Theoretical Computer Science 254 (2001) 465–499

Theoretical Computer Science

www.elsevier.com/locate/tcs

# The limits of fixed-order computation☆

Konstantinos Sagonas[a,*], Terrance Swift[b], David S. Warren[b]

[a] *Computing Science Department, Uppsala Universitet, Box 311, S-751 05 Uppsala, Sweden*
[b] *Department of Computer Science, SUNY at Stony Brook, Stony Brook, NY 11794-4400, USA*

Received March 1998; revised July 1999
Communicated by G. Levi

**Abstract**

Fixed-order computation rules, used by Prolog and most deductive database systems, do not suffice to compute the well-founded semantics (Van Gelder et al., J. ACM 38(3) (1991) 620–650) because they cannot properly resolve loops through negation. This inadequacy is reflected both in formulations of SLS-resolution (Przymusinski, in: Proc. 8th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, ACM Press, Philadelphia, Pennsylvania, March 1989, pp. 11–21; Ross, J. Logic Programming 13(1) (1992) 1–22) which is an ideal search strategy, and in more practical strategies like SLG (Chen and Warren, J. ACM 43(1) (1996) 20–74), or Well-Founded Ordered Search (Stucky and Sudarshan, J. Logic Programming 32(3) (1997) 171–206). Typically, these practical strategies combine an inexpensive fixed-order search with a relatively expensive dynamic search, such as an alternating fixed point (Van Gelder, J. Comput. System Sci. 47(1) (1993) 185–221). Restricting the search space of evaluation strategies by maximizing the use of fixed-order computation is of prime importance for efficient goal-directed evaluation of the well-founded semantics. Towards this end, the theory of *modular stratification* (Ross, J. ACM 41(6) (1994) 1216–1266), formulates a subset of normal logic programs whose literals can be statically reordered so that the program can be evaluated using a fixed-order computation rule. The class of modularly stratified programs, however, is not closed under simple program transformations such as the HiLog transformation. We address the limits of fixed-order computation by adapting results of Przymusinski (1992) to formulate the class of *left-to-right dynamically stratified programs*, and show that this class properly includes other classes of fixed-order stratified programs. We then introduce $SLG_{\mathrm{strat}}$, a variant of SLG resolution that uses a fixed-order computation rule, and prove that it correctly evaluates ground left-to-right dynamically stratified programs. Finally, we indicate how $SLG_{\mathrm{strat}}$ can be used as a basis for computing the well-founded semantics through a search strategy called $SLG_{\mathrm{RD}}$, for *SLG with Reduced use of Delaying*. © 2001 Elsevier Science B.V. All rights reserved.

*Keywords:* Stratification theories; Well-founded semantics; Non-monotonic reasoning; Tabling

☆ A preliminary version of this article appeared in Proceedings of the International Workshop on Logic in Databases (LID-96), San Miniato, Italy, Lecture Notes in Computer Science, Vol. 1154, Springer.
* Corresponding author.
*E-mail addresses:* kostis@csd.uu.se (K. Sagonas), tswift@cs.sunysb.edu (T. Swift), warren@cs.sunysb.edu (D.S. Warren).

## 1. Introduction

Non-stratified normal logic programs – logic programs that contain recursion through default negation – are beginning to be used for practical purposes after many years of study. Two main semantics have been formulated for non-stratified programs: a skeptical semantics, the well-founded semantics (WFS) [29], and a credulous semantics, the stable model semantics [9]. Each of these semantics has led to serious implementation efforts: XSB [22] for the well-founded semantics, and smodels [14] for the stable model semantics. These implementations have begun to give rise to practical applications of non-stratified programs. Examples include *Diagnostica*, a system for diagnosing pediatric psychiatric disorders [12], which uses the reasoning techniques of [1], and *XMC*, a logic model checker [18].

The WFS occupies its prominent place as a skeptical semantics for non-stratified programs for several reasons: It can be characterized as the only semantics for normal programs which is both *rational* and *cumulative* [6]. Furthermore, the WFS can be computed with polynomial data complexity for function-free programs, and as a result several practical evaluation strategies have been formulated for it. Despite all these properties, the WFS cannot be computed using a fixed computation rule. The reason is that an evaluation of a normal program may encounter unresolvable negative dependencies, such as a cycle through negation, and mechanisms are needed to resolve such dependencies. The standard procedural semantics for the WFS is SLS-resolution [17, 23]. Unfortunately, SLS-resolution is not effective in general. As formulated by Przymusinski [17] SLS-resolution uses an oracle to select literals from lower dynamic strata, while Global SLS-resolution as formulated by Ross [23] requires a positivistic computation rule and ideal evaluation of all negative literals in parallel. These departures from fixed computation are naturally reflected in practical strategies for evaluating the WFS [3, 5, 13, 27]. Typically, these strategies consist of a phase that uses a fixed computation rule, followed by some other phase – usually more expensive – that alters that computation rule to try to break recursion through negation, if necessary. In the case of the SLG resolution strategy of [5], tabling is the basis of the first of these phases while delay and simplification steps are used to break cycles through negation. In the Well-Founded Ordered Search of [27], magic templates with Ordered Search [20] are used for the first phase, while an alternating fixed point similar to that of [28] is used for the second. The limitations of fixed-order computation [1] can be seen clearly in the following example.

[1] Without loss of generality, a fixed left-to-right computation rule can be used to exemplify the problems of all fixed computation rules, and so we will equate the left-to-right computation rule with general fixed rules throughout the article. All results can be easily extended to any fixed computation rule.

**Example 1.1** (*Ross* [24]). Consider the following program and query `?- p(a)`:

```
C1: p(X) ← t(X,Y,Z), ¬p(Y), ¬p(Z).
C2: p(b).
C3: t(a,b,a).
C4: t(a,a,b).
```

The rule instance `p(a) ← t(a,Y,Z), ¬p(Y), ¬p(Z).` cannot be evaluated using any fixed computation rule. After `t(a,b,a)` has been resolved against the first literal of that rule instance, the second literal must be chosen before the third to avoid the negative dependency through `p(a)`; after `t(a,a,b)` has been used for resolution, the second literal should be skipped and the third literal must be resolved in order to properly fail for `p(a)`.

Fixed computation rules are important for efficient implementations of systems based on logic; for example most database optimizations rely on the presence of a fixed computation rule. More importantly, however, a break from a fixed computation rule subtly undermines the goal orientation of techniques like tabling or magic templates since it may unnecessarily open up the search space by introducing subgoals that need not be *relevant* to proving a query.

Given the importance of a fixed computation rule, it is natural to determine conditions under which an evaluation has to break the fixed-order computation. The most notable attempt to address this problem is through the formalism of *modular stratification*. From the perspective of fixed computation rules, the salient feature of this class of programs, is that if a program is modularly stratified, then it is statically reorderable so that it is modularly stratified for a fixed computation rule. This property has made modular stratification useful for deductive database systems based on magic-set evaluation (e.g. [21]). However, whether a program is modularly stratifiable at all is undecidable. Furthermore, programs which can be evaluated using a fixed computation rule may not be modularly stratified, as shown by the program in the following example.

**Example 1.2.** The following program is neither *modularly*, nor *weakly stratified* [15]:

```
s ← ¬s, p.
s ← ¬p, ¬q, ¬r.
p ← q, ¬r, ¬s.
q ← r, ¬p.
r ← p, ¬q.
```

To our knowledge, the only stratification class that includes the program in Example 1.2 is *dynamic stratification* of [17] (also independently introduced by Bidoit and Froidevaux as *effective stratification* in [2]). From the perspective of the well-founded semantics, dynamic stratification is extremely powerful since all normal logic programs with two-valued well-founded models are dynamically stratified. Given the power of dynamic stratification, the goal of this article is to study a restriction of this class to a fixed-order computation rule. In addition to forming a stratification class of independent interest, this restriction gives insight into the limits of fixed-order computations for normal programs, and is extended to an effective evaluation method for the well-founded semantics. More specifically, the contributions of this article are as follows.

1. Using the formalism of dynamic stratification, we define the subclass of *left-to-right dynamically stratified* (*LRD-stratified*) programs. We show that this class properly includes the class of left-to-right weakly stratified programs, and effectively properly includes all modularly stratified programs as well. We then show that the class of LRD-stratified programs is closed under meta-interpretation.
2. We introduce a variant of SLG resolution, called $SLG_{\text{strat}}$, which uses a fixed computation rule to evaluate LRD-stratified programs. $SLG_{\text{strat}}$ differs from SLG in that:
   (a) It *suspends* negative literals rather than delaying them.
   (b) It introduces the novel technique of *early completion* which uses information from *within* a dynamic stratum to break and sometimes avoid cycles through negation.

   $SLG_{\text{strat}}$ is shown to be sound and complete for ground LRD-stratified programs.
3. We introduce an extension of $SLG_{\text{strat}}$ called $SLG_{\text{RD}}$ for *SLG with Reduced use of Delaying* which preserves fixed-order computations as much as possible when evaluating normal programs.

The structure of the rest of the article is as follows. After introducing some terminology, we define LRD-stratified programs in Section 2. We then introduce $SLG_{\text{strat}}$ in Section 3, and in Section 4 show the completeness of $SLG_{\text{strat}}$ for ground LRD-stratified programs. Section 5 sketches how $SLG_{\text{strat}}$ can be used as a basis to constrain the search space of possible derivations of full SLG. We conclude with a discussion of how these results are potentially applicable to other evaluation methods for stratified negation. The two appendices contain supporting definitions and theorems for relating the class of LRD-stratified programs with other stratification classes (Appendix A), and proofs for theorems and statements made in the main part of this article (Appendix B).

## 1.1. Terminology

Throughout this article, we assume the standard terminology of logic programming [11]. We assume that any program has a finite number of non-ground rules and is defined over a countable language $\mathcal{LF}$. Rules are defined over atoms and literals in the usual way. The *Herbrand base* $H_P$ of a program $P$ is the set of all ground atoms. By a 3-*valued interpretation* $I$ of a program $P$ we mean a pair $\langle T;F\rangle$, where $T$ and $F$ are subsets of the Herbrand base $H_P$ of $P$. The set $T$ contains all ground atoms that are true in $I$, the set $F$ contains all ground atoms that are false in $I$, and the truth value of the remaining atoms in $U = H_P - (T \cup F)$ is undefined. We usually require that the interpretation $I$ is *consistent*, i.e. that sets $T$ and $F$ are disjoint. If $A$ is a ground atom from $H_P$ then we write $val_I(A) = t$ ($val_I(A) = f$ or $val_I(A) = u$) if $A$ is true (false or undefined) in $I$, respectively. We call $val_I(A)$ the *truth value* of $A$ in $I$. A 3-valued interpretation of $P$ is called a 2-valued interpretation of $P$ if all ground atoms are either true or false in $I$, i.e. if $H_P = T \cup F$. A consistent 3-valued interpretation $M$ is a 3-*valued model* of $P$ if $val_M(C) = t$, for every clause $C$ in $P$. If $M$ is 2-valued then it is called a 2-*valued* or a *total model* of $P$. Any consistent 3-valued interpretation can be viewed as a function from the Herbrand base $H_P$ to the three-element set $\{f, u, t\}$, ordered by $f < u < t$.

## 2. Left-to-right dynamically stratified programs

Most stratification theories determine components of mutually dependent rules based on information that may be statically available from a program, such as predicate dependencies. The use of static information makes these theories easy to apply, but at the same time, limits their applicability to relatively narrow classes of programs. On the other hand, components of dynamic stratification – as the name indicates – are determined dynamically. Intuitively, this dynamic construction of components eliminates atom dependencies that can never be satisfied and so allows computation of the well-founded model.

