reasoning is similar to the two cases in which it is positive.

- **Completion** By part 3 of the induction hypothesis, if a set $S$ of subgoals are completely evaluated in $S_{n-1}$, then $S$ will be completely evaluated in $F_{\text{map}(n-1)}$, so that a COMPLETION operation can be applied to $F_{\text{map}(n-1)}$ and $\text{map}(n) = \text{map}(n-1) + 1$. It is straightforward to observe that all parts of the induction hypothesis will continue to hold.

- **Answer Completion** Because answer completion applies only to answers for completely evaluated subgoals, parts 1 and 3 of the induction hypothesis will trivially continue to hold in $S_n$ and $F_{\text{map}(n)}$.

For part 2, note that the definition of supported answers in $SLG_{\text{variance}}$ (Definition 5) corresponds to that of $SLG$ 4.11 and that both rely on the notions of complete evaluation and correspondence of answers. Thus, a set $A$ of answers will be unsupported in $S_{n-1}$ only if there is an corresponding set of answers in $F_{\text{map}(n-1)}$ that is unsupported. Since the actions of answer completion are similar in $SLG$ and $SLG_{\text{variance}}$, part 2 of the induction hypothesis will continue to hold for $n$.

This concludes the cases in which $n$ is a successor ordinal, and we now consider the case where $n$ is a limit ordinal. First, note that if $n$ is a successor ordinal, then $\text{map}(n)$ is also a successor ordinal. To see this note that for every operation except NEW SUBGOAL $\text{map}(n)$ is equal to either $\text{map}(n - 1)$ or to $\text{map}(n - 1) + 1$. In the case of NEW SUBGOAL, $\text{map}(n)$ is equal to $\text{map}(n - 1) + k$ where $k$ is bounded by the number of program clauses, which is finite.

We can then define $\text{map}(n)$ to be the least upper bound of $\text{map}(m)$ such that $m < n$. Using the property that successors map to successors, we can see that $\text{map}(n) = n$, when $n$ is a limit ordinal.

1. For part 1, suppose a subgoal $S$ is in $S_n$. Then, there is some successor ordinal $m < n$ such that $S$ is in $S_m$ [3], and by the induction hypothesis $S$ is in $F_{\text{map}(m)}$, and because $F_n$ is the least upper bound of a sequence of forests monotonically increasing on $\subseteq F$, (Lemma 1), $S$ is also in $F_{\text{map}(n)}$. The same considerations show that if $S$ is a subgoal in $F_n$, it will also be a subgoal in $S_n$.

2. The argument for part 2 is similar to that for part 1, and relies on the monotonicity of $SLG$ systems within an $SLG$ evaluation, on the monotonicity of $SLG_{\text{variance}}$ forests within an $SLG_{\text{variance}}$ evaluation, and on the induction hypothesis.

3. For part 3, suppose a subgoal $S$ is completely evaluated in $S_n$ but not in $F_n$. Then, by the induction hypothesis, $S$ cannot be completely evaluated in any $S_n$ for $n < n$. Then the following conditions must hold. There exists some
subgoal $S'$ such that $S' = S$, or $S$ depends on $S'$. There is some node $N$ in the tree for $S'$ such that there is an applicable New Subgoal, Program Clause Resolution, Positive Return, or Negative Return operation that has not been performed on $N$, but which was performed on the corresponding X-element in the sequence for $S$ in $S_n$. Because of the definition of $S_n$ such an operation was applied to some system $S_n$ for a successor ordinal $n$. However, if this had happened, then part 2 or 3 of the induction hypothesis would not have held in $S_{n+1}$.

Because an operation is no $SLG_{variance}$ operation is applicable to $F_n$ unless a similar SLG transformation is applicable to the corresponding X-element in $F_n$, (see the argument for part 3 of New Subgoal) when $E$ reaches a final system, $E_V$ will also have reached a final system.

The proof can be completed by showing that given an $SLG_{variance}$ evaluation $E^V$ an SLG evaluation $E$ can be constructed so that interpretation induced by the final system of $E$ is equal to the that induced by the final forest of $E^V$. Due to the strength of the induction hypothesis used in the first part of the proof, this can be proven by induction on $E^V$ using a hypothesis similar to that of the first part of the proof.

