

Concurrent and Local Evaluation of Normal Programs

Rui Marques¹ and Terrance Swift²

¹ CITI, Dep. Informática — FCT, Universidade Nova de Lisboa

² CENTRIA — Universidade Nova de Lisboa

Abstract. Tabled evaluations can incorporate a number of features, including tabled negation, reduction with respect to the well-founded model, tabled constraints and answer subsumption. Many of these features are most efficiently evaluated using the Local evaluation strategy, which fully evaluates each mutually dependent set of tabled subgoals before returning answers to other subgoals outside of that set. In this paper, we introduce a formalism, Concurrent Local SLG by which multiple threads of computation concurrently perform Local evaluation of the well-founded semantics, and which is the framework for multi-threaded tabling in the XSB system. We first prove several properties of Local evaluation within a traditional model of tabled computation. We then introduce *Concurrent SLG* and show that the completeness and complexity of SLG are retained when extended to multiple threads of computation. Finally, we extend Local evaluation to Concurrent SLG, and show that the properties of Local evaluation continue to hold under concurrency.

Most Prologs now support some form of parallelism or multi-threading, while several Prologs also support some form of Tabled Logic Programming (TLP), leading to the practical question of how to support non-sequential TLP. This question is not new: [4] introduced the concept of *table parallelism*, and YAP [10] added tabling to a parallel engine. However, these parallel approaches are restricted to definite programs, and have not yet been extended to other TLP functionality including well-founded negation, subgoal or answer subsumption, constrained subgoals and answers, or incremental recomputation of tables that depend on dynamic information. Indeed, any sequential engine to support such functionality will be extremely complex, raising concern about the correctness of algorithms that add parallelism or concurrency to the mix.

This paper provides an operational semantics for a type of concurrent TLP that relies on a scheduling strategy called Local evaluation [5]. The model of concurrency adopted is one in which independent threads of computation execute separate subgoals while sharing completed tables. The main idea behind Local evaluation is that it fully evaluates a single mutually dependent set of tabled subgoals before performing operations (such as returning answers) to subgoals outside of that set. Experiments in several implementations have shown that Local evaluation utilizes space efficiently (see e.g. [5, 11]) and as a result it has been implemented for several Prologs, sometimes along with other scheduling strategies.

Another feature of Local evaluation is shown in an example in [5] in which tabling was used to compute the shortest path between two nodes. When Local evaluation was used the shortest path could be computed in a time proportional to the number of nodes in the graph, while if a non-local scheduling strategy was used the time was proportional to the number of *paths* in the graph – i.e. the time was exponential in the number of nodes. Comparing path lengths to compute a shortest path can be considered as an instance of *answer subsumption* in which answers are retained and propagated only if they are maximal over a partial order or are a function of answers so far produced.

Using SLG resolution [2] as a basis, this paper presents the following results about concurrent and Local evaluations.

- As analysis of Local evaluation in the literature has been mostly empirical, Local SLG evaluation is formally defined in Section 2 and shown ideally complete for queries to normal programs. Properties are derived about the structure of dependencies between subgoals in a Local evaluation, about the return of answers, and about the extent of non-completed subgoals in an evaluation.
- Section 3 presents an extension of SLG, called SLG_C , in which completed tables are shared among threads. SLG_C is complete for queries to normal programs, and its abstract complexity is the same as SLG.
- Concurrent *Local* SLG (Local SLG_C) is then defined in Section 3.1. It is shown that properties of Local SLG evaluations extend to the sub-evaluations performed by each concurrently executing thread, and a property is derived about the structure of dependencies between threads.
- Section 4 sketches the implementation of Local SLG_C in XSB, where the engine design is directly motivated by the results for subgoal and thread dependencies shown in this paper. In addition to having the properties of finite evaluations presented in this paper, XSB’s implementation of Local SLG_C has been extended to support tabled constraints, answer subsumption, tabled dynamic code, and space reclamation.

1 SLG Evaluation

Terminology and assumptions We assume the standard terminology of logic programming and an understanding of the well-founded semantics (see [15]). All programs discussed are normal, and defined over a countable language \mathcal{L} of predicates and function symbols. If L is a literal, the *underlying subgoal* of L is L if L is positive and S if $L = \text{not } S$. The *Herbrand base* H_P of a program P is the set of all ground atoms formed from \mathcal{L} . A *3-valued interpretation* I of a program P is a set of literals defined over H_P . For $A \in H_P$, if $A \in I$, A is true in I , and if $\text{not } A \in I$, A is false in I ; otherwise A and $\text{not } A$ are undefined in I . When I is an interpretation and A is an atom, $I|_A$ refers to

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

The Well-Founded Model of a program P is denoted as $WFM(P)$. In the following sections, we use the terms *goal*, *subgoal*, and *atom* interchangeably. Variant terms are considered to be identical.

This presentation of SLG follows [14], which reformulates the operations of [2] using the model of a forest of trees. The nodes in SLG trees are built from atoms and default literals along with a special type of literal called a delay literal.

Definition 1 (Delay Literals). A negative delay literal has the form *not* A , where A is a ground atom. A positive delay literal has the form $A_{Answer}^{Subgoal}$, where A is an atom whose truth value is based on that of some answer $Answer$ for the subgoal $Subgoal$. If θ is a substitution, then $(A_{Answer}^{Subgoal})\theta = (A\theta)_{Answer}^{Subgoal}$.

Positive delay literals contain information so that they may be simplified when a given answer to a given subgoal becomes unconditionally true or false.

Definition 2 (SLG Trees and Forest). An SLG forest consists of a set of SLG trees. Nodes of SLG trees have the forms:

$$Answer_Template \text{ :- } DelaySet|GoalList$$

or simply fail. In the first form, the *Answer_Template* is an atom, *DelaySet* is a set of delay literals and *GoalList* is a sequence of literals. The second form is called a failure node.

An SLG tree T is associated with a (possibly empty) marking sequence, which is a sequence of terms possibly preceded by the distinguished term *complete*. The first element of the marking sequence for T is denoted as $marking(T)$. For a term t , $setMark(T, t)$ prepends t to the marking sequence of T .

A node N is an answer when it is a leaf node for which *GoalList* is empty. If the *DelaySet* of an answer is empty it is termed an unconditional answer, otherwise, it is a conditional answer.

Definition 7 ensures that the root node of a given SLG tree has the form $S \text{ :- } |S$, where S is a subgoal. Within a forest each tree and subgoal are uniquely associated, so when T is an SLG tree in a forest \mathcal{F} whose root node is $S \text{ :- } |S$ it is sometimes convenient to use the terminology S is the root node for T ; T is the tree for S ; and S is in \mathcal{F} . If $marking(T) = complete$, we refer to both S and T as *completed*. Until Section 3, marking sequences will either be empty or will contain only the term *complete*. Literals in a *GoalList* are resolved by an arbitrary but fixed literal selection strategy. For simplicity, throughout this paper literals are always selected in a left-to-right order.

SLG operations transform one forest of trees into another. One of the operations, ANSWER RETURN is based on answer resolution, which is extended to take account of delay literals.

Definition 3 (Answer Resolution). Let N be a node $A \text{ :- } D|L_1, \dots, L_n$, where $n > 0$, and $Ans = A' \text{ :- } D'|$ an answer whose variables have been standardized

apart from N . N is SLG resolvable with Ans if $\exists i, 1 \leq i \leq n$, such that L_i and A' are unifiable with an mgu θ . The SLG resolvent of N and Ans on L_i is:

$$(A :- D|L_1, \dots, L_{i-1}, L_{i+1}, \dots, L_n)\theta$$

if D' is empty; otherwise the resolvent has the form:

$$(A :- D, L_{iA'}^{L_i}|L_1, \dots, L_{i-1}, L_{i+1}, \dots, L_n)\theta$$

The SLG COMPLETION operation marks a set of trees as complete when they can produce no more useful answers – a condition captured as follows.

Definition 4 (Completely Evaluated). A set \mathcal{S} of subgoals in a forest \mathcal{F} is completely evaluated if no $S \in \mathcal{S}$ is completed and if at least one of the following conditions holds for each $S \in \mathcal{S}$

1. The tree for S contains an answer $S :- |$; or
2. For each node N in the tree for S :
 - (a) The underlying subgoal of the selected literal of N is completed; or
 - (b) There are no applicable NEW SUBGOAL, PROGRAM CLAUSE RESOLUTION, ANSWER RETURN, NEGATION RETURN or DELAYING operations (Definition 7) for N .

In order to prevent \mathcal{S} from being repeatedly completed, the above definition explicitly prohibits \mathcal{S} from containing any subgoals from having already been completed. A COMPLETION operation based on Condition 1 is sometimes referred to as *Early Completion* [13] because it may complete a subgoal before completing other subgoals in a mutually dependent set.

According to Definition 3, if a conditional answer is resolved against the selected literal in the *GoalList* of a node, the information about the delayed literals in the answer need not be propagated — rather only the head of the answer is propagated as in Definition 3. In [2] it is shown that this form of propagation is necessary to ensure polynomial data complexity. However, in certain cases, this way of propagating delayed answers can lead to a set of *unsupported Atoms* — conditional answers that are false in the well founded model. Unsupported answers are handled by the ANSWER COMPLETION operation of Definition 7, and are defined as answers that are not supported:

Definition 5 (Supported Answer). Let \mathcal{F} be an SLG forest, S a subgoal in \mathcal{F} , and $Ans = A :- DelaySet|$ an answer in the tree for S . Then Ans is supported by S in \mathcal{F} if and only if:

1. S is not completely evaluated; or
2. there exists an answer node $A :- DelaySet'|$ of S such that for every positive delay literal $D_{Answer}^{Subgoal} \in DelaySet'$, $Answer$ is supported by $Subgoal$.

Unsupported answers reflect unfounded sets in the well-founded semantics. As an example, if in some forest the tree for a had the answer $a :- b_b^b|$, the tree

for b had the answer $b :- a_a^a$, and a and b were completely evaluated then both answers would be unsupported. The definition is recursive: condition 2 serves as a base case if there are no positive delay literals in $DelaySet'$ – i.e. if $DelaySet'$ is empty so that the answer is unconditional or $DelaySet'$ has only negative delay literals³. Condition 1 is not absolutely necessary, but is included so that the ANSWER COMPLETION operation will not be applied to non-completed subgoals.