### 2.1. Iterated fixed point under a left-to-right computation rule

In order to define LRD-stratified programs, we first adapt Przymusinski's definition of the iterated fixed point from [17] to a non-ideal left-to-right computation rule. Our main modification is the introduction of the notion of a *failing prefix* in defining the iterated fixed point operators $\mathscr{T}_I$ and $\mathscr{F}_I$ (Definition 2.1). Intuitively, $I$ represents facts currently known to be true or false and $\mathscr{T}_I(T)$ ($\mathscr{F}_I(F)$) contains facts not contained in $I$, whose truth (falsity) can be immediately derived from $P$ under the interpretation $I$, and assuming that all facts in $T$ are true (all facts in $F$ are false). The failing prefix constraint prevents false facts from inclusion in $\mathscr{F}_I(F)$, unless their falsity can be established by a left-to-right examination of the literals in clauses of $P$.

**Definition 2.1.** For sets $T$ and $F$ of ground atoms we define

$\mathscr{T}_I(T) = \{A : val_I(A) \neq t$ and there is a clause $B \leftarrow L_1, \ldots, L_n$ in $P$ and a ground substitution $\theta$ such that $A = B\theta$ and for every $1 \leqslant i \leqslant n$ either $L_i\theta$ is true in $I$, or $L_i\theta \in T\}$;

$\mathscr{F}_I(F) = \{A : val_I(A) \neq f$ and for every clause $B \leftarrow L_1, \ldots, L_n$ in $P$ and ground substitution $\theta$ such that $A = B\theta$ there exists a *failing prefix*, i.e. there is some $i$ ($1 \leqslant i \leqslant n$), such that $L_i\theta$ is false in $I$ or $L_i\theta \in F$, and for all $j$ ($1 \leqslant j \leqslant i-1$), $L_j\theta$ is true in $I\}$.

Even though the conditions $val_I(A) \neq t$ and $val_I(A) \neq f$ are inessential, they ensure that only *new* facts are included in $\mathscr{T}_I(T)$ and $\mathscr{F}_I(F)$, and will simplify the definition of dynamic strata below. Subsets $T_I$ and $F_I$ of the Herbrand base are obtained by iterating the operators $\mathscr{T}_I$ and $\mathscr{F}_I$ according to the following definition.

**Definition 2.2** (*Przymusinski* [17]). Let $I = \langle T; F \rangle$ be an interpretation, we define

$$T_I^{\uparrow 0} = \emptyset \quad \text{and} \quad F_I^{\downarrow 0} = H_P,$$
$$T_I^{\uparrow n+1} = \mathscr{T}_I(T_I^{\uparrow n}) \quad \text{and} \quad F_I^{\downarrow n+1} = \mathscr{F}_I(F_I^{\downarrow n}),$$
$$T_I = \bigcup_{n<\omega} T_I^{\uparrow n} \quad \text{and} \quad F_I = \bigcap_{n<\omega} F_I^{\downarrow n}.$$

As in [17], both $\mathscr{T}_I$ and $\mathscr{F}_I$ can be seen to be monotonic in the subset inclusion ordering. The following proposition thus holds.

**Proposition 2.1** (Przymusinski [17]). *The transfinite sequence $\{T_I^{\uparrow n}\}$ is monotonically increasing, while the transfinite sequence $\{F_I^{\downarrow n}\}$ is monotonically decreasing. The set $T_I$ is the least fixed point of the operator $\mathscr{T}_I$, and the set $F_I$ is the least fixed point of the operator $\mathscr{F}_I$ over the reverse subset inclusion ordering.*

**Definition 2.3** (*Przymusinski* [17]). Let $\mathscr{I}$ be the operator which assigns to every interpretation $I$ of $P$ a new interpretation $\mathscr{I}(I)$ defined by $\mathscr{I}(I) = I \cup \langle T_I; F_I \rangle$.

The operator $\mathscr{I}$ extends the interpretation $I$ to $\mathscr{I}(I)$ by adding to $I$ new atomic facts $T_I$ which can be derived from $P$ knowing $I$, as well as the negations of new atomic facts $F_I$ which can be assumed false in $P$ by knowing $I$. The monotonicity of $\mathscr{I}$ follows from the monotonicity of $T_I$ and of $F_I$, so that a least fixed point can be defined for this operator.

**Definition 2.4** (*Iterated fixed point*, *Przymusinski* [17]). Let

$$J_0 = \langle \emptyset; \emptyset \rangle,$$
$$J_{\alpha+1} = \mathscr{I}(J_\alpha), \ \text{i.e.} \ J_{\alpha+1} = J_\alpha \cup \langle T_{J_\alpha}; F_{J_\alpha} \rangle,$$
$$J_\alpha = \bigcup_{\beta<\alpha} J_\beta, \ \text{for limit } \alpha.$$

Let $J_{IF(P)}$ denote the fixed point interpretation $J_\delta$, where $\delta$ is the smallest countable ordinal such that both sets $T_{J_\delta}$ and $F_{J_\delta}$ are empty. ($\delta$ exists, and is a countable ordinal because both $T_I$ and $F_I$ are monotonically increasing). We refer to $\delta$ as the *depth* of program $P$.

$J_{IF(P)}$ is a consistent 3-valued interpretation of $P$. It can be proven that $J_{IF(P)}$ is the least fixed point of the operator $\mathscr{I}$. Because the definitions of this section are a restriction of those in [17], such a result is a straightforward application of the proofs in [17]. Note that $J_{IF(P)}$ is an informationally sound approximation of the well-founded model of $P$. Furthermore, as will soon be proven, $J_{IF(P)}$ is the well-founded model of $P$ iff $P$ is left-to-right dynamically stratified. We formally introduce this class of programs in the following section.

## 2.2. Left-to-right dynamic stratification

As observed in [17], the iterated fixed point induces a natural stratification on a program $P$.

**Definition 2.5** (*Dynamic stratum under a fixed-order computation rule*). Let $\alpha$ be a countable ordinal. The $\alpha$'s *dynamic stratum* $S_\alpha$ of $P$ under a left-to-right computation rule is defined as

$$S_\alpha = \{A\colon A \in (T_{J_\alpha} \cup F_{J_\alpha})\}$$

for any $\alpha$ such that $T_{J_\alpha} \cup F_{J_\alpha}$ is not empty. Let $\delta$ be the minimal ordinal such that for all non-empty strata, $S_\alpha$, $\alpha < \delta$; then we call $\delta$ the *ultimate stratum* under a left-to-right computation rule, and

$$S_\delta = \left\{A\colon A \in H_P - \bigcup_{\alpha<\delta} S_\alpha\right\}.$$

Conversely, let $A$ be a ground atom. The *dynamic stratum* of $A$ under a left-to-right computation rule is the unique ordinal number $\alpha$ such that $A \in (T_{J_\alpha} \cup F_{J_\alpha})$, and is $\delta$ otherwise. In general, for any possibly non-ground atom $A$ the *dynamic stratum* of $A$ is equal to

$$\mathit{stratum}(A) = \mathit{sup}\{\mathit{stratum}(A\theta)\colon \theta \text{ is a ground substitution}\}$$

and let $\mathit{str}(\neg A) = \min(\delta, \mathit{str}(A) + 1)$.

The dynamic stratification of $P$ under a left-to-right computation rule is therefore a decomposition of the set of all atoms in $H_P$ into disjoint strata. The $\alpha$'s dynamic stratum of $P$ is defined at level $\alpha < \delta$ as the set of all atoms that were *newly added* to the interpretation $J_\alpha$ in order to obtain $J_{\alpha+1}$ (i.e. those atoms whose truth or falsity was determined at this very level). The ultimate stratum $S_\delta$ contains all the remaining atoms, i.e., all atoms whose truth value could not be determined when the fixed point was reached using a left-to-right computation rule. Some of the atoms in the ultimate stratum may be undefined in the well-founded model $M_P$ of $P$, or their truth value could be determined by using a non-fixed (possibly ideal) computation rule.

**Definition 2.6** (*Left-to-right dynamically stratified program*). A program $P$ is *left-to-right dynamically stratified* iff its ultimate stratum $S_\delta$ is empty.

**Theorem 2.2.** *Let $P$ be a left-to-right dynamically stratified program. Then the interpretation $J_{IF(P)}$ is a 2-valued model of $P$ that coincides with the well-founded model of $P$.*

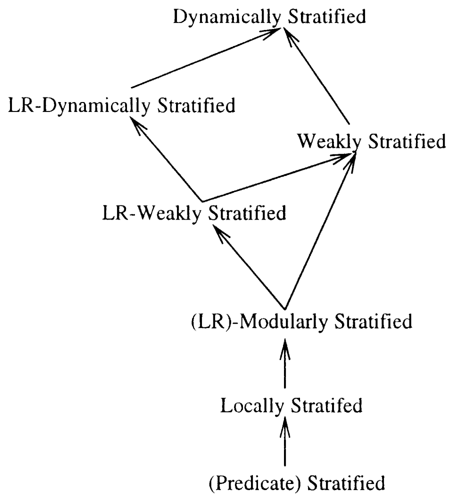

Fig. 1. Stratification hierarchy.

**Proof.** That $J_{IF(P)}$ is 2-valued is trivial, that it coincides with the well-founded model is shown in the proof to Theorem 2.3. □

## 2.3. Relationships between stratification classes

The power of dynamic stratification and of its restriction to a fixed computation rule, arises from the fact that components are determined dynamically rather than statically as they are in *weak* [15], or in *modular stratification* [24]. Another reason for the power of dynamic stratification, as opposed to, e.g. modular or local stratification, is that dynamic stratification uses information from *within* the (dynamic) stratum to eliminate atom dependencies that can never be satisfied. In fact, a hierarchy of stratification classes can be defined as in Fig. 1 where arrows indicate inclusion. Theorem 2.3 substantiates the placement of LRD-stratification within the stratification hierarchy of Fig. 1. The proof of Theorem 2.3 along with supporting definitions is given in Appendix A.

**Theorem 2.3.** *In the stratification hierarchy of Fig.* 1, *there is a path from a class* $\mathscr{C}_1$ *of programs to a class* $\mathscr{C}_2$ *iff every program in* $\mathscr{C}_1$ *is a program in* $\mathscr{C}_2$ *and there exists at least one program in* $\mathscr{C}_2$ *that is not contained in* $\mathscr{C}_1$.

Relationships between the three lowest stratification classes of Fig. 1 are well known in the literature. A stratification class that is not known is the class of *left-to-right weakly stratified* (LRW-stratified) *programs* which restricts weakly stratified programs to a fixed computation rule in a manner similar to the restriction of dynamic stratification to LRD-stratified programs. LRW-stratified programs are formally defined in

Appendix A. Programs that justify the relationships between the classes of LRW-, weakly, dynamically, and LRD-stratified programs are also presented there.

LRD-stratified programs are of interest for at least the following two reasons. First, as indicated in Fig. 1, there are useful programs that are LRD-stratified but are not contained in lower stratification classes. One example of this is the translation of the *alternation-free modal μ-calculus* [7],[2] a temporal logic used for model checking, into an LRD-stratified program [18]. Second, LRD-stratified programs contain other stratification classes as well as common transformations of these programs. The importance of this second point can be seen from the following component testing program example from [24]:

```
working(X) ← tested(X).
working(X) ← part(X,Y),¬has_suspect_part(Y).
has_suspect_part(X) ← part(X,Y),¬working(Y).
```

Provided that the `part/2` extensional database relation is acyclic, this program is modularly stratified, and in fact, is *left-to-right* modularly stratified. For purposes of illustration, we briefly review the well-known essentials of modular stratification. Evaluation of a modularly stratified program involves two steps. First, the program is broken up into recursive components using the predicate dependency graph of the program. Because two separate components cannot be mutually recursive, the component dependency graph will be acyclic. Next, a model is found for each component by reducing the rules in the component with respect to interpretations derived for lower components. More precisely, denoting the dependency relation between components as $\prec_{\mathrm{MS}}$, a program is modularly stratified if for each component $C$, there is a total well-founded model $M$ for the union of all components $C' \prec_{\mathrm{MS}} C$, and if the reduction of $C$ modulo $M$ is *locally* stratified [16].