B.2 Proof of Theorem 2

The proof of Theorem 2 relies heavily on the notion of whether a particular forest of SLG trees covers another.

**Definition 15.** Let $T_1$ and $T_2$ be SLG trees. Then $T_1 \sqsubseteq C T_2$ ($T_1$ is covered by $T_2$) if

- For each non-root node $N_1$ in $T_1$, there exists a node $N_2$ in $T_2$, such that
  - $N_1$ is subsumed by $N_2$ (Definition 13);
  - $N_1$ and $N_2$ contain the same number of ancestors; and
  - each ancestor of $N_1$ is subsumed by an ancestor of $N_2$.

If the above conditions hold for nodes $N_1$ in tree $T_1$ and $N_2$ in tree $T_2$, then $N_1$ in $T_1$ is said to be covered by $N_2$ in $T_2$. A forest $F_1 \sqsubseteq C F_2$ if for each $T_i \in F_1$, there exists a $T_j \in F_2$ such that $T_i \sqsubseteq C T_j$. Finally, $TF_1 \sqsubseteq C TF_2$ if $T_i \sqsubseteq C T_j$ and $T_j \sqsubseteq C T_i$, where $T_i$ and $T_j$ can either be a pair of trees or of forests.

As an example of the above notion, the tree for $p(X)$ in Figure 2 covers the tree for $p(a)$. We first prove some simple notions about covering.

**Lemma 2.** Let $T_1$, $T_2$ be two SLG trees.

1. If $T_1 \sqsubseteq C T_2$, let $I_{T_1}$ and $I_{T_2}$ be the interpretations induced by the forests containing the single trees $T_1$ and $T_2$ respectively, $I_{T_1} \sqsubseteq C I_{T_2}$.

2. Assume $T_1$ and $T_2$ were both produced by either a $SLG_{variance}$ or $SLG_{subsumption}$ evaluation. If the root subgoal of $T_2$ subsumes that of $T_1$, then $T_1 \sqsubseteq C T_2$ and $I_{T_1} \sqsubseteq C I_{T_2}$. 

Proof. 1. Let \( L \) be a positive literal in \( I_{T_1} \). Then, by Definition 7, \( L \) is in the ground instantiation of some answer \( A_1 \) in \( T_1 \). By condition 1, \( A \) is subsumed by some \( A_2 \in T_2 \). By Definition 13 \( A_2 \) is also an answer, and \( L \) is in the ground instantiation of \( A_2 \). Let \( L \) be a positive literal in \( I_{T_1} \). Then \( L \) is in the ground instantiation of the root subgoal of \( T_1 \), but not in the ground instantiation of any answer of \( T_1 \). Since \( T_1 \sqsubseteq T_2 \), \( L \) is in the ground instantiation of the root subgoal of \( T_2 \). Since \( T_2 \sqsubseteq T_1 \), \( L \) is not in the ground instantiation of any answer in \( T_1 \).

2. Proof is straightforward from Definition 8 or 9.

We next restate lemma 1 for \( SLG_{subsumption} \).

Lemma 3. Let \( P \) be a finite program and \( Q \) an atomic query. Then

- there exists an ordinal \( \alpha \) such that for any forest \( F \) in a \( SLG_{subsumption} \) derivation for \( Q \), number of nodes of any \( SLG \) tree in \( F \) is bounded by \( \alpha \);
- every \( SLG_{subsumption} \) derivation of \( Q \) produces a monotonically increasing sequence of forests with respect to \( \sqsubseteq_F \);
- there exists some \( SLG_{subsumption} \) derivation for \( Q \) against \( P \) \( F_0, \ldots, F_\beta \) such that \( F_\beta \) is a final system.

The proof of Lemma 3 is essentially the same as that of Lemma 1.

In general, \( SLG_{variance} \) operations require more steps than \( SLG_{subsumption} \) evaluations. The idea behind the proof of Theorem 2 is that at various steps of the \( SLG_{variance} \) evaluation, the corresponding \( SLG_{variance} \) evaluation may perform a number of steps to “catch up” with the \( SLG_{subsumption} \) evaluation.