SLG forests are related to interpretations in the following manner.

Definition 6. *Let \mathcal{F} be a forest. The interpretation induced by \mathcal{F} , $I_{\mathcal{F}}$, is the smallest set such that:*

- A (ground) atom $A \in I_{\mathcal{F}}$ iff A is in the ground instantiation of some unconditional answer $Ans :- |$ in \mathcal{F} .
- A (ground) literal $not A \in I_{\mathcal{F}}$ 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 \mathcal{F} if some tree in \mathcal{F} has an unconditional answer S . S is failed in \mathcal{F} if S is completed and the tree for S contains no answers. An atom S is successful (failed) in $I_{\mathcal{F}}$ if S' (not S') is in $I_{\mathcal{F}}$ for every S' in the ground instantiation of S . A negative delay literal $not D$ is successful (failed) in a forest \mathcal{F} if D is failed (successful) in \mathcal{F} . Similarly, a positive delay literal $D_{Answer}^{Subgoal}$ is successful in \mathcal{F} if Subgoal has an unconditional answer $Answer :- |$ and failed if Subgoal has no answer with head $Answer$.

Given these concepts, the SLG operations themselves can be stated.

Definition 7 (SLG Operations). *Given a forest \mathcal{F}_n of a SLG evaluation of program P \mathcal{F}_{n+1} may be produced by one of the following operations.*

1. NEW SUBGOAL: *Let \mathcal{F}_n contain a non-root node*

$$N = Ans :- DelaySet|G, GoalList$$

where G is the selected literal S or not S . Assume \mathcal{F}_n contains no tree with root subgoal S . Then add the tree $S :- |S$ to \mathcal{F}_n .

2. PROGRAM CLAUSE RESOLUTION: *Let \mathcal{F}_n contain a root node $N = S :- |S$ and C be a program clause $Head :- Body$ such that $Head$ unifies with S with mgu θ . Assume that in \mathcal{F}_n , N does not have a child $N_{child} = (S :- |Body)\theta$. Then add N_{child} as a child of N .*
3. ANSWER RETURN: *Let \mathcal{F}_n contain a non-root node*

$$N = Ans :- DelaySet|S, GoalList$$

whose selected literal S is positive. Let Ans be an answer node for S in \mathcal{F}_n and N_{child} be the SLG resolvent of N and Ans on S . Assume that in \mathcal{F}_n , N does not have a child N_{child} . Then add N_{child} as a child of N .

³ If an answer has a delay literal D_{Ans}^{Subg} and Ans is unconditional, an SLG SIMPLIFICATION operation is applicable to remove D_{Ans}^{Subg} from its delay set.

4. **NEGATION RETURN:** Let \mathcal{F}_n contain a leaf node

$$N = \text{Ans} \text{ :- } \text{DelaySet} \mid \text{not } S, \text{GoalList.}$$
 whose selected literal $\text{not } S$ is ground.
 - (a) **NEGATION SUCCESS:** If S is failed in \mathcal{F}_n , then create a child for N of the form: $\text{Ans} \text{ :- } \text{DelaySet} \mid \text{GoalList.}$
 - (b) **NEGATION FAILURE:** If S succeeds in \mathcal{F}_n , then create a child for N of the form fail.
5. **DELAYING:** Let \mathcal{F}_n contain a leaf node $N = \text{Ans} \text{ :- } \text{DelaySet} \mid \text{not } S, \text{GoalList,}$ such that S is ground, in \mathcal{F}_n , but S is neither successful nor failed in \mathcal{F}_n . Then create a child for N of the form $\text{Ans} \text{ :- } \text{DelaySet, not } S \mid \text{GoalList.}$
6. **SIMPLIFICATION:** Let \mathcal{F}_n contain a leaf node $N = \text{Ans} \text{ :- } \text{DelaySet} \mid,$ and let $L \in \text{DelaySet}$
 - (a) If L is failed in \mathcal{F} then create a child fail for N .
 - (b) If L is successful in \mathcal{F} , then create a child $\text{Ans} \text{ :- } \text{DelaySet}' \mid$ for N , where $\text{DelaySet}' = \text{DelaySet} - L$.
7. **COMPLETION:** Given a completely evaluated set \mathcal{S} of subgoals (Definition 4), for each $S \in \mathcal{S}$, $\text{setMark}(T, \text{complete})$, where T is the tree for S .
8. **ANSWER COMPLETION:** Given a set of unsupported answers \mathcal{UA} , create a failure node as a child for each answer $\text{Ans} \in \mathcal{UA}$.

SLG Evaluations An SLG evaluation consists of a (possibly transfinite) sequence of forests. The behavior of an evaluation at a limit ordinal is based on a least upper bound for a set of SLG trees. A rooted tree can be viewed as a partially ordered set in which each node N is represented as $\{N, P\}$ such that P is a tuple representing the path from N to the root of the tree. To make use of this, we assume a global total ordering on literals so that the elements in the DelaySet of a node can be uniformly ordered. Under this ordering a node of an SLG tree can be taken as a term to which the usual definition of variance applies. The marking sequence for a tree, which grows monotonically, can also be represented by a set.

Definition 8. Let T_1 and T_2 be SLG trees. $T_1 \sqsubseteq_T T_2$, if every element of T_1 is a subset of an element of T_2 . Let \mathcal{F}_1 and \mathcal{F}_2 be SLG forests. $\mathcal{F}_1 \sqsubseteq_F \mathcal{F}_2$ if $(\forall T_1 \in \mathcal{F}_1)(\exists T_2 \in \mathcal{F}_2)T_1 \sqsubseteq_T T_2$.

SLG operations either add new trees to the forest (as with the **NEW SUBGOAL** operation) or monotonically grow existing trees in the following sense:

Lemma 1. Let T_1 be an SLG tree, and T_2 be the extension of T_1 by any operation of Definition 7 except for **NEW SUBGOAL**. Then $T_1 \sqsubseteq_T T_2$.

Definition 9 (SLG Evaluation). Given a program P and goal G , an SLG evaluation \mathcal{E} is a sequence of SLG forests $\mathcal{F}_0, \mathcal{F}_1, \dots, \mathcal{F}_\beta$, such that:

- \mathcal{F}_0 is the forest containing a single tree $G \text{ :- } \mid G$

- For each successor ordinal, $n + 1 \leq \beta$, \mathcal{F}_{n+1} is obtained from \mathcal{F}_n by an application of an SLG operation from Definition 7.
- For each limit ordinal $\alpha \leq \beta$, \mathcal{F}_α consists of the least upper bound of $\mathcal{F}_i, i < \alpha$ with respect to the ordering \sqsubseteq_F .

If no operation is applicable to \mathcal{F}_β , \mathcal{F}_β is called a final forest of \mathcal{E} . If \mathcal{F}_β contains a leaf node with a non-ground selected negative literal, it is floundered.

The correctness is formulated as follows

Theorem 1 ([2, 14]). *Let \mathcal{E} be an SLG evaluation of a goal to a program P . Then \mathcal{E} has a final forest \mathcal{F} . Let A be an atom such that $A :- |A$ is the root of some tree in \mathcal{F} . Then if \mathcal{F} is non-floundered, $WFM(P)|_A = I_{\mathcal{F}}|_A$.*

2 Local SLG Evaluations

As noted above, a Local SLG evaluation fully evaluates each mutually dependent set of tabled subgoals before performing operations to subgoals outside of that set. We begin to formalize that notion by defining what it means for one subgoal to depend on another.

Definition 10 (Subgoal Dependency Graph). *Let \mathcal{F} be a forest, and let $S_1 :- |S_1$ be the root of a non-completed tree in \mathcal{F} . The subgoal S_1 directly depends on a subgoal S_2 iff S_2 is not completed in \mathcal{F} , and there is some node N in the tree for S_1 such that S_2 is the underlying subgoal of the selected literal of N or of a delay literal of N .*

The Subgoal Dependency Graph of \mathcal{F} $SDG(\mathcal{F}) = (V, E)$ of \mathcal{F} is a directed graph in which $(S_i, S_j) \in E$ iff subgoal S_i directly depends on subgoal S_j , and V is the underlying set of E . S_1 “depends on” S_2 in \mathcal{F} if there is a path from S_1 to S_2 in $SDG(\mathcal{F})$.

Since the SDG of a forest is a directed graph, it can be partitioned in disjoint sets of strongly connected components, or SCCs, where a node with no outgoing edges is considered to be in a *trivial* SCC. We refer to a given SCC by the set of its vertices (subgoals), and distinguish *independent* SCCs.

Definition 11 (Independent SCC). *A strongly connected component \mathcal{S} is independent if $\forall S \in \mathcal{S}$: if S depends on some S' , then $S' \in \mathcal{S}$.*

By Definition 11 it is straightforward that a trivial SCC is independent, and that each independent component is *maximal* — i.e. contained in no larger SCC. Local evaluation, then, performs operations on independent SCCs. Formally:

Definition 12 (Local SLG Evaluation). *Given a program P and goal G , a Local SLG evaluation \mathcal{E} is a sequence of SLG forests $\mathcal{F}_0, \mathcal{F}_1, \dots, \mathcal{F}_\beta$, such that:*

1. \mathcal{F}_0 is the forest containing a single tree $G :- |G$
2. For each successor ordinal, $n + 1 \leq \beta$, \mathcal{F}_{n+1} is obtained from \mathcal{F}_n by an application of an SLG operation from Definition 7 such that:

- (a) if \mathcal{F}_n contains an independent SCC
 - i. if a NEW SUBGOAL operation is applied to create a tree $S :- |S$ then S is the underlying subgoal of a selected literal in a tree whose root subgoal is in an independent SCC of $SDG(\mathcal{F}_n)$;
 - ii. a PROGRAM CLAUSE RESOLUTION, ANSWER RETURN, NEGATION RETURN or DELAYING operation is only applied to a node on a tree whose root subgoal is in an independent SCC of $SDG(\mathcal{F}_n)$;
 - iii. If the COMPLETION operation is applied to set \mathcal{S} then \mathcal{S} consists of all the subgoals in an independent SCC of $SDG(\mathcal{F}_n)$.
- (b) otherwise, any operation applicable to \mathcal{F}_n may produce \mathcal{F}_{n+1} .
- 3. For each limit ordinal $\alpha \leq \beta$, \mathcal{F}_α consists of the least upper bound of $\mathcal{F}_i, i < \alpha$ with respect to the ordering \sqsubseteq_F .