Modularly stratified programs are useful because a component-by-component evaluation method can be determined statically using the predicate dependency graph. Thus, it can be proven that if a program is modularly stratified then it can be statically reordered into a fixed-order modularly stratified program. Unfortunately, one cannot test whether a program is modularly stratified or not as the problem is undecidable. Furthermore, modular stratification is not preserved by many important programming methods and transformations. For instance, meta-interpreters which are heavily used in logic programming for such purposes as explanation and abduction, are often not modularly stratified. A so-called vanilla meta-interpreter for the above component testing

[2] Briefly, the modal $\mu$ calculus allows greatest and least fixed points to be calculated over formulas that are usually taken to represent states of a concurrent system. Intuitively, a modal $\mu$ calculus formula is alternation-free if the formula can be decomposed so that no greatest fixed point need be evaluated within a least fixed point, and no least fixed point need be evaluated within a greatest fixed point.

program would have the form:

```
demo(true).
demo(A,B) ← demo(A),demo(B).
demo(not A) ← ¬demo(A).
demo(A) ← clause(A,B),demo(B).

clause(working(X),tested(X)).
clause(working(X)),(part(X,Y), not has_suspect_part(Y)).
clause(has_suspect_part(X), (part(X,Y)), not working(Y)).
```

In this transformation the negation operator has been replaced by a unary function symbol that appears nowhere else in the program (here represented by `not/1`). Clearly the meta-interpreter, as written, is not modularly stratified. Appendix A contains a formal statement of the transformation indicated above, along with a proof of the following theorem.

**Theorem 2.4.** *Given a program $P$, let $P_{\text{meta}}$ be the program obtained by the transformation of Definition* A.6. *Then*:

1. *if the ground instantiation of $P$ is* (*LR*) *Modularly Stratified, the ground instantiation of $P_{\text{meta}}$ is LRW-stratified.*
2. *if the ground instantiation of $P$ is LRD-stratified, the ground instantiation of $P_{\text{meta}}$ is LRD-stratified.*

Similar statements can be proven about other types of transformations such as, e.g. the HiLog transformation [4]. More generally, stratification classes that are low in the hierarchy of Fig. 1 (especially those in which strata are based on predicates), may not be closed under simple program transformations.

## 3. Tabled evaluation of LRD-stratified programs

This section presents the formal definitions of $SLG_{\text{strat}}$, a resolution strategy based on tabling which will be shown to evaluate LRD-stratified programs using a left-to-right fixed-order computation rule. $SLG_{\text{strat}}$ is largely a restriction of SLG [5] and inherits most of its properties. Differences of $SLG_{\text{strat}}$ from SLG are noted as they occur. In particular, $SLG_{\text{strat}}$ adds a NEGATIVE SUSPENSION transformation and its necessary underpinnings along with an altered definition of when a set of subgoals is completely evaluated. The latter and the introduction of the NEGATIVE SUSPENSION transformation require a change of the COMPLETION transformation so that suspended computations are resumed.

## 3.1. An Example

To motivate the following sections that contain technical definitions of $SLG_{\text{strat}}$, we present the $SLG_{\text{strat}}$ evaluation of a query `?- q(a).` against the following LRD-stratified program.[3] Each program clause has been labeled for convenience of reference.

$$\begin{array}{ll} C_1\text{: } \mathtt{q(X)} \leftarrow \mathtt{t(X)}, \neg\mathtt{q(X)}. & C_4\text{: } \mathtt{p} \leftarrow \mathtt{t(a)},\ \mathtt{r}. \\ C_2\text{: } \mathtt{q(a)} \leftarrow \neg\mathtt{p}, \neg\mathtt{r}. & C_5\text{: } \mathtt{r} \leftarrow \mathtt{p}, \neg\mathtt{r}, \neg\mathtt{q}. \\ C_3\text{: } \mathtt{t(a)}. & C_6\text{: } \mathtt{q} \leftarrow \mathtt{q(X)}. \end{array}$$

A *system* is a set of pairs of the form $(A:\rho)$, where $A$ is a subgoal and $\rho$ is a sequence of annotated rules for $A$. No two pairs in a system have the same (variant) subgoal. The sequence $\rho$ of annotated rules represents the history of the addition, suspension, and/or deletion of rules for subgoal $A$. The annotation of each rule in $\rho$ indicates where the rule is derived from and whether some rule is suspended or disposed. Intuitively, a rule is disposed when it no longer has anything to contribute to the derivation. In general, the sequence $\rho$ can be transfinite.

The initial system $\mathscr{S}_0$ is empty. The query atom $q(a)$ is introduced as the first subgoal. Using the NEW SUBGOAL transformation, the initial sequence of $q(a)$ is an arbitrary sequence of all rules obtained by resolving $q(a) \leftarrow q(a)$ on $q(a)$ in the body with program clauses. After $q(a)$ is introduced the system is

$$\mathscr{S}_1 = \left\{ q(a)\text{: } \begin{bmatrix} 0\text{: } & \mathtt{q(a)} \leftarrow \mathtt{t(a)}, \neg\mathtt{q(a)}. & \langle C_1 \rangle \\ 1\text{: } & \mathtt{q(a)} \leftarrow \neg\mathtt{p}, \neg\mathtt{r}. & \langle C_2 \rangle \end{bmatrix} \right\}.$$

An annotated rule of the form $G\langle C\rangle$, where $C$ is (the label of) a program clause, means that $G$ is obtained by resolution with program clause $C$. The sequence of a subgoal is written as a sequence of pairs $\alpha:AG$ where $\alpha$ is an ordinal and $AG$ is an annotated rule. In general, the annotation in $AG$ indicates how the rule in $AG$ was derived – either using a program clause or an earlier rule in the sequence. In the latter case the annotation also indicates whether the earlier rule in the sequence is disposed. The head of each non-disposed rule in the sequence of a subgoal captures variable bindings that have been accumulated, and the body of each such rule body contains literals that remain to be solved in order to derive answers for the subgoal. For a rule with a non-empty body, a literal is selected from the rule body by a computation rule. The atom of the selected literal is added as a new subgoal if it is not in the current system. In the example above, `t(a)` is selected from the body of `q(a)` $\leftarrow$ `t(a)`, $\neg$`q(a)` and is added as a new subgoal. Like $q(a)$, $t(a)$ has its own sequence of annotated rules obtained by resolution of $t(a) \leftarrow t(a)$ on $t(a)$ in the body with program clauses. So the system becomes

$$\mathscr{S}_2 = \left\{ q(a)\text{: } \begin{bmatrix} 0\text{: } & \mathtt{q(a)} \leftarrow \mathtt{t(a)},\ \neg\mathtt{q(a)}. & \langle C_1 \rangle \\ 1\text{: } & \mathtt{q(a)} \leftarrow \neg\mathtt{p},\ \neg\mathtt{r}. & \langle C_2 \rangle \end{bmatrix} \quad t(a)\text{: } [\, 0\text{: } \ \mathtt{t(a)}. \ \langle C_3 \rangle \,] \right\}.$$

[3] Throughout the rest of this article, we refer to programs with variables as LRD-stratified if their ground instantiation is LRD-stratified.

A rule with an empty body is an *answer*, e.g., t(a) in the above sequence for $t(a)$. Note that this subgoal has obtained all answers it will ever have; no other rules can have anything to contribute to the derivation of answers for t(a). In such a case, if an answer is derived which is a variant of its subgoal, the subgoal and its sequence can be marked as *completed* – that no further operations are required for this sequence. We call such subgoals *early completable* and the operation that is then performed *early completion* of subgoals. Specifically, early completion is a special case where a set of subgoals is completely evaluated and a COMPLETION transformation (see Definition 3.9) is applicable.

Answers are returned to the appropriate selected literals through the POSITIVE RETURN transformation. By returning the answer t(a), the sequence of $q(a)$ is extended as follows:

$$q(a)\text{:} \begin{bmatrix} 0\text{:} & \mathtt{q(a)} \leftarrow \mathtt{t(a)},\ \neg\mathtt{q(a)}. & \langle C_1\rangle \\ 1\text{:} & \mathtt{q(a)} \leftarrow \neg\mathtt{p},\ \neg\mathtt{r}. & \langle C_2\rangle \\ 2\text{:} & \mathtt{q(a)} \leftarrow \neg\mathtt{q(a)}. & \langle 0, t(a)\rangle \end{bmatrix}.$$

This last annotation indicates that q(a) ← ¬q(a) is obtained from the rule corresponding to ordinal 0 by solving the selected atom using an answer $t(a)$. The selected literal of this sequence is negative. Its subgoal does not have any answers, and it is not yet known whether it will ever get any. Consequently, the selected negative literal cannot be resolved at this point. $SLG_{\text{strat}}$ *suspends* negative selected literals whose truth value is not known. This is done by employing the NEGATIVE SUSPENSION transformation. By suspending rule 1 of $q(a)$ (on ¬p) and rule 2 of $q(a)$ (on ¬q(a)), the sequence of $q(a)$ is extended in two steps to that shown in system $\mathscr{S}_9$ below.

Program clause resolution is then used to derive answers for $p$ which is the subgoal of the selected literal of the second suspended rule. After a sequence of NEW SUBGOAL, POSITIVE RETURN, and NEW SUBGOAL transformations the system is as follows:

$$\mathscr{S}_9 = \left\{ \begin{array}{rl} q(a)\text{:} & \begin{bmatrix} 0\text{:} & \mathtt{q(a)} \leftarrow \mathtt{t(a)},\ \neg\mathtt{q(a)}. & \langle C_1\rangle \\ 1\text{:} & \mathtt{q(a)} \leftarrow \neg\mathtt{p},\ \neg\mathtt{r}. & \langle C_2\rangle \\ 2\text{:} & \mathtt{q(a)} \leftarrow \neg\mathtt{q(a)}. & \langle 0, t(a)\rangle \\ 3\text{:} & \textit{suspended} & \langle 2\rangle \\ 4\text{:} & \textit{suspended} & \langle 1\rangle \end{bmatrix} \\ t(a)\text{:} & [0\text{: } \mathtt{t(a)}.\ \langle C_3\rangle] \\ p\text{:} & \begin{bmatrix} 0\text{: } \mathtt{p} \leftarrow \mathtt{t(a)}, \mathtt{r}. & \langle C_4\rangle \\ 1\text{: } \mathtt{p} \leftarrow \mathtt{r}. & \langle 0, t(a)\rangle \end{bmatrix} \\ r\text{:} & [0\text{: } \mathtt{r} \leftarrow \mathtt{p},\ \neg\mathtt{r},\ \neg\mathtt{q}.\langle C_5\rangle] \end{array} \right\}.$$

Note that subgoals $p$ and $r$ form an *unfounded set* [29] and cannot generate any answers. But they are not completed since they have rules in their sequences which have a selected literal and are not disposed. On the other hand, rule 1 of $q(a)$ cannot be unsuspended until the truth value of p is known. $SLG_{\text{strat}}$ handles this situation through the COMPLETION transformation that disposes all rules (that are not answers) of a set of mutually dependent subgoals when these subgoals have performed all possible operations. Applying the COMPLETION transformation to the set $\{p, r\}$ disposes all rules of these subgoals, and unsuspends rules of the system that are suspended on the completion of $p$ or $r$. The resulting system is the following:

$$\mathcal{S}_{10} = \left\{ \begin{array}{rl} q(a)\text{:} & \left[ \begin{array}{ll} 0\text{: } \texttt{q(a)} \leftarrow \texttt{t(a)}, \neg\texttt{q(a)}. & \langle C_1 \rangle \\ 1\text{: } \texttt{q(a)} \leftarrow \neg\texttt{p}, \neg\texttt{r}. & \langle C_2 \rangle \\ 2\text{: } \texttt{q(a)} \leftarrow \neg\texttt{q(a)}. & \langle 0, t(a) \rangle \\ 3\text{: } \textit{suspended} & \langle 2 \rangle \\ 4\text{: } \textit{suspended} & \langle 1 \rangle \\ 5\text{: } \textit{unsuspended} & \langle 1 \rangle \end{array} \right] \\ t(a)\text{:} & [0\text{:}\texttt{t(a)}.\ \langle C_3 \rangle] \\ p\text{:} & \left[ \begin{array}{ll} 0\text{: } \texttt{p} \leftarrow \texttt{t(a)}, \texttt{r}. & \langle C_4 \rangle \\ 1\text{: } \texttt{p} \leftarrow \texttt{r}. & \langle 0, t(a) \rangle \\ 2\text{: } \textit{disposed} & \langle 1 \rangle \end{array} \right] \\ r\text{:} & \left[ \begin{array}{ll} 0\text{: } \texttt{r} \leftarrow \texttt{p}, \neg\texttt{r}, \neg\texttt{q}. & \langle C_5 \rangle \\ 1\text{: } \textit{disposed} & \langle 0 \rangle \end{array} \right] \end{array} \right\}.$$