Theorem 2. Let \( P \) be a finite program and \( Q \) a query. Then there exists a \( SLG_{variance} \) evaluation, \( \mathcal{E}_V \), of \( Q \) against \( P \) with final state \( F_V \) if and only if there exists an \( SLG_{subsumption} \) evaluation \( \mathcal{E}_S \) of \( Q \) against \( P \) with final state \( F_S \) such that

\[
I_{F_V} = I_{F_S}
\]

Proof. We begin by considering an arbitrary \( SLG_{subsumption} \) evaluation \( \mathcal{E}_S \) and showing how to construct a \( SLG_{variance} \) evaluation \( \mathcal{E}_V \) whose final system has the same interpretation as the final system of \( \mathcal{E}_S \). We show this by induction on the forests of \( \mathcal{E}_S \).

For our induction statement, we construct a monotonic function \( map \) which maps ordinals to ordinals, and show that for each \( F_S \), there exists a \( F_{map(i)} \) such that

1. For \( T_S \in F_S \), if the root subgoal \( S \) of \( T_S \) is not subsumed by the root subgoal of any other tree in \( F_S \), then there exists a \( T_Y \in cF_V \) such that \( T_S \equiv C T_Y \);
2. A subgoal \( S \) is in \( F_S \) only if it is in \( F_V \);
3. If any \( SLG_{variance} \) operation is applied to \( F_V \), then it is not the case that \( F_{map(i)} \sqsubseteq F_S \);
4. if \( S \) is completely evaluated in \( F_S \), then it will be completely evaluated in \( F_V \).
Note that the induction statement implies that $I_{F_{map(n)}} = I_{F_n}$. To see this, note that part 1 of the induction statement states that if $S$ is a root subgoal of a tree $T_S^S \in F_n$ such that $S$ not subsumed by any other subgoal in $F_n^S$, then there exists $T_S^S \in F_{map(n)}^S$ such that $T_S^S \equiv_c T_S^S$. By Lemma 2 part 2, the union of all such trees will be $I_{F_n^S}$, and by Lemma 2 part 1, $I_{F_n^S} \equiv_c I_{T_S^S}$. By part 2 of the induction hypothesis, a subgoal is in $F_{map(n)}^V$ only if it is in $F_n^V$. Thus, the union of interpretations all such trees is equal to $I_{F_{map(n)}}$.

We begin our proof by assuming that $n$ is a successor ordinal. In the base case, $F_0^S$ consists of the forest $Q \preceq |Q|$, as does the forest $F_1^V$, so that $map(0) = 0$, and the induction statement holds trivially. Next, we assume that the statement holds for all $m < n$ to show that the statement is maintained at $n$.

- NEW SUBGOAL. Suppose $S$ is the subgoal for which the $SLG_{subsumption}$ NEW SUBGOAL transformation is applicable. Then $S$ must occur in the selected literal of a node $N^S$ in a tree with subgoal $S_{root}$ in $F_{n-1}^S$. By property 2, $S_{root}$ is a subgoal in $F_{map(n-1)}^S$. By property 3, all operations needed to produce $N^V$ will have been performed for $F_{map(n-1)}^V$, so that there will be a variant node $N^V$ in the tree for $S_{root}$ in $F_{map(n-1)}^V$. Now clearly, if a new subgoal operation is applicable for $S$ in $F_{n-1}^S$, it will be applicable in $F_{map(n-1)}^V$ — otherwise $F_{map(n-1)}^V$ would contain a tree with root subgoal $S$ and part 2 of the induction hypothesis would not hold. Thus $F_{map(n)}^V$ is produced by applying the appropriate new subgoal operation to $F_{map(n-1)}^V$ and $map(n) = map(n-1) + 1$. It is easy to see that parts 1-3 of the induction statement continues to hold at $n$.