\mathcal{E} is delay avoiding if no DELAYING operation is performed in a forest if any other operation is applicable.

By Definition 12, a Local (SLG) evaluation works as a general SLG evaluation whenever an independent SCC does not exist in a forest.

Example 1. The program P_1 and a Local evaluation of the goal $p(X)$ to P_1 are shown in Figure 1, where the number before each node indicates the order of its creation. The clause $p(a) :- q(a)$ is first used for PROGRAM CLAUSE RESOLUTION against node 1, producing node 2. The independent SCC of \mathcal{F}_2 contains only the subgoal $q(a)$, so no other use of PROGRAM CLAUSE RESOLUTION will be applicable for node 1 until $p(a)$ is in an independent SCC. Accordingly, a NEW SUBGOAL operation creates a tree for $q(a)$, and the process continues, creating trees for $s(a)$, $t(a)$ and $u(a)$. The first PROGRAM CLAUSE RESOLUTION operation for $u(a)$, which creates node 10, does not in fact create a new independent SCC, so that operations continue to be applicable to the tree for $u(a)$. Although there is also a PROGRAM CLAUSE RESOLUTION operation applicable, the selected literal of node 10 is delayed in this evaluation, producing the conditional answer in node 10a. However before the answer can be returned outside of the SCC $\{u(a)\}$, $u(a)$ must be completed. Further program clause resolution produces node 11, and an unconditional answer for $u(a)$, node 11. This unconditional answer is then used in a SIMPLIFICATION operation to produce the failure node, 10b. At this point the SCC $\{u(a)\}$ is completely evaluated and can be completed. Afterward, the answer $u(a) :- |$ can be used outside of the SCC, to produce node 12. Evaluation continues until node 14 is produced, at which time the independent SCC becomes $\{q(a), s(a), t(a), v(a)\}$. Operations are now applicable to any of the trees in this SCC, and evaluation continues until all subgoals in this SCC have been completely evaluated, at node 19.

This evaluation illustrates several points. First, the subgoals $q(a), s(a), t(a), v(a)$ and $u(a)$ are all completed before the clause $p(X) :- r(X)$ can be resolved against node 1. Second, Local evaluation prevents the conditional answer in node 10a from being resolved against the selected literal of node 8. Third, although Local evaluations can be delay avoiding, this evaluation is not, as DELAYING is used to produce node 10a when there was

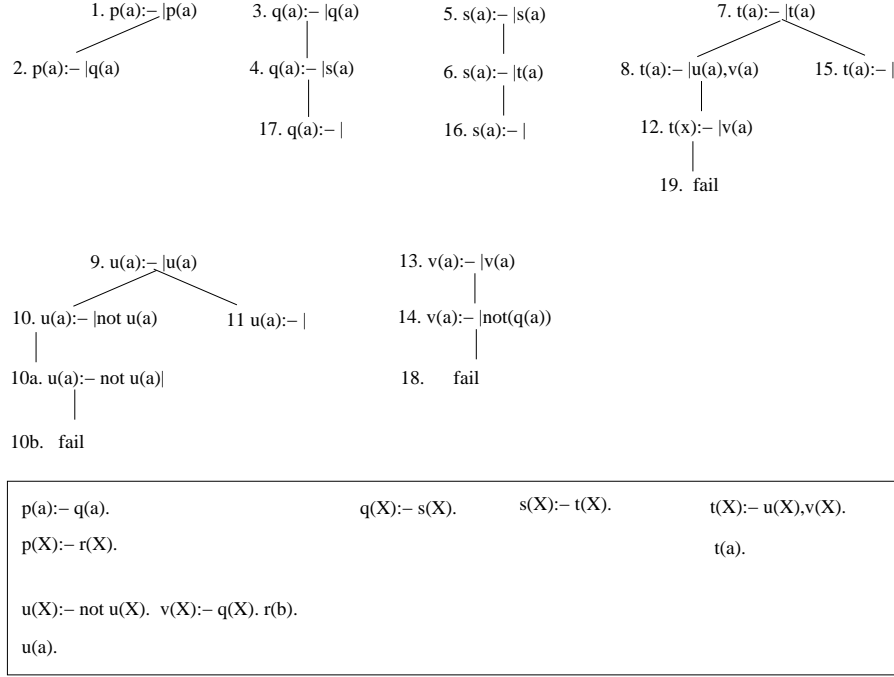


Fig. 1. A Program P_1 and a Local Evaluation for P_1

a non-DELAYING operation applicable. Finally, note that Condition 2(a)iii of Definition 12 prevents the use of Early Completion. If Early Completion could have been used, $t(a)$ could have been completed after the node 15 was produced, but doing so would have disconnected the SDG into two connected components: $\{p(a), q(a), s(a)\}$ and $\{v(a)\}$.

Theorem 2 (Completeness of Local Evaluation). *Let P be a program and G a goal. Then there exists an SLG evaluation \mathcal{E} of G against P with final forest \mathcal{F} if and only if there exists a local SLG evaluation \mathcal{E}^L of G against P with final forest \mathcal{F}^L such that $I_{\mathcal{F}}|_G = I_{\mathcal{F}^L}|_G$.*

Local evaluation is ideally complete for the well-founded semantics. However, its importance arises from its efficiency for certain classes of programs, along with properties that can be used to ensure the correctness of implementations.

Theorem 3. *Let \mathcal{E}^L be a finite Local SLG evaluation. For each \mathcal{F} in \mathcal{E}^L SDG(\mathcal{F}) has one and only one independent SCC.*

In any forest \mathcal{F}^L of a finite Local evaluation \mathcal{E}^L , no operation may be applied to a tree outside the independent SCC of \mathcal{F}^L (Definition 12). From Theorem 3 \mathcal{F}^L contains a single independent SCC \mathcal{S} . Thus any edge into a subgoal in \mathcal{S} must be added at the creation, via a NEW SUBGOAL operation, of the oldest

subgoal in \mathcal{S} – i.e. the subgoal whose tree was created the earliest in \mathcal{E}^L . This argument shows the following corollary.

Corollary 1. *Let \mathcal{E}^L be a finite Local SLG evaluation. For each \mathcal{F} in \mathcal{E}^L there is at most one incoming edge for each SCC in $SDG(\mathcal{F})$.*

The following corollary captures the notion that in a Local evaluation, a subgoal may only return answers out of its SCC once its SCC has been completed.

Corollary 2. *In any forest \mathcal{F} of a Local SLG evaluation, if an answer A is used in an ANSWER RETURN operation to a node in a tree with root subgoal S , then the tree for A has been completed, or is in the same SCC as S in $SDG(\mathcal{F})$.*

Corollary 2 has practical importance for answer subsumption since it implies that no answer A will be returned out of an SCC if the model entails an answer that is preferred to A . In addition, it is easy to see that if Local evaluation were extended to ensure that all appropriate SIMPLIFICATION and ANSWER COMPLETION operations are performed for an independent SCC just after it has been completed, the following statement would also hold. If a forest in Local evaluation contains a conditional answer $A = S :- D$, and S is true or false A will never be propagated outside of the SCC for S if S is true or false in $WFM(P)$. This strategy reduces the overall number of SIMPLIFICATION and ANSWER COMPLETION operations and has been adopted by the XSB engine when computing non-stratified programs [12].

2.1 Early Completion and Local Evaluations

According to Definition 4 a subgoal S can be determined to be completely evaluated if (condition 1, Early Completion) the tree for S contains an answer $S :- |$, or (condition 2) if all applicable SLG operations have been performed for S and for subgoals upon which S depends. Note that condition 2(a)iii of Definition 12 of Local evaluation precludes the use of Early Completion. The following theorem shows that if it is delay-avoiding, each state of a Local evaluation has a “minimum” number of non-completed subgoals compared to non-Local evaluations that do not use Early Completion. The theorem is a formal statement of the space efficiency of Local evaluation, as tabled evaluations require more resources (e.g. stack space) for non-completed subgoals than for completed subgoals.

Theorem 4. *Let \mathcal{E}^L be a finite delay-avoiding Local evaluation of a goal G to a program P , and let \mathcal{E} be an SLG evaluation of G to a P such that \mathcal{E} does not use Early Completion. Then for any forest \mathcal{F}^L in \mathcal{E}^L , there exists a forest \mathcal{F} in \mathcal{E} such that $SDG(\mathcal{F}^L)$ is a subgraph of $SDG(\mathcal{F})$.*

Early Completion is necessary to evaluate fixed-order dynamically stratified programs without delaying [13]. The XSB engine for Local evaluation adopts a pragmatic approach of using Early Completion for ground subgoals, but maintaining a view of the SDG that treats early-completed subgoals as if they were in the independent SCC.

3 SLG_C

In this section SLG is extended to a model of concurrency in which threads share only *completed* tables. This model is somewhat restrictive as *thread compatibility restrictions* prevent a thread from consuming answers from a table owned by another thread. This disallows consume-producer models of concurrency and implies that different threads may not collaborate to evaluate subgoals in an SCC. However, within a Local evaluation the compatibility restrictions may not be binding, since Local evaluations prevent consumer-producer models by their nature, and since the scope of an SCC in a Local evaluation is relatively small.

Formally, this is accomplished by marking every non-completed tree in a given forest with a *thread identifier* (cf. Definition 2). As terminology, if N is a node in a tree T , $\text{marking}(N)$ denotes $\text{marking}(T)$, and if S is a subgoal $\text{marking}(S)$ denotes $\text{marking}(T)$, where T is the tree for S .