Subgoals $p$ and $r$ are completed without any answers; by negation as failure literals $\neg\texttt{p}$ and $\neg\texttt{r}$ can be removed using two NEGATIVE RETURN transformations. Note that since the value of subgoal $r$ is known, a NEGATIVE SUSPENSION transformation is not needed. Like $t(a)$, subgoal $q(a)$ also got its most general answer. As a result, it is early completable. A COMPLETION transformation on $q(a)$ takes place and the resulting final system is as follows:

$$\mathcal{S}_{13} = \left\{ \begin{array}{rl} q(a)\text{:} & \left[ \begin{array}{lll} 0\text{:} & \texttt{q(a)} \leftarrow \texttt{t(a)},\ \neg\texttt{q(a)}. & \langle C_1 \rangle \\ 1\text{:} & \texttt{q(a)} \leftarrow \neg\texttt{p},\ \neg\texttt{r}. & \langle C_2 \rangle \\ 2\text{:} & \texttt{q(a)} \leftarrow \neg\texttt{q(a)}. & \langle 0, t(a) \rangle \\ 3\text{:} & \textit{suspended} & \langle 2 \rangle \\ 4\text{:} & \textit{suspended} & \langle 1 \rangle \\ 5\text{:} & \textit{unsuspended} & \langle 1 \rangle \\ 6\text{:} & \texttt{q(a)} \leftarrow \neg\texttt{r}. & \langle 1 \rangle \\ 7\text{:} & \texttt{q(a)} & \langle 6 \rangle \\ 8\text{:} & \textit{disposed} & \langle 2 \rangle \end{array} \right] \\ t(a)\text{:} & [0\text{: } \texttt{t(a)}.\ \langle C_3 \rangle] \\ p\text{:} & \left[ \begin{array}{lll} 0\text{:} & \texttt{p} \leftarrow \texttt{t(a)}, \texttt{r}. & \langle C_4 \rangle \\ 1\text{:} & \texttt{p} \leftarrow \texttt{r}. & \langle 0, t(a) \rangle \\ 2\text{:} & \textit{disposed} & \langle 1 \rangle \end{array} \right] \\ r\text{:} & \left[ \begin{array}{lll} 0\text{:} & \texttt{r} \leftarrow \texttt{p},\ \neg\texttt{r},\ \neg\texttt{q}. & \langle C_5 \rangle \\ 1\text{:} & \textit{disposed} & \langle 0 \rangle \end{array} \right] \end{array} \right\}.$$

Note that all the subgoals of the final system are completed, and that the use of the NEGATIVE SUSPENSION transformation in conjunction with the *early completion* operation

– which takes into account the truth value of subgoals in the system – avoided the loop through negation involving subgoal $q(a)$.

## 3.2. SLG systems

A *subgoal* is an atom. In our tabling framework, two subgoals are considered the same if they are identical up to variable renaming. A *literal* is an atom, or the negation of an atom $A$ denoted as $\neg A$.

$SLG_{\text{strat}}$ evaluations are modeled through sequences of *X-rules* associated with subgoals encountered in a system. The set of subgoals with their sequences of X-rules intuitively corresponds to the forest of trees representation commonly used by tabling methods [3].

**Definition 3.1** (*Chen and Warren* [5]). An *X-rule* $C$ is of the form

$$H \leftarrow L_1, \ldots, L_n,$$

where $H$ is an atom, each $L_i (1 \leqslant i \leqslant n)$ is a literal, and $n \geqslant 0$. If $n = 0$, $C$ is called an *answer*.

A *computation rule* is an algorithm that selects from the body of an X-rule $C$ a literal $L$. $L$ is called the *selected literal* of $C$. If $L$ is an atom, it is also called the *selected atom* of $C$.

Annotated $X$-rules form $X$-elements which comprise the above-mentioned sequences.

**Definition 3.2** (*X-element*). Let $P$ be a program, $C$ be a rule in $P$, $G$ an X-rule, $\alpha$ an ordinal, and $H$ an atom. Then an *X-element* is of the form $G\langle C\rangle, G\langle\alpha\rangle, G\langle\alpha, H\rangle$, *disposed*$\langle\alpha\rangle$, *suspended*$\langle\alpha\rangle$, or *unsuspended*$\langle\alpha\rangle$.

**Definition 3.3** (*X-sequence*, *Chen and Warren* [5]). An *X-sequence* $\rho$ is a mapping from all ordinals that are smaller than some ordinal $\alpha$ to the set of X-elements. The ordinal $\alpha$ is the *length* of $\rho$.

Each subgoal in a system has an associated X-sequence. The X-sequence captures the history of the addition/disposal of X-rules and, in $SLG_{\text{strat}}$, the suspension/unsuspension of X-rules for the subgoal during evaluation. Each X-element in the X-sequence indicates the origin of an X-rule – a rule in a program or an X-rule earlier in the X-sequence – along with whether an X-rule is disposed, suspended, or unsuspended.

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

- If $e$ is of the form $G\langle C\rangle$, where $G$ is an X-rule and $C$ is a rule in a program, then $G$ is created by resolving $A \leftarrow A$ on $A$ in the body with $C$.
- If $e$ is of the form $G\langle\alpha\rangle$, where $G$ is an X-rule, then the X-rule corresponding to $\alpha$ in $\rho$ is disposed and replaced by $G$.

- If $e$ is of the form $G\langle\alpha, H\rangle$, where $G$ is an X-rule and $H$ is an atom, then $G$ is obtained by solving the selected atom of the X-rule corresponding to $\alpha$ in $\rho$ using an answer $H$.
- If $e$ is *disposed*$\langle\alpha\rangle$, then the X-rule corresponding to $\alpha$ in $\rho$ is simply disposed.
- If $e$ is *suspended*$\langle\alpha\rangle$, then the X-rule corresponding to $\alpha$ in $\rho$ is marked as suspended.
- If $e$ is *unsuspended*$\langle\alpha\rangle$, then the X-rule corresponding to $\alpha$ in $\rho$ is marked as unsuspended.

Intuitively, X-rules that are disposed from an X-sequence of a system do not appear in any future point in the sequence. Similarly, X-rules of a subgoal that are marked as suspended do not participate in the evaluation of the subgoal until they are marked as unsuspended.

We consider two main operations over X-sequences: concatenation and the least upper bound of an increasing chain of X-sequences. The former is used to extend the X-sequence of a subgoal, and the latter is needed for the transfinite definition of an $SLG_{\mathrm{strat}}$ derivation that will be presented in Section 3.4.

**Definition 3.4.** Let $\rho_1$ and $\rho_2$ be X-sequences of length $\alpha_1$ and $\alpha_2$, respectively. The *concatenation* of $\rho_1$ with $\rho_2$, denoted by $\rho_1 \cdot \rho_2$, is an X-sequence of length $\alpha_1 + \alpha_2$ such that $(\rho_1 \cdot \rho_2)(i) = \rho_1(i)$ for every $i$ $(0 \leqslant i < \alpha_1)$, and $(\rho_1 \cdot \rho_2)(\alpha_1 + j) = \rho_2(j)$ for every $j$ $(0 \leqslant j < \alpha_2)$.

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

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

**Definition 3.5.** Let $\beta$ be an ordinal and $\rho_i$ $(0 \leqslant i < \beta)$ be an increasing chain of X-sequences (with respect to $\subseteq$), and let $\alpha$ be an ordinal such that the length of each $\rho_i$ $(0 \leqslant i < \beta)$ is less than $\alpha$. Then the least upper bound of the chain $\rho_i$ $(0 \leqslant i < \beta)$, denoted by $\bigcup\{\rho_i \mid 0 \leqslant i < \beta\}$, exists since the length of each $\rho_i$ $(0 \leqslant i < \beta)$ is less than $\alpha$. It is the X-sequence that is the union of all $\rho_i$ $(0 \leqslant i < \beta)$ when an X-sequence is viewed as a set.

Intermediate states of the evaluation of a query are represented by *systems*.

**Definition 3.6** (*System*). Let $P$ be a program, and $R$ be a computation rule. A *system* $\mathscr{S}$ is a set of pairs of the form $(A : \rho)$, where $A$ is a subgoal and $\rho$ is its X-sequence, such that no two pairs in $\mathscr{S}$ have the same subgoal. A subgoal $A$ is said to be *in* $\mathscr{S}$ if $(A : \rho) \in \mathscr{S}$ for some X-sequence $\rho$.

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

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

- $G$ is *disposed* if for some $j>\alpha$, $\rho(j)$ is either *disposed*$\langle\alpha\rangle$ or is $G'\langle\alpha\rangle$ for some X-rule $G'$;
- $G$ is an *answer of A* if $G$ is not disposed and has no literals in its body;
- $G$ is *suspended* on $B$ if $\neg B$ is the selected literal of $G$, $G$ is not disposed, and for some $j>\alpha$, $\rho(j)$ is *suspended*$\langle\alpha\rangle$, and there is no $k>j$ such that $\rho(k)$ is *unsuspended*$\langle\alpha\rangle$; we say that $G$ is *suspended* if there exists a subgoal $B$ such that $G$ is suspended on $B$.
- $G$ is an *active X-rule of A* if $G$ is not disposed or suspended and has a selected literal.

A subgoal $A$ in $\mathscr{S}$ is *completed* if all X-rules of $A$ are either answers, or are disposed.

## 3.3. Transformations of systems of subgoals

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

**Definition 3.7** (*X-resolution*). Let $G$ be an X-rule, of the form $H\leftarrow L_1,\ldots,L_n$, and $L_i$ be the selected atom of $G$ for some $i$ $(1\leqslant i\leqslant n)$. Let $C$ be a rule and $C'$, of the form $H'\leftarrow L'_1,\ldots,L'_m$, be a variant of $C$ with variables renamed so that $G$ and $C'$ have no variables in common. Then $G$ is *X-resolvable* with $C$ if $L_i$ and $H'$ are unifiable. The X-rule:

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

is the *X-resolvent* of $G$ with $C$, where $\theta$ is the mgu of $L_i$ and $H'$.

The selected atom in the body of an X-rule can be solved by an answer; this process is called *answer resolution*. The selected negative literal in the body of an X-rule can be solved if the positive counterpart succeeds or fails, according to the following definition.

**Definition 3.8.** Let $\mathscr{S}$ be a system. A subgoal $A$ in $\mathscr{S}$ *succeeds* if $A$ has an answer of the form $A'\leftarrow$ where $A'$ is a variant of $A$, and *fails* if $A$ is completed in $\mathscr{S}$ without any answers. If $A$ neither succeeds nor is completed in $\mathscr{S}$, then we say that $A$ is *evolving* in $\mathscr{S}$.

Note that a subgoal containing variables succeeds if it has any answer that does not bind any of its variables. As a special case, a ground subgoal succeeds if it has an answer.

The notion of *completely evaluated* captures necessary and sufficient conditions for a set of subgoals to have produced all possible answers.