For part 4, we note that no subgoal will become $SLG_{subsumption}$ completely evaluated in $F_n^S$ unless $S$ itself becomes completely evaluated in $F_n^S$. $S$ can become completely evaluated only if there is a program clause resolution operation applicable to the root. By inspecting Definitions 8 and 9, it can be seen that an $SLG_{subsumption}$ program clause resolution operation will be applicable to $S \preceq |S|$ only if an $SLG_{variance}$ program clause resolution operation will be applicable to $S \preceq |S|$. In order to show that this property holds for other subgoals in $S_n$ that have $S$ as a selected literal, we first consider that, if our induction hypothesis holds, then given any node $N_1 S_{n-1}$ then a $SLG_{subsumption}$ transformation is applicable to $N_1$ only if a similar $SLG_{variance}$ operation is applicable to $N_1$. This can be seen in a straightforward manner by comparing Definition 8 to Definition 9. Now, suppose a given subgoal $S'$ not equal to $S$ became completely evaluated in $F_n^S$. By part 2 of the induction hypothesis, the argument above, $S'$ is in $F_{map(n)}^V$. By part 3 of the induction hypothesis, there can be no operations applicable for the tree for $S'$ in $F_{map(n)}^V$ unless they would create a tree that was not covered by a tree in $F_n^S$. Since there are no applicable operations for the tree for $S'$ in $F_n^S$, there will be no applicable operations for the tree for $S'$ in $F_n^V$. We note that the fact that $S'$ is completely evaluated in $F_n^S$ only if $S'$ is completely evaluated in $F_n^V$.
does not depend on the the new subgoal operation. It will be used in other cases of the proof, where it will be referred to as Property 2.1.

- **Program Clause Resolution** Suppose that a program clause $C$ is unified with a root $S \Rightarrow S$ of a tree in $F_{n-1}^S$, producing a node $N$. By part 2 of the induction hypothesis, $S \Rightarrow S$ is also the root of a tree in $F_{map(n-1)}^V$.

There are two situations to consider depending on whether $S$ is subsumed by the root subgoal of any other tree in $F_{n-1}^S$ or not.

- **Situation 1** We begin by assuming that $S$ is not subsumed by any other subgoal in $F_{n-1}^S$. In this situation, a program clause resolution step is also applicable in $F_{map(n-1)}^V$. By applying this step, part 1 of the induction hypothesis continues to hold. A possibly infinite number of $SLG_{variance}$ steps must be performed to maintain part 3. To begin with, any other $SLG_{variance}$ program clause resolution steps must be performed on $S \Rightarrow S$ that produce nodes subsumed by $N$, since these operations are not applicable in $F_S$. Next, consider a tree in $F_V$ for a subgoal $S'$ which is subsumed by $S$. The tree for $S'$ may not exist in $F_{n-1}^S$, so $SLG_{variance}$ program clause resolution steps must also be performed to produce nodes subsumed by $N$. At this point, let $F'$ be the forest created from $F_{map(n-1)}^V$ in these two steps, and let $N'$ be the set of nodes created in these two steps. If some node in $N'$ contains a selected literal whose subgoal $S''$ is subsumed by the root subgoal of some tree in $F_{n}^S$, a new subgoal operation must be performed for $S''$, and operations performed on this tree as long as doing so keeps the tree covered by $F_{n}^S$. If a tree for some $S''$ is infinite, $map(n)$ may be a limit ordinal even if $n$ is not. However, since there are a finite number of subgoals which $S$ subsumes, and since any $SLG_{variance}$ tree must be countably infinite, part 2 of the induction hypothesis continues to hold at some ordinal.

- **Situation 2** Alternately, consider the situation in which $S$ is subsumed by another subgoal in $F_{n-1}^S$. In this case, a program clause resolution step may or may not be applicable to $F_{map(n-1)}^V$. Indeed, if the tree for $S$ in $F_{n}^S$ is covered by a tree in $F_{n-1}^S$, the program clause resolution step will have been performed by the $SLG_{variance}$ evaluation in accordance with property 3 of the induction hypothesis. In this particular case, $map(n) = map(n-1)$ and the induction hypothesis is preserved. If the tree for $S$ in $F_{n}^S$ is not covered by a tree in $F_{n-1}^S$, a $SLG_{variance}$ program clause resolution step will be performed on $S \Rightarrow S$ and the $SLG_{variance}$ evaluation extended to preserve property 3 of the induction hypothesis just as in the previous two paragraphs.