Definition 13 (Thread). *A thread identifier is an element of a set of terms that does not include the term complete. Given an SLG forest \mathcal{F} in an evaluation \mathcal{E} , a thread state is the maximal set \mathcal{T} of trees in \mathcal{F} such that for all $T \in \mathcal{T}$ $\text{marking}(T) = t$ where t is a thread identifier. A thread in \mathcal{E} is the sequence of thread states for a given marking. A thread is active in \mathcal{F} if its thread state in \mathcal{F} is non-empty.*

Let S be a subgoal, T the tree for S , and N a node in a forest. N is thread compatible with S if $\text{marking}(T) = \text{complete}$ or $\text{marking}(T) = \text{marking}(N)$.

SLG_C uses SLG forests and other notions from Definitions 1- 6, but differs in that certain SLG_C operations may create or change thread markings, and thread markings may restrict the applicability of operations. Definition 14 presents a new operation called USURPATION, along with those operations that differ from Definition 7 where the difference in each altered operation is underlined

Definition 14 (SLG_C Operations). *Given an SLG forest \mathcal{F}_n \mathcal{F}_{n+1} may be produced by one of the following operations.*

1. NEW SUBGOAL: *Let \mathcal{F}_n contain a non-root node*

$$N = \text{Ans} \text{ :- } \text{DelaySet} \mid G, \text{GoalList}$$

where G is the selected literal S or not S . Assume \mathcal{F}_n contains no tree with root subgoal S . Then add the tree $T = S \text{ :- } \mid S$ to \mathcal{F}_n , and $\text{setMark}(T, \text{marking}(N))$.

2. ANSWER RETURN: *Let \mathcal{F}_n contain a non-root node N whose selected literal S is positive. Let Ans be an answer node for S in \mathcal{F}_n such that N is thread compatible with S and let N_{child} be the SLG resolvent of N and Ans on S . Assume that in \mathcal{F}_n , N does not have a child N_{child} . Then add N_{child} as a child of N .*
3. NEGATION RETURN: *Let \mathcal{F}_n contain a leaf node*

$$N = \text{Ans} \text{ :- } \text{DelaySet} \mid \text{not } S, \text{GoalList}.$$

whose selected literal not S is ground where N is thread compatible with S .

- (a) **NEGATION SUCCESS:** If S is failed in \mathcal{F}_n , then create a child for N of the form: $Ans :- DelaySet|GoalList$.
- (b) **NEGATION FAILURE:** If S succeeds in \mathcal{F}_n , then create a child for N of the form $fail$.
- 4. **COMPLETION:** Given a completely evaluated set \mathcal{S} of subgoals such that for all $S \in \mathcal{S}$, $marking(S) = t$, then for each $S \in \mathcal{S}$, $setMark(T, complete)$, where T is the tree for S .
- 5. **USURPATION:** Let \mathcal{S} be a set of subgoals in deadlock (Definition 15), $S_U \in \mathcal{S}$, and T_U the tree for S_U . For each $S \in \mathcal{S}$, $setMark(T, marking(T_U))$.

The thread compatibility restrictions can mean that an SLG operation is applicable in a given forest, but that the corresponding SLG_C operation is not. The USURPATION operation is designed to address cases where SLG_C operations might get stuck – which are formalized as situations of deadlock.

Definition 15 (Deadlock). A set \mathcal{S} of subgoals in a forest \mathcal{F} is in deadlock if:

1. For each $S \in \mathcal{S}$ there are no applicable **NEW SUBGOAL**, **PROGRAM CLAUSE RESOLUTION**, **ANSWER RETURN**, **NEGATION RETURN** or **DELAYING operations** of Definition 14.
2. There exists no \mathcal{S}' such that $\mathcal{S} \subseteq \mathcal{S}'$ and \mathcal{S}' is completely evaluated in \mathcal{F} .

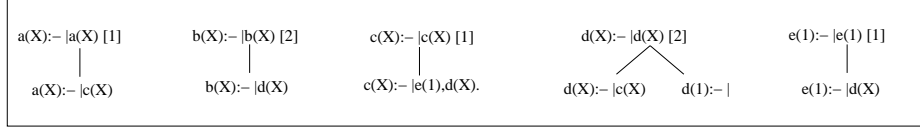
Example 2. SLG_C evaluations may use any scheduling strategy, and are not restricted to Local evaluations. They also begin with a *set* of goals rather than with a single goal. Figure 2 illustrates a simple, non-Local, SLG_C evaluation of the goal $\{a(X), b(X)\}$ to the program P_2 , where $a(X)$ is initially marked with thread identifier 1 and $b(X)$ with thread identifier 2. Through **NEW SUBGOAL** operations, trees for $c(X)$ and $e(X)$ are created and associated with thread identifier 1, while $d(X)$ is created and associated with thread identifier 2. Evaluation continues until there is a deadlock, as shown in Figure 2b. Note in Figure 2b, that while there is an answer that could be returned to the node $e(1) :- !d(X)$ in a non-Local evaluation, the node is associated with thread identifier 1, while the answer is associated with thread identifier 2 so that the return is prohibited by the thread compatibility restrictions. USURPATION is the only operation applicable to this forest; assume that thread identifier 1 performs the USURPATION, marking trees for $c(X)$, $d(X)$, and $e(X)$ with identifier 1. Afterward, an answer for $e(1)$ is derived, leading to Figure 2c. At this stage, $e(1)$ is completely evaluated and can be early completed. Further **ANSWER RETURN** operations lead to Figure 2d (in which the completed tree for $e(1)$ is not shown.) All of the subgoals in thread identifier 1 have been completely evaluated, but the subgoal $b(X)$ in thread identifier 2 cannot be completely evaluated until $d(X)$ is completely evaluated, and an answer returned to the node $b(X) :- !d(X)$.

The definition of a SLG_C evaluation is nearly the same as for SLG (Definition 9), but is initialized so that each atomic query in the set of goals it is presented with is marked with a different thread identifier.

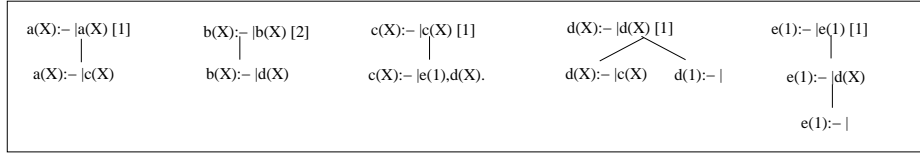
$$\mathbf{a(X):- c(X). \quad b(X):- d(X). \quad c(X):- e(1),d(X). \quad d(X):- c(X). \quad d(1).}$$

$$\mathbf{e(1):- d(X)}$$

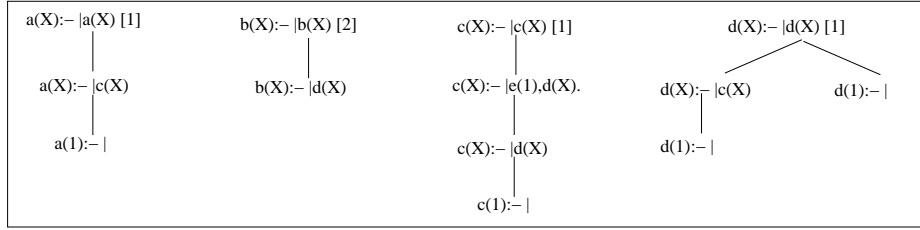
(a) The Program P_2



(b) State 1: Deadlock



(c) State β : $e(1)$ Early Completable



(d) State γ : Complete Evaluation for Thread Identifier 1

Fig. 2. A non-Local SLG_C Evaluation of P_2

Definition 16 (SLG_C Evaluation). Given a program P , a set \mathcal{T} of thread identifiers, and a finite non-empty set \mathcal{G} of goals, a SLG_C evaluation \mathcal{E} is a sequence of SLG forests $\mathcal{F}_0, \mathcal{F}_1, \dots, \mathcal{F}_\beta$, such that:

- \mathcal{F}_0 is a set-minimal forest containing the trees $T_i = G_i :- |G_i$, for each $G_i \in \mathcal{G}$, where for each T_i there is a $t_i \in \mathcal{T}$ such that $\text{marking}(T_i) = t_i$, and $t_i \neq t_j$ if $i \neq j$.
- For each successor ordinal, $n + 1 \leq \beta$, \mathcal{F}_{n+1} is obtained from \mathcal{F}_n by an application of a SLG_C operation from Definition 14
- For each limit ordinal $\alpha \leq \beta$, \mathcal{F}_α is the least upper bound of $\mathcal{F}_i, i < \alpha$ with respect to the ordering \sqsubseteq_F .

SLG_C forests are based on Definition 2, so the definition of an interpretation induced by a forest is identical in both frameworks, leading to: the following theorem.

Theorem 5 (Correctness of SLG_C). Let P be a program and \mathcal{G} a finite non-empty set of goals. Then a SLG_C evaluation of \mathcal{G} against P exists with final state

$\widehat{\mathcal{F}}$, iff for every $G_i \in \mathcal{G}$ there exists an SLG evaluation of G_i against P with final state \mathcal{F}^i and $I_{\widehat{\mathcal{F}}} = (\bigcup I_{\mathcal{F}^i})$.

The completeness portion of the theorem follows from a demonstration that for any SLG operation on a forest, an equivalent SLG_C operation is applicable after zero or more USURPATION operations. The following theorem bounds the number of USURPATION operations in a finite evaluation, which implies that the abstract complexity of SLG_C is the same as that of SLG.

Theorem 6 (Complexity of USURPATION). *Let \mathcal{E} be a finite SLG_C evaluation with final forest \mathcal{F} , and $S_{\mathcal{F}}$ the set of all subgoals in \mathcal{F} . Then if \mathcal{E} does not make use of Early Completion, there are at most $|S_{\mathcal{F}}|$ USURPATION operations performed, and at most $2|S_{\mathcal{F}}|$ USURPATION operations performed otherwise.*

3.1 Concurrent Local Evaluations

In SLG_C the Subgoal Dependency Graph (Definition 10) can be partitioned into disjoint sub-graphs for each thread state of a forest.