**Definition 3.9** (*Completely evaluated*). Let $\mathscr{S}$ be a system and $\Sigma$ be a non-empty set of subgoals in $\mathscr{S}$, none of which is completed. $\Sigma$ is said to be *completely evaluated* iff for every subgoal $A \in \Sigma$, where $(A:\rho) \in \mathscr{S}$ for some X-sequence $\rho$, either

- $A$ succeeds, or
- $A$ does not contain any suspended X-rule, and for every active X-rule $G$ of $A$ corresponding to some ordinal $\alpha$ in $\mathscr{S}$, there exists an atom $A_1$ such that:
  - $A_1$ is the selected atom of $G$; and
  - $A_1$ is a subgoal in $\mathscr{S}$ that is either completed or in $\Sigma$; and
  - for every atom $H$ that is the head of some answer of $A_1$ in $\mathscr{S}$, there exists an ordinal $i > \alpha$ and an X-rule $G'$ such that $\rho(i) = G'\langle\alpha, H\rangle$.

For convenience, we call a subgoal $A$ *completely evaluated* iff there exists a set of subgoals $\Sigma$ that is completely evaluated, and $A \in \Sigma$.

Note that according to this definition, any set $\{A\}$ consisting of a single subgoal $A$ that succeeds is a completely evaluated set of subgoals. This condition will be referred to as *early completion* of subgoals. Also note that in a completely evaluated set of subgoals, only those subgoals that succeed can contain X-rules that are suspended on the completion of another subgoal.

Let $P$ be a program, $R$ be an arbitrary but fixed computation rule, and $Q$ be an atomic query. The transformations of $SLG_{\text{strat}}$ are listed below:

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

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

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

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

**Remark.** Repeated subgoals are not solved by resolution with program clauses.

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

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

where $G_2$ is the X-resolvent of $G$ with $C$.

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

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

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

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

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

NEGATIVE SUSPENSION: Let $(A:\rho)\in\mathscr{S}$, where $A$ is a subgoal and $\rho$ is its X-sequence. Let $G$ be an active X-rule of subgoal $A$ corresponding to an ordinal $\alpha$ in $\mathscr{S}$, and let $\neg A_1$ be the selected literal of $G$ such that $A_1$ is evolving in $\mathscr{S}$. Then

$$\frac{\mathscr{S}}{\mathscr{S}-\{(A:\rho)\}\cup\{(A:\rho\cdot \mathit{suspended}\langle\alpha\rangle)\}}.$$

**Remark.** Negative literals whose positive counterparts either are completed or succeed in $\mathscr{S}$ are never suspended.

COMPLETION: Let $\Sigma$ be a non-empty set of subgoals in $\mathscr{S}$ that is completely evaluated, and let $\Xi$ be the set of subgoals in $\mathscr{S}$ containing an X-rule that is suspended on a literal $\neg A_1$ where $A_1\in\Sigma$. Then

$$\frac{\mathscr{S}}{\text{for every } B\in\Xi,\ \text{replace } (B:\varrho)\in\mathscr{S} \text{ with } (B:\varrho\cdot\varrho')},$$

$$\text{and then}$$

$$\text{for every } A\in\Sigma,\ \text{replace}(A:\rho)\in\mathscr{S} \text{ with } (A:\rho\cdot\rho')$$

where

- $\varrho'$ is an arbitrary sequence of X-elements of the form *unsuspended*$\langle\alpha\rangle$, where $\alpha$ is an ordinal such that the X-rule of $B$ corresponding to $\alpha$ in $\varrho$ is an X-rule that is suspended on a literal $\neg A_1$ where $A_1\in\Sigma$.
- $\rho'$ is an arbitrary sequence of X-elements of the form *disposed*$\langle\alpha\rangle$, where $\alpha$ is an ordinal such that the X-rule of $A$ corresponding to $\alpha$ in $\rho$ is an active X-rule.

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

## *3.4. Derivation and SLG$_{\text{strat}}$ resolution*

**Definition 3.10.** Let $P$ be a program, $R$ be an arbitrary but fixed computation rule, and $Q$ be an atomic query. An *SLG$_{\text{strat}}$ derivation* for $Q$ (or *evaluation* of $Q$) is a sequence of systems $\mathscr{S}_0,\mathscr{S}_1,\ldots,\mathscr{S}_\alpha$ such that:

- $\mathscr{S}_0$ is the empty system $\emptyset$;

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

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

*$SLG_{\text{strat}}$ resolution* is the process of constructing an $SLG_{\text{strat}}$ derivation for a query $Q$ with respect to a program $P$ under a computation rule $R$.

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

**Definition 3.11.** Let $P$ be a program, and $\mathscr{S}_1$ and $\mathscr{S}_2$ be systems. Then $\mathscr{S}_1 \sqsubseteq \mathscr{S}_2$ if for every $(A:\rho_1) \in \mathscr{S}_1$, there exists $(A:\rho_2) \in \mathscr{S}_2$ such that $\rho_1 \subseteq \rho_2$, i.e., $\rho_1$ is a prefix of $\rho_2$.

The following proposition ensures that the addition of the *suspended* and *unsuspended* statuses in Definition 3.2, and the introduction of the NEGATIVE SUSPENSION transformation do not affect fundamental properties of SLG resolution. Its proof is contained in Appendix B.

**Proposition 3.1.** *Let $P$ be a program, $R$ be an arbitrary but fixed computation rule, and $Q$ be an atomic query. Then*

(a) *there exists some ordinal $\lambda$ such that for any system $\mathscr{S}$ in any $SLG_{\text{strat}}$ derivation for $Q$, the length of the X-sequence of each subgoal in $\mathscr{S}$ is bounded by $\lambda$;*

(b) *every $SLG_{\text{strat}}$ derivation of $Q$ is a monotonically increasing sequence of systems with respect to $\sqsubseteq$; and*

(c) *there exists some $SLG_{\text{strat}}$ derivation for $Q, \mathscr{S}_0, \mathscr{S}_1, \ldots, \mathscr{S}_\alpha$, for some ordinal $\alpha$ such that $\mathscr{S}_\alpha$ is a final system.*

**Proof.** The proof is given in Appendix B.1. □

Proposition 3.1 shows that some final system can be derived for an atomic query $Q$, given a program $P$ and an arbitrary but fixed computation rule $R$. In our restriction of SLG, it can be the case that for a final system $\mathscr{S}$, no $SLG_{\text{strat}}$ transformation is applicable because some subgoals that are evolving in $\mathscr{S}$ contain suspended X-rules (on subgoals that are also evolving in $\mathscr{S}$); as a result the COMPLETION transformation cannot be applied to them. In this case, we say that the final system is *flummoxed*.

**Definition 3.12** (*Completed and flummoxed system*). Let $\mathscr{S}$ be a system. $\mathscr{S}$ is *completed* if every subgoal in $\mathscr{S}$ is completed, and is *flummoxed* if $\mathscr{S}$ is final and is not completed.

**Example 3.1.** A completed system is $\mathscr{S}_{12}$ of Section 3.1. A system that is flummoxed is shown below; it is the final system of the $SLG_{\text{strat}}$ evaluation of query `?- p(a)` against the program of Example 1.1, which is not LRD-stratified.

$$\mathscr{S} = \left\{ \begin{array}{rl} p(a)\text{:} & \left[ \begin{array}{ll} 0\text{: } p(a) \leftarrow t(a,Y,Z),\ \neg p(Y),\ \neg p(Z). & \langle C_1 \rangle \\ 1\text{: } p(a) \leftarrow \neg p(b),\ \neg p(a). & \langle 0, t(a,b,a) \rangle \\ 2\text{: } \textit{suspended} & \langle 1 \rangle \\ 3\text{: } \textit{disposed} & \langle 1 \rangle \\ 4\text{: } p(a) \leftarrow \neg p(a),\ \neg p(b). & \langle 0, t(a,a,b) \rangle \\ 5\text{: } \textit{suspended} & \langle 4 \rangle \end{array} \right] \\ t(a,Y,Z)\text{:} & \left[ \begin{array}{l} 0\text{: } t(a,b,a).\ \langle C_3 \rangle \\ 1\text{: } t(a,a,b).\ \langle C_4 \rangle \end{array} \right] \\ p(b)\text{:} & \left[ 0\text{: } p(b).\ \langle C_2 \rangle \right] \end{array} \right\}$$

Subgoals $t(a,Y,Z)$ and $p(b)$ are completed, but rule 4 of subgoal $p(a)$ is suspended, and at this point, no $SLG_{\text{strat}}$ transformation is applicable to any rule.

Note that floundering is captured as a special instance of a flummoxed system. In a non-ground program, the final system may be flummoxed either because it contains a rule that cannot be unsuspended, or because some suspended X-rule of a subgoal $A$ in $\mathscr{S}$ has a non-ground selected negative literal that neither succeeds nor fails.[4] Consequently, the corresponding subgoal $A$ cannot be completed. The following lemma thus holds:

**Lemma 3.2.** *Let $P$ be a program, $R$ be an arbitrary but fixed computation rule, and $Q$ be an atomic query. Then for every final system $\mathscr{S}$ for $Q, \mathscr{S}$ is either completed or flummoxed.*

### *3.5. Subgoal dependencies*

An SLG system $\mathscr{S}$ induces a natural dependency relation between its subgoals. A subgoal $A$ *directly depends positively* (*negatively*) on a subgoal $B$ if $B$ ($\neg B$) is the selected literal of an active (suspended) rule of $A$. A subgoal $A$ *depends on* a subgoal $B$ if there is a sequence of subgoals $A_0, A_1, \ldots, A_n$, such that $A_0 = A$, $A_n = B$, and $A_i$ directly depends on $A_{i+1}$, for each $0 \leqslant i \leqslant n-1$. We say that $A$ *depends positively on* $B$ if $A$ depends on $B$ through only positive direct dependencies, and we say that $A$ *depends negatively on* $B$ if $A$ depends on $B$ otherwise. The dependency relation outlined above can be extended to a *subgoal dependency graph* (*SDG*) for a system $\mathscr{S}$ whose vertices are non-completed subgoals in $\mathscr{S}$ and whose labeled links are determined by the dependency relation.

[4] The notion of a flummoxed system in $SLG_{\text{strat}}$ bears some resemblence to the notion of blocked nodes in [3].

**Theorem 3.3.** *Let $P$ be a program, $R$ be an arbitrary but fixed computation rule, $Q$ be an atomic query, and $\mathscr{S}$ be a flummoxed system in an arbitrary $SLG_{\mathrm{strat}}$ derivation for $Q$. Then every non-completed subgoal in $\mathscr{S}$ depends negatively on a subgoal in $\mathscr{S}$ that is not completely evaluated.*

**Proof.** The proof is given in Appendix B.1. □

When restricted to a program and query that reaches a final system after a finite number of steps, the theorem implies that in a flummoxed system, all subgoals that are not completed depend (directly or indirectly) on some subgoal that is involved in a negative loop.

## 3.6. Interpretation defined by an SLG system

Let $P$ be a program, $R$ be an arbitrary but fixed computation rule, and $\mathscr{S}$ be a system in any SLG derivation for an atomic query. As observed in [5], SLG resolution can be viewed as program transformation technique that specializes a program with respect to a query. More specifically, SLG resolution tries to transform a system of subgoals, each having an associated sequence of X-rules, into one that contains only X-rules that are answers for the subgoals. Under this prism, for each subgoal $A$ in $\mathscr{S}$, the multiset of X-rules of $A$ in $\mathscr{S}$ that are not disposed constitutes the set of partial answers for $A$. The head of each X-rule contains relevant variable bindings that have been accumulated; while the remaining literals in the rule body are yet to be solved with respect to the original program $P$. Each SLG system $\mathscr{S}$ represents a program $P_{\mathscr{S}}$ containing partial answers for its subgoals. It is natural, therefore, to associate with every SLG system $\mathscr{S}$ a 3-valued interpretation $I_{\mathscr{S}}$ of $P_{\mathscr{S}}$ that is constructed from $\mathscr{S}$ according to the following definition.