As for the induction hypothesis, part 2 will hold trivially. The construction ensures that parts 1 and 3 will also continue to hold. For part 4, we first consider the case in which $S$ becomes completely evaluated. According to Definition 4, there are three cases. The first is when an answer $S \Rightarrow 1$ is derived, which, by construction, will also be derived in $F_{map(n)}^V$. The second is the case in which $S$ is subsumed by some other subgoal in $F_{n}^S$, and in this
case, $S$ is completely evaluated in $F^S_{n-1}$ also, so by the induction hypothesis, $S$ will be completely evaluated in $F^V_{map(n-1)}$ and $F^V_{map(n)}$. The third case occurs when $S$ belongs to a completely evaluated set of subgoals $S$ that contains subgoal(s) other than $S$. In this case, the statement follows by Property 2.1.

- **Positive Return** Suppose in $F^S_{n-1}$, Positive Return is used to resolve an answer $Ams$ against a node $N$ in a tree $T$ for $S$ whose selected literal contains the subgoal $L$. If $S$ is subsumed by the root subgoal of some other tree in $F^S_{n-1}$, then the argument is essentially the same as in situation 1 of the Program Clause Resolution case. Otherwise, let $Ams$ be an answer node in a tree $L$ in $F^S_{n-1}$. By property 4, the tree for $L$ must be present in $F^V_{map(n-1)}$. Furthermore, the tree for $L$ is covered by a tree in $F^S_{n-1}$, either by the induction hypothesis, because $L$ is not subsumed by another root subgoal in $F^S_{n-1}$ or through part 2 of the induction hypothesis and Lemma 2. By a straightforward induction, using part 3 of the induction hypothesis, it can be shown that the tree for $L$ in $F^V_{map(n-1)}$ must also contain an answer $Ams_L$. We next must designate what node, $N^V \in F^V_{map(n-1)}$ a Positive Return operation is applied. We first note that because (for this case) $S$ is not subsumed by the root subgoal of any other tree in $F^S_{n-1}$, $F^V_{map(n-1)}$ will contain a tree $T^V$ such that $T^V \subset C T^V$ by the induction hypothesis. Thus, $T^V$ will contain at least one node $N^V$ such that $N^V$ in $T^V$ is covered by $N$ in $T$, and vice-versa. We choose one such node for our initial Positive Return operation, and will expand the rest to maintain part 3. Now, suppose the variables in $S$ and $L$ are standardized apart. Once the initial Positive Return operation is performed, a series of SLG variance operations is performed in a manner similar to the case of Program Clause Resolution.

The argument that all parts of the induction hypothesis will hold for $F^S_{n-1}$ and $F^V_{map(n-1)}$ is similar to the case for Program Clause Resolution.

- **Negative Return** Suppose that a Negative Return operation is applied to a node $N$ with selected literal not $L$ in a tree with root $S$ in $F^S_{n-1}$. By reasoning similar to that of the Positive Return case, there will be a node $N^V$ in the tree for $S$ in $F^V_{map(n-1)}$ that covers $N$ and that is covered by $N$. To show that a Negative Return operation is in fact applicable, two cases must now be considered.

1. Negation Failure $L \in L^S_{n-1}$. By Definition 7, $L : |$ must be the answer for some tree with root $L'$ in $F^S_{n-1}$ where $L'$ subsumes $L$. By arguments similar to those made in the case of Positive Return, $L \in L^V_{map(n-1)}$. Furthermore, by part 3 of the induction hypothesis, $F^V_{map(n-1)}$ has been extended to also produce a tree with root subgoal $L$ such with answer $L : |$. Thus, a Negative Return operation will be applicable to $N^V$. The argument that the induction hypothesis will continue to hold, is the same as for previous statements.
2. **Negation Success** not \( L_i \in I_{F^S_{n-1}} \). In this case, the induction hypothesis implies that not \( L_i \in I_{F^V_{\text{map}(n-1)}} \). This tree is covered by some tree in \( F^S_{n-1} \) by part 1, and is completely evaluated by part 4. Furthermore, a node \( N^V \) in a tree for \( S \) will exist in \( F^V_{\text{map}(n-1)} \) by an argument similar to previous cases. Thus a **Negation Return** will be applicable to \( N \) in \( F^V_{\text{map}(n-1)} \). Again, the **SLG variance** evaluation is extended to perform **Negation Return** on any trees in \( F^V_{\text{map}(n-1)} \) that are covered by the tree for \( S \), in the same manner as for previous operations, and the induction hypothesis will continue to hold, as for previous statements.