Definition 17 (Thread Subgoal Dependency Graph). *For each thread state t in a forest \mathcal{F} , the Thread Subgoal Dependency Graph of t ($Thread_SDG(\mathcal{F}, t)$) consists of the sub-graph of $SDG(\mathcal{F})$ determined by subgoals in \mathcal{F} whose marking is t .*

Concurrent Local SLG evaluation is based on independent SCCs within Thread SDGs, rather than within a global SDG.

Definition 18 (Concurrent Local SLG Evaluation). *Given a program P , a set \mathcal{T} of thread identifiers, and a finite non-empty set G of goals, a Concurrent Local SLG evaluation \mathcal{E} is a sequence of forests $\mathcal{F}_0, \mathcal{F}_1, \dots, \mathcal{F}_\beta$, such that:*

1. \mathcal{F}_0 is a set-minimal forest containing the trees $T_i = G_i :- |G_i$, for each $G_i \in \mathcal{G}$, where for each T_i there is a $t_i \in \mathcal{T}$ such that $marking(T_i) = t_i$, and $t_i \neq t_j$ if $i \neq j$.
2. For each successor ordinal, $n + 1 \leq \beta$, \mathcal{F}_{n+1} is obtained from \mathcal{F}_n by an application of an operation from Definition 14 such that:
 - (a) if \mathcal{F}_n contains an independent SCC
 - i. if a NEW SUBGOAL is applied to create a tree $T = S :- |S$ then S is in an independent SCC of $Thread_SDG(\mathcal{F}_n, marking(T))$;
 - ii. if a PROGRAM CLAUSE RESOLUTION, ANSWER RETURN, NEGATION RETURN or DELAYING operation is applied in \mathcal{F}_n then it is applied to a tree T whose root subgoal is in an independent SCC of $Thread_SDG(\mathcal{F}_n, marking(T))$.
 - iii. If the COMPLETION operation is applied to set S in \mathcal{F}_n , then S consists of all the subgoals in an independent SCC of $Thread_SDG(\mathcal{F}_n, t)$ for some thread identifier t .
 - (b) otherwise, any operation applicable to \mathcal{F}_n may produce \mathcal{F}_{n+1} .

3. For each limit ordinal $\alpha \leq \beta$, \mathcal{F}_α is the least upper bound \mathcal{F}_U of $\mathcal{F}_i, i < \alpha$, with respect to the ordering \sqsubseteq_F .

Theorem 7 (Correctness of Concurrent Local SLG). *Let P be a program and \mathcal{G} a finite non-empty set of goals. Then a Concurrent Local SLG evaluation of \mathcal{G} against P exists with final state $\widehat{\mathcal{F}}$, iff every $G_i \in \mathcal{G}$ there exists an SLG evaluation of G_i against P with final state \mathcal{F}^i and $I_{\widehat{\mathcal{F}}} = (\bigcup I_{\mathcal{F}^i})$.*

The following theorem implies that each thread of a Concurrent Local SLG evaluation has the properties of Section 2.

Theorem 8. *Let \mathcal{F} be a forest in a finite Concurrent Local SLG evaluation. Then for each active thread t in \mathcal{F} , $\text{Thread_SDG}(\mathcal{F}, t)$ has one and only one independent SCC.*

The Thread Dependency Graph can be seen as a homomorphism of the SDG of a given SLG_C forest.

Definition 19 (Thread Dependency Graph). *Let t_1 and t_2 be two active threads in a SLG forest \mathcal{F} . t_1 directly depends on t_2 if there exist a subgoal in t_1 that directly depends on a subgoal in t_2 (according to Definition 10). The Thread Dependency Graph $\text{TDG}(\mathcal{F}) = (V, E)$ of \mathcal{F} is a directed graph where V is the set of active threads in \mathcal{F} and $(t_i, t_j) \in E$ iff t_i directly depends on t_j .*

Based on the thread dependency graph, the following theorem shows that any thread depends on at most one other thread.

Theorem 9. *Let \mathcal{F} be a forest in a Concurrent Local SLG evaluation. Then for each node in $\text{TDG}(\mathcal{F})$ there is at most one outgoing edge.*

As a practical matter, this theorem indicates that each thread of computation will wait on the results from at most one other thread. so that the thread communication and dependency detection required to implement the USURPATION operation will be relatively simple.

4 SLG_C

We discuss at a summary level the changes made to XSB's SLG-WAM in order to implement Local SLG_C . Our discussion omits numerous optimizations required for efficiency. In particular, due to space restrictions we do not discuss the propagation of subgoal dependencies between threads, or the handling of subgoals that have been usurped multiple times (see [9] for details). We first describe Local SLG_C for definite programs before considering negation.

Since XSB's SLG-WAM implements Local evaluation, it is evident from Section 3 that the main addition is the USURPATION operation, which mainly affects the SLG_C -WAM tabletry instruction. This instruction occurs at the entry point

of a tabled predicate when a tabled subgoal *Subg* is called ⁴. In the sequential SLG-WAM `tabletry` is essentially responsible for determining whether a NEW SUBGOAL operation is required. The instruction compares the WAM argument registers in order to determine whether the current subgoal is in the table. If the subgoal is not in the table, a NEW SUBGOAL operation is effectively performed: the subgoal will have been copied to the table during the check and a *generator* choice point is created to backtrack through program clauses, to check whether a SCC has been fully evaluated, and to schedule ANSWER RETURN operations if not. On the other hand, if *Subg* is in the table, `tabletry` creates a *consumer* choice point to backtrack through any answers to *Subg* in the table and thereby perform ANSWER RETURN operations.

Instruction `tabletry`

```

/* Subg is in argument registers;  $T_{current}$  is current thread */
Perform the subgoal_check_insert(Subg) operation in the table for this predicate
If Subg is not new and is marked by another thread
    lock global TDG mutex
    If deadlock( $T_{current}, Subg.ThreadMark$ )
        /* all other threads in the independent SCC are suspended at deadlock */
        usurp( $T_{current}, Subg, Subg.ThreadMark$ )
    Else unlock TDG mutex; suspend the calling thread until Subg completes
Proceed as in the sequential case; if Subg was usurped, treat it as a new subgoal

```

```

deadlock( $T_{current}, depends\_thread$ )
while( depends_thread  $\neq$  NULL )
    if( depends_thread =  $T_{current}$  ) return true;
    else depends_thread  $\leftarrow$  subgoal_thread.suspended_on_thread);
return false;

```

```

usurp( $T_{current}, dep\_SF, first\_usurped$ )
Traverse SCC to reset suspended_on_thread dependencies
Unlock global mutex that protected TDG
Traverse SCC to
    Reset stacks of each (suspended) usurped thread
    Propagate the proper subgoal dependency to each usurped thread

```

Fig. 3. Summary of Concurrent Local SLG implementation in the SLG-WAM.

Extensions to `tabletry` for Local SLG_C are summarized in Figure 3. If *Subg* is new, it is copied into the table as in the sequential case, but a thread identifier is associated with the *ThreadMark* cell of the *subgoal frame* of *Subg*, where the subgoal frame is a structure in the SLG-WAM containing information about a tabled subgoal. The essential difference from the sequential case occurs when *Subg* is not new and is currently marked by another thread (and therefore not

⁴ If a tabled predicate has only one clause, a specialization called `tabletrysingle` is used.

marked as completed). In this case deadlock detection is performed: if a deadlock is not found the calling thread $T_{current}$ suspends as it does not have any applicable Local SLG_C operations; otherwise $T_{current}$ performs a USURPATION operation. In addition to changes in `tabletry`, a change is made to the SLG-WAM completion instruction so that any thread suspended on $Subg$ is awakened when $Subg$ is completed (a condition variable based on the predicate symbol of $Subg$ is used for this awakening).

The design of deadlock detection in the SLG-WAM relies on Theorem 9, which states that each thread may be suspended on at most one other thread. The SLG-WAM adds a `suspended_on_thread` field to the context of each thread to denote any thread dependency. As shown in Figure 3, when a thread $T_{current}$ performs deadlock detection, it starts by checking whether the thread marking $Subg$ is suspended using this `suspended_on_thread` field: if the thread is not suspended, $T_{current}$ may suspend without fear of deadlock and it will be awakened when $Subg$ is completed. If the marker of $Subg$ is suspended, the deadlock detection code follows the `suspended_on_thread` field. By a simple corollary to Theorem 9, any loop in the TDG must be a simple cycle so that deadlock detection is a simple while loop that exits in one of two cases. If $T_{current}$ is found in the `suspended_on_thread` field for one of the traversed threads, then $T_{current}$ depends (transitively) on itself and deadlock occurs; otherwise if the `suspended_on_thread` of a traversed thread is null, $T_{current}$ transitively depends on a subgoal that is actively being computed.

The fact that the thread dependencies for deadlocked threads form a simple cycle also underlies the control flow of the `usurp()` function which consists of two traversals of the deadlocked TDG cycle, denoted SCC_{dl} . In the first traversal, the usurping thread updates the TDG, setting the `suspended_on_thread` field of each usurped thread to its own id. Adjusting the TDG must be performed under global mutual exclusion: otherwise two usurping threads concurrently adjusting the TDG might produce an incoherent TDG. In the second traversal, which is not under mutual exclusion, the execution stacks in each usurped thread $T_{usurped}$ are examined and manipulated – an operation that is safe since $T_{usurped}$ has suspended on a subgoal due to thread compatibility restrictions. The manipulation ensures that $T_{usurped}$ will no longer be generating answers for usurped subgoals that it has marked, but rather will be set to consume answers. In fact, Corollary 1 together with Theorem 8 imply that the SDG for $T_{usurped}$ will depend on a *single* usurped subgoal $S_{T_{usurped}}$ – the first subgoal in SCC_{dl} that $T_{usurped}$ encountered, considerably simplifying the stack manipulation needed. In determining $S_{T_{usurped}}$, it is not sufficient to consider only dependencies contained in $T_{usurped}$, but subgoal dependencies across usurped threads must also be considered. Accordingly, `usurp()` also propagates cross-thread dependencies and uses these dependencies when resetting the stacks of $T_{usurped}$. This resetting of stacks will cause $T_{usurped}$ to call $S_{T_{usurped}}$ again from scratch upon awakening. Thus in this implementation of Local SLG_C , any partial computations for the usurped subgoals are lost, and will be recomputed by the usurping thread, a point further discussed below.