**Definition 3.13** (*Chen and Warren* [5]). Let $P$ be a program, $R$ be an arbitrary but fixed computation rule, and $Q$ be an atomic query. Let $\mathscr{S}$ be a system in an SLG derivation for $Q$. We associate with $\mathscr{S}$ a pair $I_{\mathscr{S}} = \langle T_{\mathscr{S}}; F_{\mathscr{S}} \rangle$, where $T_{\mathscr{S}}$ and $F_{\mathscr{S}}$ are subsets of the Herbrand base $H_P$ of $P$ constructed as follows:
- if a (possibly non-completed) subgoal $A$ in $\mathscr{S}$ has an answer of the form $H \leftarrow$, then every ground instance $B$ of $H$ is true in $I_{\mathscr{S}}$; that is, $B \in T_{\mathscr{S}}$;
- if a subgoal $A$ in $\mathscr{S}$ is completed and $B$ is a ground instance of $A$ such that $B$ is not an instance of any answer of $A$, then $B$ is false in $I_{\mathscr{S}}$; that is, $B \in F_{\mathscr{S}}$;
- no other element of $H_P$ is in $T_{\mathscr{S}} \cup F_{\mathscr{S}}$.

In general, the interpretation $I_{\mathscr{S}}$ captures ground instances of subgoals that succeed or fail. For example, if a subgoal $A$ succeeds in $\mathscr{S}$, then every ground instance of $A$ is in $T_{\mathscr{S}}$, and if a subgoal $A$ fails, then $B \in F_{\mathscr{S}}$ for every ground instance $B$ of $A$. As a special case, in ground programs, a ground atom $A \in T_{\mathscr{S}}$ ($A \in F_{\mathscr{S}}$) if and only if $A$ in $\mathscr{S}$ and $A$ succeeds (fails) in $\mathscr{S}$. From the correctness of our restriction of SLG resolution (to be presented as Theorem 4.3), and the way that the interpretation $I_{\mathscr{S}}$ is constructed, it can be seen that the sets $T_{\mathscr{S}}$ and $F_{\mathscr{S}}$ are disjoint, so that the following proposition can be easily seen to hold:

**Proposition 3.4.** *Let $P$ be a program, $R$ be an arbitrary but fixed computation rule, and let $\mathscr{S}$ be a system in an arbitrary $SLG_{\text{strat}}$ derivation for an atomic query $Q$. Then $I_{\mathscr{S}}$ is a consistent interpretation of $P$.*

Since $I_{\mathscr{S}}$ is a consistent interpretation of $P$, it can be viewed as valuation function of the elements of the Herbrand base $H_P$. In general, a ground instance of a subgoal $A$ that is in $\mathscr{S}$ may be true, false, or undefined in $I_{\mathscr{S}}$, while all other elements of $H_P$ that are not instances of subgoals in $\mathscr{S}$ are undefined in $I_{\mathscr{S}}$. If $B$ is a ground atom from $H_P$, we denote the truth value of $B$ in $I_{\mathscr{S}}$ as $val_{I_{\mathscr{S}}}(B)$.

We say that an interpretation $I' = \langle T'; F'\rangle$ *extends* an interpretation $I = \langle T; F\rangle$ if $T \subseteq T'$ and $F \subseteq F'$. We will denote this fact by $I \leqslant I'$. The following proposition states that each $SLG_{\text{strat}}$ transformation extends the interpretation of the system to which the transformation is applied.

**Proposition 3.5.** *Let $P$ be a program, $R$ be an arbitrary but fixed computation rule, $Q$ be an atomic query, and $\mathscr{S}_0, \mathscr{S}_1, \ldots, \mathscr{S}_\alpha$ be any arbitrary $SLG_{\text{strat}}$ derivation for $Q$, where $\alpha$ is an ordinal. Then the sequence $\{I_{\mathscr{S}_i},\ i > 0\}$ is monotonically increasing, i.e. $I_{\mathscr{S}_i} \leqslant I_{\mathscr{S}_j}$ if $i \leqslant j$.*

**Proof.** The proof follows in a straightforward manner from two observations about $SLG_{\text{strat}}$ evaluation. First, no answer will be disposed once it is added to an X-sequence. Second, once a subgoal is completed, its set of answers will not change. $\square$

## 4. Correctness of $SLG_{\text{strat}}$ for LRD-stratified programs

Recall that for simplicity of presentation a left-to-right computation rule is used, and that the dynamic stratum of a ground atom is also defined under this left-to-right computation rule (Definition 2.5). All results of this section can be extended to any fixed-order computation rule. Proofs for the following two lemmas are contained in Appendix B.2.

**Lemma 4.1.** *Let $P$ be a ground LRD-stratified program, $Q$ be a ground atomic query, and $\mathscr{E}$ be an $SLG_{\text{strat}}$ derivation for $Q$ under a left-to-right computation rule. Then for any completely evaluated subgoal $A$ in a system $\mathscr{S}$ in $\mathscr{E}$: $val_{I_{\mathscr{S}}}(A) = val_{J_{IF(P)}}(A)$. That is, the truth value of subgoal in $I_{\mathscr{S}}$ is the value of the subgoal in $J_{IF(P)}$.*

The following lemma states that the $SLG_{\text{strat}}$ evaluation of ground LRD-stratified programs will never result in a flummoxed system.

**Lemma 4.2.** *Let $P$ be a ground LRD-stratified program, $Q$ be a ground atomic query, and $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_\alpha$ be any $SLG_{\text{strat}}$ derivation for $Q$ under a left-to-right computation rule, such that $\mathcal{S}_\alpha$ is a final system. Then $\mathcal{S}_\alpha$ is completed.*

In order to prove correctness of $SLG_{\text{strat}}$ resolution, we want to compare the interpretation of a system $\mathcal{S}$ for a program $P$ and query $Q$ with the model $M_P$ for that program. In that case, we use the notation $M_P|_{\mathcal{S}}$ to denote the restriction of $M_P$ to (ground) atoms which unify with subgoals in $\mathcal{S}$.

**Theorem 4.3.** *Let $P$ be a ground LRD-stratified program and $Q$ be an atomic ground query. Then any $SLG_{\text{strat}}$ derivation for $Q$ under a left-to-right computation rule will produce a final system $\mathcal{S}_\alpha$ such that $I_{\mathcal{S}_\alpha} = M_P|_{\mathcal{S}_\alpha}$.*

**Proof.** By Proposition 3.1, such an ordinal $\alpha$ will exist. The proof is then by transfinite induction on operations of the $SLG_{\text{strat}}$ evaluation. By Lemma 4.2, $\mathcal{S}_\alpha$ will be completed. By Lemma 4.1, the truth value of completed subgoals will reflect that of the model of $P$. □

Actually, Lemma 4.1 holds for all normal logic programs (whether LRD-stratified or not) and independently of the use of operations like early completion. The use of early completion is necessary however for Lemma 4.2 (and consequently for Theorem 4.3). The following program from [25] illustrates this point:

**Example 4.1.** Consider the execution of query `?- a.` against the following program:

$C_1$: $a \leftarrow b,\ \neg c.$
$C_2$: $b \leftarrow a.$
$C_3$: $b \leftarrow d.$
$C_4$: $b.$
$C_5$: $c \leftarrow \neg d.$
$C_6$: $d \leftarrow b, e.$

The execution of this query causes cascading suspensions, of $a$ on $c$ and of $c$ on $d$, as seen from the (non-final) system below.

$$\mathcal{S} = \left\{ \begin{array}{lll} a\!: \begin{bmatrix} 0\!: a \leftarrow b, \neg c. & \langle C_1 \rangle \\ 1\!: a \leftarrow \neg c. & \langle 0, b \rangle \\ 2\!: \textit{suspended} & \langle 1 \rangle \end{bmatrix} & b\!: \begin{bmatrix} 0\!: b \leftarrow a. & \langle C_2 \rangle \\ 1\!: b \leftarrow d. & \langle C_3 \rangle \\ 2\!: b. & \langle C_4 \rangle \end{bmatrix} & \\ & & e\!: [\ ] \\ d\!: \begin{bmatrix} 0\!: d \leftarrow b, e. & \langle C_6 \rangle \\ 1\!: d \leftarrow e. & \langle 0, b \rangle \end{bmatrix} & c\!: \begin{bmatrix} 0\!: c \leftarrow \neg a. & \langle C_5 \rangle \\ 1\!: \textit{suspended} & \langle 0 \rangle \end{bmatrix} & \end{array} \right\}$$

Observe that subgoals $a$, $b$, $c$, and $d$ depend on each other. However, subgoal $b$ can be early completed through the use of the COMPLETION transformation. This action breaks the loop though negation that the subgoals are involved in and allows $SLG_{\text{strat}}$ resolution to proceed with the completion of the remaining subgoals, in the order $d$, $c$, and finally $a$. If early completion is not performed, the system shown above is final and is flummoxed.

## 5. Beyond LRD-stratified programs

### 5.1. Goal-orientation and relevance

Since as shown by the program of Example 1.1, the well-founded semantics cannot be computed using a fixed computation rule, it is non-trivial to obtain an evaluation strategy that is completely goal directed. In particular, it is difficult given a query $Q$ to guarantee that the computation rule will always select literals that are absolutely essential to prove or disprove $Q$. This measure of relevance is ideal, and it appears to require selecting literals based on knowing a priori their truth value. Assuming that this requirement cannot be met in practice, it is then natural to ask what measure of relevance is possible for an evaluation strategy for the well-founded semantics that uses a non-ideal (for example a mostly fixed-order) computation rule.

Restricting attention to the case where the evaluation encounters only ground subgoals, we describe the notion of *relevance of subgoals to a query* under a fixed computation rule (again, for simplicity of presentation we use a left-to-right computation rule). Let $Q$ be a ground query to a program $P$ and let $P$ contain a rule:

$$r : p(\bar{t}) \text{:-} l_1(\overline{t_1}), \ldots, l_n(\overline{t_n}),$$

where $l_i(\overline{t_i}) = (\neg)s_i(\overline{t_i})$. Then, the subgoal $s_i(\overline{b_i})$ of literal $l_i(\overline{b_i})$ is relevant to $Q$ if there is a ground instance

$$r' : p(\bar{a}) \text{:-} l_1(\overline{b_1}), \ldots, l_i(\overline{b_i}), \ldots$$

of (the head and the first $i$ body literals of) $r$ such that the head $p(\bar{a})$ is relevant to $Q$, and all instantiated literals $l_1(\overline{b_1}), \ldots, l_{i-1}(\overline{b_{i-1}})$ are either true or undefined in the well-founded model of $P$. We believe that even this criterion for relevance is very strong for all normal logic programs and also appears to be unobtainable by non-ideal evaluation strategies. Notice, however, that all subgoals encountered during the $SLG_{\text{strat}}$ evaluation of a query $Q$ against a left-to-right dynamically stratified program are relevant to $Q$ under this criterion. These considerations comprise another reason for our claim that fixed-order dynamic stratification can be considered as the limit of fixed-order computation.

### 5.2. $SLG_{\text{strat}}$ *as a basis for efficient evaluation of the well-founded semantics*

As originally presented, SLG delays literals whose truth value may be undefined (e.g. by a loop through negation), maintaining these literals in a delay list. Once the truth values of these delayed literals becomes known, the literals may be removed from the delay lists (if the literals are true) or answers containing the literal in their delay lists may be removed from the systems (if the literals are false). By including the SLG DELAYING, SIMPLIFICATION, and ANSWER COMPLETION transformations, $SLG_{\text{strat}}$ can be naturally extended to evaluate the well-founded semantics of all normal logic programs in a goal-oriented fashion.[5] The resulting evaluation strategy, which we call $SLG_{\text{RD}}$ (*SLG with reduced use of delaying*), inherits the properties of SLG resolution: it is sound and search space complete with respect to the well-founded partial model for all (non-floundering) queries, preserves all their three-valued stable models, and has polynomial data complexity for function-free programs.