Once the initial **Negation Return** operation is performed, a series of **SLG variance** operations is performed in a manner similar to the case of **Program Clause Resolution**. Verification that the induction hypothesis continues to hold is the same as in the case of **Program Clause Resolution**.

- **Delaying** Verification that each part of the induction hypothesis will continue to hold at \( F^V_{\text{map}(n)} \) is essentially similar to previous cases.

- **Simplification** Suppose that **Simplification** is applied to a delayed literal \( L \) of an answer \( N \) in a tree for a subgoal \( S \) in \( F^S_{\text{map}(n-1)} \). By reasoning similar to that of the **Positive Return** case, there will be an answer \( N^V \) in the tree for \( S \) in \( F^V_{\text{map}(n-1)} \) that covers \( N \) and that is covered by \( N^V \).

We now consider the various cases of simplification.

1. The delayed literal \( L = \text{Sub} \langle \text{Call} \rangle \) is positive.
   (a) \( L \in I_{F^S_{n-1}} \). By the induction hypothesis, \( L \in I_{F^V_{\text{map}(n-1)}} \), and by an argument similar to the **Negation Failure** case of **Negation Return**, there is an answer \( \text{Ans} \) in the tree for **Call**. A **Simplification** operation is then applied to \( L \) in \( N^V \).
   (b) not \( L \in I_{F^S_{n-1}} \). In this case not \( L \in I_{F^V_{\text{map}(n-1)}} \), and it remains to be shown that **Call** is failed in \( F^V_{\text{map}(n-1)} \), which is done by an argument similar to the **Negation Success** case of **Negation Return**. A **Simplification** operation is then applied to the delayed literal \( L \) in \( N^V \).

In either case, additional simplification operations may need to be performed on nodes in \( F^V_{\text{map}(n-1)} \) to preserve property 3 of the induction hypothesis.

2. The arguments used for the case in which the delayed literal \( L \) is negative are analogous to those used for the case in which it is positive.

Verification that the induction hypothesis continues to hold is the same as in the case of **Program Clause Resolution** with the exception that parts 2 and 4 are trivial, since **Simplification** affects only trees that are already completely evaluated and will create no new subgoals.

- **Completion** By part 4 of the induction hypothesis, if a set \( S \) of subgoals are completely evaluated in \( F^S_{n-1} \), then \( S \) will be completely evaluated in \( F^V_{\text{map}(n-1)} \), so that a **Completion** operation can be applied to \( F^V_{\text{map}(n-1)} \) and \( \text{map}(n) = \text{map}(n-1) + 1 \). It is straightforward to observe that all parts of the induction hypothesis will continue to hold.
Answer Completion Let \( \mathcal{U}_A \) be a set of unsupported answers in \( \mathcal{F}^{S}_{n-1} \), let \( A \in \mathcal{U}_A \), and let \( A = \text{Template} \cdot \text{Delay set} \). Furthermore, assume that \( A \) occurs in a tree with root subgoal \( S_A \). By reasoning similar to that of the Positive Return case, there will be an answer \( A^V \) in the tree for \( S \) in \( \mathcal{F}^{V}_{\text{map}(n-1)} \) that covers \( A \) and that is covered by \( A \). Then according to Definition 5, there exists a positive literal \( L \in \text{Delay set} \) such that there is a subgoal \( S^L \) in \( \mathcal{F}^{S}_{n-1} \) that is completely evaluated. Again by reasoning similar to that of the Positive Return case, there will be an answer \( A^V \) in the tree for \( S \) in \( \mathcal{F}^{V}_{\text{map}(n-1)} \) that covers \( L \) and that is covered by \( L \). Furthermore, by part 4 of the induction hypothesis, \( S^L \) is completely evaluated in \( \mathcal{F}^{V}_{\text{map}(n-1)} \).