Extensions for Negation and Answer Subsumption As suggested by the changed operations in Definition 14, the SLG-WAM requires few modifications beyond those presented in order to implement the well-founded semantics. Consider first the case of stratified programs. In the sequential SLG-WAM, if the underlying (tabled) subgoal *Subg* of a selected negative literal is not new and not complete, the computation path “suspends” and resumes only when *Subg* has been completed. These operations are essentially the same as the interactions between threads so far described. In the case of non-stratified negation the first new operation to consider is the SLG DELAYING operation. If *Subg* is involved in a loop through negation, the resumption mechanism is the same except that a bit in the subgoal frame of *Subg* is set to indicate that *Subg* was delayed rather than completed. Several cycles of delaying may be needed before *Subg* is finally completed, but these may all be handled using the thread suspension and usurpation mechanisms described. When *Subg* is completed, any SIMPLIFICATION operations for SCC_{dl} are also performed before awakening any threads suspended on *subg*, so that SIMPLIFICATION is not affected by the concurrency mechanisms. Beyond negation, Local evaluation is highly useful for applications of answer subsumption. Implementations of answer subsumption are performed as extensions to the SLG-WAM `new_answer` operation which is unaffected by the changes for Local SLG_C .

The described implementation has to recompute answers for usurped subgoals. Theorem 6 states that the maximal number of USURPATION operations in a SLG_C evaluation is linear in $atoms(P)$, the number of atoms in a program P . In [9] it is shown that USURPATION operations affect only constant-time operations so that even if answers for usurped subgoals are recomputed, the complexity of the well-founded semantics is unaffected.

5 Discussion

Local SLG_C has several strengths. By Theorem 7 several threads can cooperate to correctly compute the well-founded semantics, and by Theorem 6 the abstract complexity is the same as a sequential SLG evaluation. By Theorem 8 each thread in a Local SLG_C evaluation will have a single independent SCC, and so each thread will have properties of a Local evaluation, including the space efficiency property of Theorem 4. By Corollary 2 (and Theorem 8) each thread will only return answers from completed tables, a useful property for computing the well-founded semantics and answer subsumption. As noted in Section 4, the implementation of Local SLG_C directly relies on Theorem 9 and Corollary 1. As a result of the theorie-oriented design, the implementation of Local SLG_C , although delicate, mainly requires about 300 lines of code to be added to the `tabletry` instruction so that Local SLG_C should be relatively easy to implement in other tabling engines that implement Local evaluation.

There are also limitations to Local SLG_C . Local evaluation itself prevents certain types of parallelism – for instance one thread may not consume answers from a table that are concurrently produced by another thread, if the tables are in different SCCs. Beyond this, a Local SLG_C evaluation may have a number

of threads suspended on incomplete or usurped subgoals, although Theorem 6 puts a limit on the number of USURPATION operations. These limitations are best addressed by analysis of performance on different programs. Results (cf. [7,8]) indicate that the changes in Section 4 causes no perceptible overhead when the search space of each thread is disjoint – i.e. in the “best” extremal case where thread suspensions and deadlocks are not necessary. Performance results on a bad extremal case, where multiple threads concurrently evaluate right recursion on highly connected random graphs indicate that the number of deadlocks are relatively small, apparently due to the fact that SCCs in Local evaluation are small and the subgoals in them are quickly completed. In addition for these graphs, the elapsed time for the Local SLG_C is never worse than that for a Local single-threaded evaluation. In short, Local SLG_C is best suited for multi-threaded evaluations that benefit from Local evaluation and can provide speedups on problems that can be subdivided relatively easily; it is not intended as an approach to general table parallelism.

These strengths and limitations distinguish (Local) SLG_C from previous work, which we briefly discuss (see [7] for more details). [6] presents an approach to distributed tabling in which the SDG (Definition 10) is distributed among threads, the dependencies partially represented by numerical encodings associated with subgoals, and a message-counting algorithm used for termination detection. Maintaining the distributed SDG leads to an approach that is cubic in the number of messages. SLG_C differs from [6] in being a more minimal extension of SLG requiring only the addition of markings and USURPATION, and in retaining the complexity of SLG. Another distributed tabling method, [3] (later extended in [1]) avoids the cubic message complexity by using a centralized table manager to maintain dependency and other information and a credit-recovery algorithm to detect completion of the SCCs. SLG_C differs from [3] in not requiring an explicit table manager and in using the “optimistic” USURPATION operation to control concurrency, as well as in being a formalism sufficient for proving completeness and other properties. [10] presents algorithms for adding tabling to an or-parallel engine and implements these algorithms in YAP, with impressive results for definite programs. As mentioned above, unlike [10] Local SLG_C does not address general table parallelism, although it addresses normal programs and is based on a formalization which permits a concise implementation. Perhaps the closest work is [4] which allows threads to share answers when tables are not completed: Concurrent SLG differs from this work in using a simpler method of concurrency control, as well as in modelling normal rather than definite programs.

References

1. M. Alves, C. Damásio, W. Nejdl, and D. Olmedilla. A distributed tabling algorithm for rule based policy systems. In *Seventh IEEE International Workshop on Policies for Distributed Systems and Networks*, 2006.
2. W. Chen and D. S. Warren. Tabled Evaluation with Delaying for General Logic Programs. *Journal of the ACM*, 43(1):20–74, January 1996.

3. C. Damásio. A distributed tabling system. In *Proceedings of the 2nd Workshop on Tabulation in Parsing and Deduction, TAPD'2000*, pages 65–75, 2000.
4. J. Freire, R. Hu, T. Swift, and D. S. Warren. Parallelizing tabled evaluation. In *7th International PLILP Symposium*, pages 115–132. Springer-Verlag, 1995.
5. J. Freire, T. Swift, and D. S. Warren. Beyond depth-first: Improving tabled logic programs through alternative scheduling strategies. *JFLP*, 1998(3), 1998.
6. R. Hu. *Distributed Tabled Evaluation*. PhD thesis, SUNY at Stony Brook, 1997.
7. R. Marques. *Concurrent Tabling: Algorithms and Implementation*. PhD thesis, Universidade Nova de Lisboa, 2007.
8. R. Marques, T. Swift, and J. Cunha. Extending tabled logic programming with multi-threading: A systems perspective. Available at <http://www.cs.sunysb.edu/~tswift>, 2008.
9. R. Marques, T. Swift, and J. Cunha. A simple and efficient implementation of concurrent local tabling. Available at <http://www.cs.sunysb.edu/~tswift>, 2008.
10. R. Rocha, F. Silva, and V. S. Costa. On applying or-parallelism and tabling to logic programs. *Theory and Practice of Logic Programming*, 4(6), 2004.
11. R. Rocha, F. Silva, and V. Santos Costa. Dynamic mixed-strategy evaluation of tabled logic programs. In *ICLP*, page 250264, 2005.
12. K. Sagonas, T. Swift, and D. S. Warren. An abstract machine for efficiently computing queries to well-founded models. *JLP*, 45(1-3):1–41, 2000.
13. K. Sagonas, T. Swift, and D. S. Warren. The limits of fixed-order computation. *Theoretical Computer Science*, 254(1-2):465–499, 2000.
14. T. Swift. A new formulation of tabled resolution with delay. In *Recent Advances in Artificial Intelligence*, volume 1695 of *LNAI*, pages 163–177. Springer-Verlag, 1999.
15. A. van Gelder, K.A. Ross, and J.S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

A Proofs

Theorem 2 [Completeness of Local Evaluation] *Let P be a program and G a goal. Then there exists an SLG evaluation \mathcal{E} of G against P with final forest \mathcal{F} if and only if there exists a local SLG evaluation \mathcal{E}^L of G against P with final forest \mathcal{F}^L such that $I_{\mathcal{F}}|_G = I_{\mathcal{F}^L}|_G$.*

Proof. (Sketch)

\Leftarrow Trivial, as any operation on a forest with successor ordinal in \mathcal{E}^L is applicable for \mathcal{E} .

\Rightarrow We begin by proving a special case, where P has the *bounded term depth* property – i.e. an undecidable property that an evaluation will produce subgoals and answers whose depth is finite. In such a case, it is straightforward to show that \mathcal{F} has a finite number of finite trees. It is then straightforward to see that \mathcal{E}^L will completely traverse such a finite structure, since in a bounded term depth program any forest \mathcal{F}^L in \mathcal{E}^L , $SDG(\mathcal{F}^L)$ will be finite, and it will require a finite number of operations to complete the independent SCC of $SDG(\mathcal{F}^L)$.

Next, suppose that P has bounded term-depth in the answers it may produce, but not in its subgoals – i.e. any query to P will have an SLG evaluation in which for the final forest each tree is finite, but where there may be an infinite number of them. Suppose, then that \mathcal{F} has an infinite number of finite trees. In this case, there must have been some first forest \mathcal{F}_{inf} in \mathcal{E} that contained an infinite SDG. \mathcal{E}^L will also first contain an infinite SDG at some forest \mathcal{F}_{inf}^L in its evaluation (note that Local evaluation does not act differently than SLG at limit ordinals). At this point, \mathcal{E}^L can be constructed to be a superset of \mathcal{F}_{inf} . Extending this argument into an induction shows that $I_{\mathcal{F}}|_Q = I_{\mathcal{F}^L}|_Q$.

Next, suppose that P has bounded term-depth in the subgoals it may produce, but not in its answers – i.e. any query to P will have an SLG evaluation in which for the final forest there are a finite number of trees, but where some of these trees may be infinite. Suppose, then that \mathcal{F} has a finite number of infinite trees. If at some successor ordinal n , \mathcal{E}^L encounters in a maximal SCC a tree for which an infinite number of operations will be applicable no operations may be performed outside of that SCC until the first limit ordinal after n . At that time the maximal SCC will be completely evaluated and a Local evaluation will perform a COMPLETION operation. Again, extending this argument into an induction shows that $I_{\mathcal{F}}|_Q = I_{\mathcal{F}^L}|_Q$.