However, through the introduction of the NEGATIVE SUSPENSION transformation, $SLG_{\text{RD}}$ significantly restricts the space of possible SLG derivations. SLG allows an arbitrary strategy for scheduling transformations to apply. This flexibility, though theoretically appealing, may result in inefficient evaluation due to unnecessary use of the DELAYING (and as a result of the SIMPLIFICATION and the ANSWER COMPLETION) transformations. Unnecessarily DELAYING ground negative literals that the computation rule stumbles on, opens up the search space by expanding subgoals that may not be relevant to the query. Contrary to SLG evaluation, $SLG_{\text{RD}}$ suspends that computation path, and later applies the DELAYING transformation to these negative literals only if their subgoals have no left-to-right proof, i.e. only if their evaluation results in a flummoxed ($SLG_{\text{strat}}$) system. Specifically, after the mechanics of $SLG_{\text{RD}}$ are defined, the following theorem can be proven using Theorem 4.3.

**Theorem 5.1.** *Let $\neg S$ be a selected literal in an $SLG_{\text{RD}}$ evaluation of a ground normal logic program $P$ under a left-to-right computation rule. Then the* DELAYING *transformation is applied to $\neg S$ if and only if $S$ belongs to the ultimate left-to-right dynamic stratum of $P$.*

This theorem can be seen as addressing the question of relevance in $SLG_{\text{RD}}$, our restriction of SLG resolution. In principle, if a rule instance is used in a left-to-right well-founded evaluation of a query, a literal of that rule should be relevant only if it is preceded by a sequence of literals that are true or undefined in the well-founded model. As discussed in the previous section, this criterion for relevance is very strong and appears unobtainable by practical strategies that mainly use a fixed computation rule. Theorem 5.1 thus states our approximation to this ideal measure of relevance by using the notion of the ultimate left-to-right dynamic stratum. The discussion of the previous section argues that this approximation is tight – possibly the tightest that can be achieved using a fixed computation rule.

[5] We do not repeat the definitions of these SLG transformations here; they can be included as presented in [5].

## 6. Applicability of results

The results of Section 4 show that $SLG_{\text{strat}}$ can form the basis of an engine to evaluate LRD programs, while those of the previous section indicate that $SLG_{\text{strat}}$ can also be used as a basis to restrict search in engines that evaluate the well-founded semantics. In fact, $SLG_{\text{strat}}$ has been implemented in the SLG-WAM of XSB [22], and has been extended to $SLG_{\text{RD}}$ described in Section 5. A detailed description of this implementation effort is given in [25, 19, 8, 26], and $SLG_{\text{RD}}$ forms the basis of version 1.8 of XSB. As mentioned in the introduction, the implementation of $SLG_{\text{RD}}$ has proven useful in practice. [10] presents a model checker based on the full *alternating modal $\mu$-calculus*, which has been implemented using $SLG_{\text{RD}}$, as is the commercial psychiatric diagnosis system mentioned in the introduction.

Despite the fact that our results have been presented using (variants of) SLG resolution, we believe that the underlying ideas are potentially applicable to deductive database systems whose engines have radically different designs from XSB. For example, CORAL [21] evaluates modularly stratified programs using Ordered Search [20]. Described simply, this technique dynamically builds a tree of dependencies and restricts magic evaluation to magic facts that are in a set of mutually dependent subqueries. Both Ordered Search and its extension Well-Founded Ordered Search as originally formulated do not contain a mechanism like early completion; a mechanism that usually eliminates many spurious negative cycles, thereby reducing the need for dynamic search. It appears possible to extend these strategies to include early completion by checking whether a given magic fact is subsumed by an answer fact. The resulting systems may have the same relevance properties for LRD programs as $SLG_{\text{strat}}$.

## Appendix A. Left-to-right weak stratification

In order to prove Theorems 2.3 and 2.4, we first present the notion of left-to-right weak stratification (LRW-stratification) based on the notion of weak stratification of [15].

**Definition A.1** (*Atom dependency graph*, *Przymusinska and Przymusinski* [15]). For a ground program $P$, the vertices of the *atom dependency graph* $G_P$ are the atoms appearing in $P$. The edges of $G_P$ are directed and labeled either *positive or negative*. For every ground instantiation of a rule

$$H \leftarrow B_1, \ldots, B_n$$

in $P$, there are $n$ edges in $G_P$. For $i = 1, \ldots, n$, if $B_i$ is an atom then there is a positive edge from $B_i$ to $H$ in $G_P$; if $B_i$ is a negated atom, say $\neg C_i$, then there is a negative edge from $C_i$ to $H$ in $G_P$.

*Atom dependency relations* $\leqslant_P$ and $<_P$ between ground atoms of $P$ are defined as:

- $B \leqslant_P A$ iff there is a directed path from $B$ to $A$ in $G_P$.
- $B <_P A$ iff there is a directed path from $B$ to $A$ in $G_P$ passing through a negative edge.

Note that in the above definition there can be both a positive and a negative edge between two ground atoms. The underlying idea behind weak stratification is to introduce the notion of the *components* of the atom dependency graph $G_P$ and to use them for the *vertices* in the usual definitions of predicate local stratification.

**Definition A.2** (*Component*, *Przymusinska and Przymusinski* [15]). For a program $P$, let $\sim_P$ be the equivalence relation between ground atoms defined as follows:

$$A \sim_P B \textit{ if and only if } (A = B) \vee (A <_P B \wedge B <_P A)$$

We shall refer to the equivalence classes introduced by $\sim_P$ as *components* of $G_P$. A component of $G_P$ is *trivial* iff it consists of only one element, say $A$, and $A \not<_P A$.

The relation $\sim_P$ denotes either equality of mutual negative dependence. In other words, according to the above definition, two distinct ground atoms $A$ and $B$ are equivalent if they are related by *mutual negative recursion*, i.e. recursion passing through negative literals.

Let $C_1$ and $C_2$ be two components of $G_P$. We define $C_1 \prec_P C_2$ if and only if $C_1 \neq C_2$ and there exist ground atoms $A_1 \in C_1$ and $A_2 \in C_2$ such that $A_1 <_P A_2$. A component $C$ is called *minimal* if there is no component $C'$ such that $C' \prec_P C$. For a program $P$, we define the *bottom stratum* $S(P)$ to be the union of all minimal components with respect to $\prec_P$. We define the *bottom layer* $L(P)$ of $P$ to be the set of rules from $P$ whose heads belong to $S(P)$. Herbrand models of $L(P)$ are identified with subsets $M \subseteq S(P)$ of the bottom stratum $S(P)$.

In order to define LRW-stratified programs, we include the notion of a rule prefix from [24].

**Definition A.3** (*Rule prefix*). A *rule prefix* is formed from a rule with $n$ subgoals in the body by deleting the rightmost $m$ subgoals, where $0 \leqslant m < n$.

**Definition A.4** (*Reduction of P modulo M*). Let $M \subseteq S(P)$ be a 3-valued interpretation of $P$. By the *reduction of P modulo M* we mean a new program $P/M$ obtained from $P$ by performing the following operations:

1. Remove from $P$ all rules that contain a *failing rule prefix*: for all $j$ $(1 \leqslant j \leqslant i-1)$, $L_j$ is true in $M$, and $L_i$ is false in $M$.
2. Remove from $P$ all rules whose head belongs to $M$.
3. Remove from all the remaining rules those literals that are members of $M$.
4. Finally, from the program resulting from the above operations, delete all rules with nonempty bodies whose heads appear in the program as unit facts.

The above definition of reduction differs from that given in [15] only in the addition of the failing rule prefix condition in step 1. Given the modified reduction operator, construction of the LRW model is identical to that of the weakly stratified model.

**Definition A.5** (*Left-to-right weak stratification*, *Przymusinska and Przymusinski* [15]). Let $P$ be a program; let $P_0 = P$; and let $M_0 = \emptyset$. Suppose that $\alpha > 0$ is a countable ordinal such that programs $P_\delta$ and 3-valued interpretations $M_\delta$ have already been defined for all $\delta < \alpha$. Let

$$N_\alpha = \bigcup_{0 \leqslant \delta < \alpha} M_\delta$$

and let $P_\alpha = P/N_\alpha$ be the reduction of $P$ with respect to $N_\alpha$.

- If the program $P_\alpha$ is empty, then the construction stops and the program $P$ is *left-to-right weakly stratified.* Then, $N_\alpha$ is the two valued model of $P$, and any ground atom that appears neither positively nor negatively in $N_\alpha$ is assumed false.
- Otherwise, if the bottom stratum $S(P_\alpha)$ of $P_\alpha$ is empty or if it contains a nontrivial component, then the construction stops. In such a case, $P$ is not LRW-stratified.
- Otherwise, the 3-valued interpretation $M_\alpha$ is defined as the least (2-valued) model (restricted to the literals whose atoms are in $S(P_\alpha)$ of the bottom layer $L(P_\alpha)$ of $P_\alpha$, and the construction continues.

**Theorem 2.3.** *In the stratification hierarchy of Fig.* 1, *there is a path from a class $\mathscr{C}_1$ of programs to a class $\mathscr{C}_2$ iff every program in $\mathscr{C}_1$ is a program in $\mathscr{C}_2$ and there exists at least one program in $\mathscr{C}_2$ that is not contained in $\mathscr{C}_1$.*

**Proof.** The relation of dynamically stratified programs to weakly stratified programs follows directly from the fact, proven in [17], that a program is dynamically stratified iff it has a well-founded model. Ref. [23] presents the relation of (LR) modularly stratified programs to weakly stratified programs and to locally stratified programs. Finally, [16] presents the relation of locally stratified programs to the stratified programs of [16]. It thus remains to show that the placement of LRD-stratified and LRW-stratified programs within the hierarchy is correct.

- To see that the set of LRD-stratified programs is properly contained within the set of dynamically stratified programs, consider that the operator $\mathscr{I}$ (Definition 2.3) will be monotonic over 3-valued interpretations regardless of whether the underlying definition of $\mathscr{F}_I$ uses a failing prefix or not. Proof of containment is thus a straightforward induction on this iterated fixed point operator. Example 1.2 shows that this containment is proper.
- Next, we show that the set of LRW-stratified programs is properly contained within the set of LRD-stratified programs. We begin by showing, for any 3-valued interpretation $I$, the 3-valued interpretation of $P/I$ produced by the construction of Definition A.4 is contained in $\langle T_I, F_I \rangle$ (Definition 2.3). Let $\langle TW_I, FW_I \rangle$ represent the least model of $P/I$ restricted to the atoms in the bottom layer of $S(P/I)$. Note that condition 2

of Definition A.4 and condition 3 of Definition A.5 together ensure that no element of $I$ will be in $TW_I$ or in $FW_I$. Using this fact, we next need to show that $TW_I \subseteq T_I$. From Definition A.5 it can be seen that $TW_I$ is the least fixed point of those rules in $P/I$ which contain no negative literals. By Definition 2.1, $TW_I$ is thus contained in the least fixed point produced by $T_I$. The elements of $FW_I$ are those atoms of $S(P/I)$ that are not in the least fixed point of $L(P/I)$. These will be contained in the greatest fixed point produced by $F_I$. Because both $\mathscr{I}(I)$ (Definition 2.3) and the least model of $P/I$ are monotonic with respect to $I$, the overall proof of containment then follows by a routine induction.

A simple example of a program that is LRD-stratifed but not LRW-stratified is the following:

```
p ← q, ¬r.
q ← r, ¬p.
r ← p, ¬q.
```

- Showing that every (left-to-right) modularly stratified program is LRW-stratified, is a straightforward adaptation of the proof in [24] that every modularly stratified program is also weakly stratified. The meta-interpretation example of Section 2.3 shows that the inclusion is proper. □

As shown in Fig. 1, the classes of LRD-stratified programs and Weakly Stratified programs are incompatible and we now briefly substantiate this claim: As mentioned, the program of Example 1.2 is LRD-stratified but not weakly stratified. On the other hand, the program of Example 1.1 is weakly stratified, but not LRD-stratified.

To prove Theorem 2.4, we first give the precise definition of the meta-interpreter transformation, which was informally introduced in Section 2.3.