Thus, for arbitrary \( A \in \mathcal{U}_A \), it can be shown that \( A \) exists in \( \mathcal{F}^{V}_{\text{map}(n-1)} \) and is unsupported. Thus \( \mathcal{U}_A \) will also be unfounded in \( \mathcal{F}^{V}_{\text{map}(n-1)} \) and an Answer Completion operation will be applicable in \( \mathcal{F}^{V}_{\text{map}(n-1)} \).

Verification that the induction hypothesis continues to hold is the same as in the case of Program Clause Resolution with the exception that parts 2 and 4 are trivial, since Answer Completion affects only trees that are already completely evaluated and will create no new subgoals.

This concludes the cases in which \( n \) is a successor ordinal, and we now consider the case where \( n \) is a limit ordinal. We define \( \text{map}(n) \) to be the least upper bound of \( \text{map}(m) \) such that \( m < n \). We can see that when \( n \) is a limit ordinal, \( \text{map}(n) \) will also be a limit ordinal.

1. For part 1, suppose a tree \( T^V \) is in \( \mathcal{F}^{V}_{\text{map}(3)} \), but that \( T^V \) is not covered by a tree in \( \mathcal{F}^{V}_{3} \). Let \( N \) be a node of minimal depth in \( T^V \) such that \( N \) is not covered by any node of any tree in \( \mathcal{F}^{V}_{3} \). Because \( \mathcal{F}^{V}_{\text{map}(n)} \) is the least upper bound of a sequence of forests monotonically increasing on \( \subseteq \) (Lemma 3) there was some successor ordinal \( n \) such that \( N \) first appeared in \( \mathcal{F}^{V}_{\text{map}(n)} \), contradicting parts 1 or 3 of the induction hypothesis. The case in which a tree \( T^S \) in \( \mathcal{F}^{S}_{3} \) is not covered by a tree in \( \mathcal{F}^{S}_{3} \) can be handled similarly.

2. For part 2, suppose a subgoal \( S \) is in \( \mathcal{F}^{S}_{n} \). Then, there is some successor ordinal \( m < n \) such that \( S \) is in \( \mathcal{F}^{S}_{m} \) and by the induction hypothesis \( S \) is in \( \mathcal{F}^{S}_{\text{map}(m)} \), and because \( \mathcal{F}^{S}_{\text{map}(n)} \) is the least upper bound of a sequence of forests monotonically increasing on \( \subseteq \) (Lemma 3), \( S \) is also in \( \mathcal{F}^{V}_{\text{map}(n)} \).

3. The argument for part 3 is similar to that of parts 1 and 2, but relies on both Lemma 1 and 3.

4. For part 4, suppose a subgoal \( S \) is completely evaluated in \( \mathcal{F}^{S}_{n} \) but not in \( \mathcal{F}^{V}_{\text{map}(n)} \). Then, by the induction hypothesis, \( S \) cannot be completely evaluated in any \( \mathcal{F}^{S}_{m} \) for \( m < n \). Then the following conditions must hold. There exists some subgoal \( S' \) in \( \mathcal{F}^{V}_{\text{map}(n)} \) such that \( S' = S \), or \( S \) depends on \( S' \). There is some node \( N \) in the tree for \( S' \) such that there is an applicable New Subgoal, Program Clause Resolution, Positive Return, or Negative Return operation that has not been performed on \( N \), but which was performed on the corresponding node in the tree for \( S \) in \( \mathcal{F}^{S}_{n} \). However, if
this had happened, then part 1 or 3 of the induction hypothesis would not have held in $\mathcal{F}_{n+1}$.

The proof can be completed by showing that given an $SLG_{\text{variance}}$ evaluation $\mathcal{E}^V$ an $SLG_{\text{subsumption}}$ evaluation $\mathcal{E}^S$ can be constructed so that interpretation induced by the final system of $\mathcal{E}^S$ is equal to the that induced by the final forest of $\mathcal{E}^V$. Due to the strength of the induction hypothesis used in the first part of the proof, this can be proven by induction on $\mathcal{E}^V$ using a hypothesis similar to that of the first part of the proof.