Next, if \mathcal{F} has an infinite number of infinite trees the argument is a combination of the previous two: \mathcal{E}^L will be able to complete any infinite tree after a limit ordinal, and if there are an infinite number of trees, \mathcal{E}^L will be unrestricted and can construct a corresponding forest in \mathcal{F} .

Other cases are that \mathcal{F} is finite, but \mathcal{F}^L is infinite in its number or size of trees; or that \mathcal{F}^L may be infinite in a respect that \mathcal{F} is not. These cases are argued as above: at a limit ordinal either \mathcal{F}^L completes its infinite trees, or, if it has an infinite number of trees, reverts to an unrestricted SLG evaluation.

Theorem 3 Let \mathcal{E}^L be a finite local SLG evaluation. For each \mathcal{F} in \mathcal{E}^L $SDG(\mathcal{F})$ has one and only one independent SCC.

Proof. Let $\mathcal{E} = \mathcal{F}_0 \dots \mathcal{F}_n$ be a finite Local SLG evaluation. We prove by induction that $\forall 0 \leq k \leq n : SDG(\mathcal{F}_k)$ has one and only one independent SCC.

Induction base: $\mathcal{F}_0 = \{Q :- |Q\}$ has only the tree for subgoal Q . $SDG(\mathcal{F}_0) = (Q, (Q, Q))$ which is itself an independent SCC.

Induction hypothesis: Let $\phi(k)$ be the formula: $\forall 0 \leq i \leq k : SDG(\mathcal{F}_i)$ is connected and has a single independent SCC. Assume $\phi(k-1)$ to show $\phi(k)$.

Clearly, k must be a successor ordinal, so from \mathcal{F}_{k-1} we obtain \mathcal{F}_k by applying one of the SLG operations of Definition 7 to one of the trees whose root is in the independent SCC of $SDG(\mathcal{F}_{k-1})$. This may be:

1. **NEW SUBGOAL:** For a node N in a tree for subgoal S' , this operation creates a new tree $S :- |S$ in \mathcal{F}_k . For this operation to be applied, there must have been no tree for S in \mathcal{F}_{k-1} . Thus S was in a trivial SCC in $SDG(\mathcal{F}_{k-1})$. Since trivial SCCs are independent, S was in an independent SCC in \mathcal{F}_{k-1} , and by the induction hypothesis it was the only independent SCC. Since $SDG(\mathcal{F}_k)$ is the same as $SDG(\mathcal{F}_{k-1})$ except for the addition of the edge from S to itself, $\phi(k)$ holds.
2. **PROGRAM CLAUSE RESOLUTION:** Applied to a root node $S' :- |S'$, this operation creates a node $S' :- |Body$ in \mathcal{F}_k . If $Body$ is empty, $SDG(\mathcal{F}_k) = SDG(\mathcal{F}_{k-1})$ and $\phi(k)$ holds. There are several cases where $Body$ is not empty. Let S be the underlying subgoal of the selected literal of $Body$.
 - (a) There is no tree $S :- |S$ in \mathcal{F}_{k-1} . In this case, $SDG(\mathcal{F}_k)$ adds a new edge from S' to S . The SCC for S is trivial so S is independent. By $\phi(k-1)$ S' was the only independent SCC in \mathcal{F}_{k-1} so $\phi(k)$ holds.
 - (b) There is an edge from S' to S already in \mathcal{F}_{k-1} . $SDG(\mathcal{F}_{k-1}) = SDG(\mathcal{F}_k)$ and by $\phi(k-1)$, $\phi(k)$ holds.
 - (c) There is a tree $S :- |S$ in \mathcal{F}_{k-1} , and S is in the independent SCC of \mathcal{F}_{k-1} . In this case by $\phi(k-1)$, S' must also have been in the independent SCC of \mathcal{F}_{k-1} , so that while it may contain a new edge from S' to S , the single independent SCC of \mathcal{F}_k is unchanged from \mathcal{F}_{k-1} , and $\phi(k)$ holds.
 - (d) There is a tree $S :- |S$ in \mathcal{F}_{k-1} , and S is *not* in the independent SCC of \mathcal{F}_{k-1} . In this case, \mathcal{F}_k contains an edge from S' to S . By $\phi(k-1)$, and since S' was in the single independent SCC of $SDG(\mathcal{F}_{k-1})$, and since $SDG(\mathcal{F}_{k-1})$ was connected, there was a path from S to S' . The SCCs of S and S' are thus merged, along with any SCCs between that of S and that of S' . This new SCC forms the new single independent SCC of \mathcal{F}_k , and $\phi(k)$ holds.
3. **ANSWER RETURN and DELAYING:** Each of these operations create a new node $S' :- DelaySet|Body$ within an existing SLG tree, where $Body$ may either be empty or have a selected subgoal S . The analysis of these cases is identical to that of PROGRAM CLAUSE RESOLUTION.

4. **NEGATION RETURN:** This operation creates a new node $S' :- DelaySet|Body$ or a failure node within an existing SLG tree. If a node $S' :- DelaySet|Body$ is produced, the analysis of cases is identical to that of **PROGRAM CLAUSE RESOLUTION**. If a failure node is produced, then $SDG(\mathcal{F}_k) = SDG(\mathcal{F}_{k-1})$.
5. **SIMPLIFICATION:** This operation creates a new node $S' :- DelaySet|Body$ or a failure node within an existing SLG tree. If a node $S' :- DelaySet|Body$ is produced, then $Body$ is the same as the parent of S' and $SDG(\mathcal{F}_k) = SDG(\mathcal{F}_{k-1})$. If a failure node is produced, then $SDG(\mathcal{F}_k) = SDG(\mathcal{F}_{k-1})$.
6. **COMPLETION:** Let $\mathcal{F}_i, i < k$ be the forest in \mathcal{E} such that the oldest subgoal S_{first} in the independent SCC of $SDG(\mathcal{F}_k)$ was added by the operation that produced \mathcal{F}_{i+1} . Consider any operation applied to an arbitrary $\mathcal{F}_j, i < j < k$:
 - (a) (**NEW SUBGOAL**) added a tree T to $SDG(\mathcal{F}_j)$. As this will be the only new independent SCC (induction hypothesis), the independent SCC of $SDG(\mathcal{F}_{j+1})$ will contain it. By $\phi(k-1)$ the subgoal for T will be later either be merged into the independent SCC of $SDG(\mathcal{F}_{k-1})$ or will not be present in $SDG(\mathcal{F}_{k-1})$ if T was marked as completed by a **COMPLETION** operation between \mathcal{F}_j and \mathcal{F}_k .
 - (b) (**COMPLETION**) By $\phi(n-1)$, removed the only independent SCC of $SDG(\mathcal{F}_j)$ which (the SCC of) S_{first} depended on (perhaps indirectly).
 - (c) (**Other Operations**). These operations affect $SDG(\mathcal{F}_j)$ in one of the following manners
 - i. $SDG(\mathcal{F}_j)$ is unchanged.
 - ii. The operation added a new edge to a node in the independent SCC of $SDG(\mathcal{F}_j)$. In this case, the independent SCC of $SDG(\mathcal{F}_j)$ is unchanged.
 - iii. The operation added a new edge to a node *not* in the independent SCC of $SDG(\mathcal{F}_j)$. As with the case of **NEW SUBGOAL**, this independent SCC of $SDG(\mathcal{F}_j)$ will either be merged into the independent SCC of $SDG(\mathcal{F}_{k-1})$ or removed by a **COMPLETION** operation.

Completing the independent SCC from $SDG(\mathcal{F}_{k-1})$ will thus remove all vertices and edges from $SDG(\mathcal{F}_{k-1})$ that were added between \mathcal{F}_j and \mathcal{F}_{k-1} and that had not been removed by a prior **COMPLETION** operation. This the independent SCC of $SDG(\mathcal{F}_k)$ will be that of $SDG(\mathcal{F}_i)$.
7. **ANSWER COMPLETION:** This operation will create failure nodes only, which will not affect the SDG. Thus $SDG(\mathcal{F}_k) = SDG(\mathcal{F}_{k-1})$.

Theorem 4 *Let \mathcal{E}^L be a finite delay-avoiding Local evaluation of a goal G to a program P , and let \mathcal{E} be an SLG evaluation of G to a P such that \mathcal{E} does not use Early Completion, Then for any forest \mathcal{F}^L in \mathcal{E}^L , there exists a forest \mathcal{F} in \mathcal{E} such that $SDG(\mathcal{F}^L)$ is a subgraph of $SDG(\mathcal{F})$.*

Proof. Shown by induction on the sequence of forests in \mathcal{E}^L .

Induction base: The initial forest for both evaluations is the same, so the induction base holds.

Induction hypothesis: Let $\phi(k)$ be the formula: *For all $k' < k$, if $\mathcal{F}_{k'}^L$ is a forest in \mathcal{E}^L then there is a forest \mathcal{F}_j in \mathcal{E} such that $SDG(\mathcal{F}_k^L)$ is a subgraph of $SDG(\mathcal{F}_j)$.* Assume $\phi(k-1)$ to show $\phi(k)$. k must of course be a successor ordinal, so we consider different effects of the SLG operations on $SDG(\mathcal{F}_k^L)$.

- PROGRAM CLAUSE RESOLUTION, ANSWER RETURN, NEGATION RETURN, DELAYING: These operations may add a new edge and possibly a new vertex to $SDG(\mathcal{F}_k^L)$. Suppose then that one of these operations was applied to a tree for subgoal S in \mathcal{F}_{k-1}^L . By $\phi(k-1)$ $SDG(\mathcal{F}_{k-1}^L)$ is a subgraph of $SDG(\mathcal{F}_j)$ for some \mathcal{F}_j in \mathcal{E} . Since \mathcal{E} does not use Early Completion, the same operation must be applied to the tree for S at some later point in \mathcal{E} , to produce a forest that we denote as $\mathcal{F}_{j'}$. To show our induction step it is sufficient to show that $SDG(\mathcal{F}_k^L)$ is a subgraph of $SDG(\mathcal{F}_{j'})$. Every subgoal in $SDG(\mathcal{F}_k^L)$ depends on S due to Theorem 3. These subgoals cannot be completed until S itself is completed. Since \mathcal{E} does not make use of Early Completion, and since one of the above operations is used to produce $SDG(\mathcal{F}_{j'})$, S cannot be completed in $SDG(\mathcal{F}_{j'})$, nor can any other subgoal in $SDG(\mathcal{F}_{j'})$ that is also in $SDG(\mathcal{F}_k^L)$. Since edges cannot be removed from an SDG except by completing subgoals, $SDG(\mathcal{F}_k^L)$ must be a subgraph of $SDG(\mathcal{F}_{j'})$.
- NEW SUBGOAL This operation adds a new node and edge to $SDG(\mathcal{F}_k^L)$. The argument to support the induction step is essentially the same as with the above operations.
- COMPLETION: Since COMPLETION removes vertices and edges from $SDG(\mathcal{F}_{k-1})$, $\phi(k)$ follows immediately from $\phi(k-1)$.
- SIMPLIFICATION, ANSWER RETURN: These operations do not alter $SDG(\mathcal{F}_k^L)$, so $\phi(k)$ follows immediately from $\phi(k-1)$.

Theorem 5 [Correctness of SLG_C] *Let P be a program and \mathcal{G} a finite non-empty set of goals. Then a SLG_C evaluation of \mathcal{G} against P exists with final state $\widehat{\mathcal{F}}$, iff for every $G_i \in \mathcal{G}$ there exists an SLG evaluation of G_i against P with final state \mathcal{F}^i and $I_{\widehat{\mathcal{F}}} = (\bigcup I_{\mathcal{F}^i})$.*

Proof.

- **Soundness:** When the cardinality of \mathcal{Q} is 1, the proof is straightforward, as the SLG_C operations of Definition 14 differ from those of Definition 7 in two ways. First, there is a new USURPATION operation added to Definition 14, and second, some operations are altered in Definition 14 to take account of thread identifiers. Since thread identifiers do not affect the interpretation of a forest, neither the USURPATION operation nor the alterations in SLG operations will affect soundness.
When the cardinality of \mathcal{Q} is greater than 1, note that three-valued interpretations are sets of literals and can be unioned as sets.
- **Completeness** Due to the similarity of SLG and SLG_C operations, demonstration of completeness of SLG_C hinges on showing that any final forest \mathcal{F} in a non-floundering SLG_C evaluation has completed each tree in \mathcal{F} , and

performed all SIMPLIFICATION and ANSWER COMPLETION operations. Since SIMPLIFICATION and ANSWER COMPLETION operations are unchanged from SLG to SLG_C , we need only consider the case where an SLG_C has a final forest with at least one non-completed tree. Let \mathcal{F} a forest with at least one non-completed tree, and suppose that no SLG_C operation is applicable in \mathcal{F} with the possible exception of USURPATION. USURPATION will be applicable to some set \mathcal{S} of subgoals in \mathcal{F} if \mathcal{S} is in deadlock (Definition 15). If \mathcal{F} is finite, then there must be some SCC, \mathcal{S} , in $SDG(\mathcal{F})$ in which the subgoals have different markings. However, in this situation \mathcal{S} fits the definition of a deadlock. Alternately, suppose \mathcal{F} is not finite. In this case there is also the possibility of an infinite chain in $SDG(\mathcal{F})$ in which the subgoals in the chain are marked with different threads. However, this situation is again captured by the definition of deadlock. Thus a USURPATION operation must be applicable to \mathcal{F} and completeness can thereby be shown.

Theorem 6 [Complexity of USURPATION] *Let \mathcal{E} be a finite SLG_C evaluation with final forest \mathcal{F} , and $\mathcal{S}_{\mathcal{F}}$ the set of all subgoals in \mathcal{F} . Then if \mathcal{E} does not make use of Early Completion, there are at most $|\mathcal{S}_{\mathcal{F}}|$ USURPATION operations performed, and at most $2|\mathcal{S}_{\mathcal{F}}|$ USURPATION operations performed otherwise.*

Proof. First, consider the case where \mathcal{E} does not make use of Early Completion. Whenever a USURPATION operation occurs on a set \mathcal{S} of subgoals, all subgoals \mathcal{S} are set to have the same marking. No USURPATION operation will be applicable to any subgoal in \mathcal{S} until the subgoals in \mathcal{S} depend on at least one other subgoal not in \mathcal{S} . Thus there can be at most one USURPATION operation per subgoal in \mathcal{S} leading to a maximum of $|\mathcal{S}_{\mathcal{F}}|$ usurpation operations overall.

Next, consider the case where \mathcal{E} does make use of Early Completion, and a set \mathcal{S} or subgoals to which a USURPATION operation has been applied. If some $S \in \mathcal{S}$ is completed early, then the set of subgoals $\mathcal{S} - \{S\}$ may become disconnected. However, these subgoals will retain their now uniform thread marking so that no $S_1 \in \mathcal{S}$ will require another USURPATION operation until it becomes deadlocked with some $S' \notin \mathcal{S}$. Consider the situation where the marking of S_1 is changed through another USURPATION operation and then later becomes mutually dependent with some $S_2 \in \mathcal{S}$. For this to happen, the SCC whose subgoals are affected by the USURPATION operation must contain at least one subgoal S' not in \mathcal{S} . At this stage, all subgoals in $\mathcal{S} \cup \{S'\}$ are now marked with the same thread, so that the cardinality of the set has grown by 1 at the expense of two USURPATION operations leading to a maximum of $2|\mathcal{S}_{\mathcal{F}}|$ usurpation operations overall.

Theorem 7 [Correctness of Concurrent Local SLG] *Let P be a program and \mathcal{G} a finite non-empty set of goals. Then a Concurrent Local SLG evaluation of \mathcal{G} against P exists with final state $\hat{\mathcal{F}}$, iff every $G_i \in \mathcal{G}$ there exists an SLG evaluation of G_i against P with final state \mathcal{F}^i and $I_{\hat{\mathcal{F}}} = (\bigcup I_{\mathcal{F}^i})$.*

Proof.

- **Soundness of Local SLG_C :** Since Local evaluation restricts the SLG_C operations available, soundness follows from soundness of Concurrent Theorem 5).
- **Completeness of Local SLG_C :** The argument here is much like that for completeness of SLG_C Theorem 5 and Local Evaluation Theorem 2. If $|\mathcal{Q}| = 1$, no deadlocks will ensure, so no USURPATION operations will be performed. Alterations of SLG operations in SLG_C affect only the markings of trees, so completeness follow directly from Theorem 2. If $|\mathcal{Q}| > 1$, consider first the case where P has bounded term-depth. In such a case, a given forest may have no operations applicable for a non-completed tree T with a given thread marking t , because they depend on subgoals marked by another thread. In such a case, either the subgoal for T that t depends on will be completed by another thread, or it will be usurped by another thread, completed, and a tree marked by t will have operations applicable to it, either T itself, or possibly an ancestor of T if T was usurped. In either case, t will be able to complete its goal. Finally, consider the case where $|\mathcal{Q}| > 1$ and P does not have bounded term-depth. In threads states with a finite number of active trees, a thread performs local SLG_C evaluation as in the preceding paragraph; otherwise it performs general SLG_C evaluation where completeness is covered by Theorem 5.

Theorem 8 *Let \mathcal{F} be a forest in a finite Concurrent Local SLG evaluation \mathcal{E} . Then for each active thread t in \mathcal{F} , the Thread SDG(\mathcal{F}) for t has one and only one independent SCC.*

Proof. (Sketch) The proof is a modification of Theorem 3 to apply to the Thread SDGs and to handle the case of the USURPATION operation on a forest \mathcal{F}_k for successor ordinal k . Modifications for Threads SDGs are trivial. To extend for the USURPATION operation, consider that for the usurping thread, $t_{usurper}$, the USURPATION by definition extends the independent SCC of $Thread_SCC(\mathcal{F}, t_{usurper})$ (the single independent SCC exists by the induction hypothesis) with additional subgoals that are strongly connected to subgoals in $Thread_SCC(\mathcal{F}, t_{usurper})$.

For an usurped thread, $t_{usurped}$ USURPATION simply removes all subgoals in the independent SCC of $Thread_SCC(\mathcal{F}, t_{usurped})$ (the single independent SCC exists by the induction hypothesis). In this case, the alteration of $Thread_SCC(\mathcal{F}, t_{usurped})$ is exactly the same as with the COMPLETION operation, and follows from a similar argument.

Theorem 9 *Let \mathcal{F} be a forest in a finite Concurrent Local SLG evaluation. Then for each node in TDG(\mathcal{F}) there is at most one outgoing edge.*

Proof. Clearly the theorem holds in the case where no tree marked by a given thread depends on a subgoal marked by a different thread. Accordingly, let

t_0 be a thread state in a forest \mathcal{F} , and suppose the independent SCC of $Thread_SDG(\mathcal{F}, t_0)$ depends on a subgoal S marked by another thread, t_1 . Clearly, the theorem holds in this case. Three different actions could occur in subsequent forests. First, S may be completed by t_1 or another usurping thread other than t_0 without expanding the SCC of S into other subgoals marked by t_1 , and the theorem will continue to hold as at this point t_0 will depend on no other threads in $TDG(\mathcal{F})$. Second, S may be involved in a deadlocked set and usurped by t_0 – and again the theorem will hold as t_0 will depend on no other threads in $TDG(\mathcal{F})$. Finally, the SCC of S may expand to include other subgoals marked by t_0 leading to a deadlocked set in a forest \mathcal{F}_{dl} . In such a case of deadlock, additional subgoals may depend on subgoals marked by t_0 , but $Thread_SDG(\mathcal{F}_{dl}, t_0)$ will be unchanged from $Thread_SDG(\mathcal{F}, t_0)$. If t_0 usurps the deadlocked set, the theorem will continue to hold as t_0 will mark no trees depending on subgoals owned by another thread. Finally if some other thread t_2 usurps the deadlocked set, the theorem will hold by Corollary 1 applied to the $Thread_SDG$ of t_0 .