**Definition A.6.** Given an input program $P$, the *meta-interpretation transformation* produces a program $P_{\text{meta}}$ such that:

- $P_{\text{meta}}$ contains a fact `clause(`*Head*`,`*Body*`)` iff $P$ contains a rule *Head* ← *Body*′, where *Body* is such that every occurrence in *Body*′ of the operator ¬/1 is replaced by the function symbol `not/1`.
- $P_{\text{meta}}$ contains a fact `clause(`*Head*`,true)` iff $P$ contains a rule *Head* ←.
- $P_{\text{meta}}$ contains the additional rules:

```
demo(true).
demo(A,B) ← demo(A), demo(B).
demo(not A) ← ¬demo(A).
demo(A) ← clause(A,B), demo(B).
```

- $P_{\text{meta}}$ contains no other rules.

**Theorem 2.4.** *Given a program $P$, let $P_{\text{meta}}$ be the program obtained by the transformation of Definition* A.6. *Then*

1. *if the ground instantiation of $P$ is (LR) Modularly Stratified, the ground instantiation of $P_{\text{meta}}$ is LRW-stratified.*
2. *if the ground instantiation of $P$ is LRD-stratified, the ground instantiation of $P_{\text{meta}}$ is LRD-stratified.*

**Proof** (*sketch*)**.** (1) Let $C$ be a LR-modularly stratified recursive component (see [24]) in $P$ and consider atoms of the form: `demo(Term1)` where `Term1` is in $C$. Then all such atoms will lie in the same (weak) stratum, and will produce a 3-valued interpretation using the construction of Definition A.5. $P_{\text{meta}}$ will also contain cycles in its atom dependency graph formed by the ground instantiation of the rule `demo(A)` $\leftarrow$ `clause(A,B),demo(B)`.

However, these nontrivial components will be broken during the iterated reductions that comprise the construction of the LRW-stratified model. To see this, consider that a fact `clause(A,B)` will be true in the ground instantiation of $P_{\text{meta}}$ iff $A \leftarrow B$ is a rule in the ground instantiation of $P$. Thus the first reduction $P_{\text{meta}}$ will make `clause/2` true if and only if it represents a rule in $P$ and false otherwise. It is then easy to see that the iteration will continue until a LRW-stratified model is produced for $P_{\text{meta}}$.

(2) The proof that $P_{\text{meta}}$ is LRD-stratified is a straightforward induction relying on the fact that if a rule $A \leftarrow B$ has a failing prefix in $P$, then in $P_{\text{meta}}$, a subgoal for `demo(A)` will depend on a rule $R$ that evaluates literals of $B$ such that $R$ has a failing prefix. □

## Appendix B. Lengthy proofs

### B.1. Proofs from Section 3

**Proposition 3.1.** *Let $P$ be a countable program, $R$ be an arbitrary but fixed computation rule, and $Q$ be an atomic query. Then*

(a) *there exists a countable ordinal $\lambda$ such that for any system $\mathscr{S}$ in any $SLG_{\text{strat}}$ derivation for $Q$, the length of the X-sequence of each subgoal in $\mathscr{S}$ is bounded by $\lambda$;*
(b) *every $SLG_{\text{strat}}$ derivation of $Q$ is a monotonically increasing sequence of systems with respect to $\sqsubseteq$; and*
(c) *there exists some $SLG_{\text{strat}}$ derivation for $Q, \mathscr{S}_0, \mathscr{S}_1, \ldots, \mathscr{S}_\alpha$, for some ordinal $\alpha$ such that $\mathscr{S}_\alpha$ is a final system.*

**Proof.** (a) Let $\Pi_P$ be the maximum number of literals in the body of a rule in $P$. Let $\mathscr{S}$ be any system in an $SLG_{\text{strat}}$ derivation for $Q$, and $(A : \rho)$ be any pair in $\mathscr{S}$, where $A$ is a subgoal and $\rho$ is its X-sequence.

- If $P$ has finite number of rules with variables, the length of $\rho$ is bounded based upon the following observations:
  - When a subgoal $A$ is added to a system, its initial X-sequence contains the X-rules that are obtained by resolving $A \leftarrow A$ on $A$ in the body with rules in $P$.

The number of literals in the body of each X-rule is bounded by $\Pi_P$. Since $P$ is finite, the initial X-sequence of $A$ is finite.

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

(i) If $e$ is *disposed*$\langle\alpha\rangle$, then $G$ is diposed and can be disposed at most once by the construction of an $SLG_{\text{strat}}$ derivation;

(ii) If $e$ is *suspended*$\langle\alpha\rangle$, then $G$ is suspended on its selected literal. Since the NEGATION SUSPENSION transformation requires that $G$ be an active X-rule and that the subgoal of the selected literal be evolving, $G$ can be suspended at most once on each of its negative literals;

(iii) If $e$ is *unsuspended*$\langle\alpha\rangle$, then $G$ is marked as unsuspended and, since each atom is completed at most once (through the COMPLETION transformation), a suspended X-rule can be unsuspended at most once for each of its negative literals;

(iv) If $e$ is $G'\langle\alpha\rangle$, where $G'$ is an X-rule, then $G$ is disposed and replaced by $G'$ (using the NEGATIVE RETURN transformation);

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

Therefore each X-rule $G$ of a subgoal $A$ corresponding to an ordinal $\alpha$, the number of X-rules $G'$ that can be obtained direcly from $G$ is bounded by the number of distinct atoms, which is countable. In both (iv) and (v), the number of literals in the body of $G'$ is one fewer than that in $G$. For any chain of X-rules, $G_0, G_1, \ldots, G_l$, where each $G_{j+1}$ $(0 \leqslant j < l)$ is obtained from $G_j$ by some transformation, $l \leqslant 3\Pi_P$; the longest chain is created when each of the literals in the body of $G_0$ is a negative literal that gets suspended, unsuspended, and deleted through a NEGATIVE RETURN. Thus, there exists some countable ordinal $\lambda$ by which the length of the X-sequence $\rho$ of $A$ in $\mathscr{S}$ is bounded. The ordinal $\lambda$ depends upon only the program $P$ and the language $\mathscr{LF}$ that is countable, and is applicable to the X-sequence $\rho$ of any subgoal $A$ in any system $\mathscr{S}$ in an arbitrary $SLG_{\text{strat}}$ derivation for $Q$.

• If $P$ is a ground program, the argument is similar, but with the difference that, while an initial X-sequence for a subgoal $A$ may be infinite, no $SLG_{\text{strat}}$ transformation for an X-element will add more than a finite number of new X-elements to an X-sequence. Once again, the X-sequence for $A$ will be bounded by a countable ordinal.

The proofs of (b) and (c) are identical to those for SLG (cf. Theorem 4.1 of [5]), for both programs with a finite number of rules, and for ground programs. □

**Theorem 3.3.** *Let $P$ be a program, $R$ be an arbitrary but fixed computation rule, $Q$ be an atomic query, and $\mathscr{S}$ be a flummoxed system in an arbitrary $SLG_{\text{strat}}$ derivation for $Q$. Then every noncompleted subgoal in $\mathscr{S}$ that depends negatively on a subgoal in $\mathscr{S}$ with a suspended n negative literal.*

**Proof.** Since $\mathscr{S}$ is flummoxed, by Definition 3.12 there exists a set $\Sigma$ of subgoals in $\mathscr{S}$ that is not completed in $\mathscr{S}$. Furthermore, since $\mathscr{S}$ is a flummoxed system, no transformation is applicable to $\mathscr{S}$; in particular, the COMPLETION transformation is not applicable to any set of subgoals $\Sigma'$ in $\Sigma$. Therefore, no $\Sigma' \subseteq \Sigma$ is completely evaluated. Consider an arbitrary subgoal $A \in \Sigma$. $A$ does not succeed and must contain at least one X-rule that has a selected literal and is not disposed, i.e. it contains at least one *suspended* or *active* X-rule. Let $\Sigma_A$ be the smallest set of subgoals that are not completed in $\mathscr{S}$, such that $A \in \Sigma_A$ and such that for each subgoal $A'$ in $\Sigma_A$, the atom of each selected literal of each nondisposed clause for $A'$ is also in $\Sigma_A$. First, $\Sigma_A \subseteq \Sigma$, since if not, a clause would contain a selected literal whose subgoal was not in $\mathscr{S}$ in which case a NEW SUBGOAL operation would be applicable. Now, if no subgoal in $\Sigma_A$ contained a selected negative literal, $\Sigma_A$ would be completely evaluated. Thus some subgoal in $\Sigma_A$ must depend negatively on another subgoal in $\Sigma_A$. Furthermore, $\Sigma_A$ was defined so that $A$ depends on every subgoal in $\Sigma_A$. Thus, $A$ depends negatively on at least one subgoal that is not completely evaluated, and since $A$ was chosen arbitrarily, the statment must hold for all noncompleted subgoals. □

## *B.2. Proofs from Section 4*

**Lemma 4.1.** *Let $P$ be a ground LRD-stratified program, $Q$ be a ground atomic query, and $\mathscr{E}$ be an $SLG_{\text{strat}}$ derivation for $Q$ under a left-to-right computation rule. Then for any completely evaluated subgoal $A$ in a system $\mathscr{S}$ in $\mathscr{E}$,*

$$val_{I_s}(A) = val_{J_{IF(P)}}(A),$$

*that is, the truth value of the subgoal in $I_{\mathscr{S}}$ is the value of the subgoal in $M_P$.*

**Proof.** First, note that since $P$ and $Q$ are ground, the subgoals that are contained in systems of any $SLG_{\text{strat}}$ derivation for $Q$ are also ground. Our proof is by induction on the index on interpretation for states of $\mathscr{E} = \mathscr{S}_0, \mathscr{S}_1, \ldots, \mathscr{S}_\alpha$.

- For the base case, there are no completely evaluated subgoals, and the statement trivially holds.
- For $I_{\mathscr{S}_N}$ where $N$ is a successor ordinal, we assume the statement true for ordinals less than $N$, and consider each $SLG_{\text{strat}}$ transformation used to produce $N$. In fact, the only two $SLG_{\text{strat}}$ operation that differ from SLG are NEGATION SUSPENSION and COMPLETION, so that if it can be shown that these are informationally sound with respect to $J_{IF(P)}$, the theorem will hold when $N$ is a successor ordinal. THe case of NEGATION SUSPENSION follows easily. For the case of COMPLETION, the $SLG_{\text{strat}}$ COMPLETION operation differs from that of SLG in two respects. First, a completion

operation may be directly applicable to a successful subgoal, which, again does not affect $I_{\mathscr{S}_N}$. Second, a COMPLETION operation may unsuspend suspended X-rules. In this case, each of the unsuspended X-rules will become active (Definition 3.6), and a NEGATIVE RETURN operation applicable to the successful or failed literal. By the induction hypothesis and the correctness of SLG, this future NEGATIVE RETURN operation will maintain correctness of the induced interpretation.

- We next consider the case where $I_{\mathscr{S}_N}$ where $N$ is a limit ordinal. From Definition 3.10 it can be seen that for a limit ordinal $N, I_{\mathscr{S}_N} = \bigcup I_{\mathscr{S}_M}, M < N$. Given this observation, the case in which $N$ is a limit ordinal follows directly from the induction hypothesis. □

**Lemma 4.2.** *Let P be a ground LRD-stratified program, Q be a ground atomic query, and* $\mathscr{S}_0, \mathscr{S}_1, \ldots, \mathscr{S}_\alpha$ *be any* SLG$_{\text{strat}}$ *derivation for Q under a left-to-right computation rule, such that* $\mathscr{S}_\alpha$ *is a final system. Then* $\mathscr{S}_\alpha$ *is completed.*

**Proof** (*sketch*)**.** Suppose $\mathscr{S}_\alpha$ is not completed. Then because $P$ is ground, by Lemma 3.2, $\mathscr{S}_\alpha$ must be flummoxed. Let $U_S$ be the set of uncompleted subgoals in $\mathscr{S}_\alpha$, and let $N$ be the minimal dynamic stratum of any element in $U_{\mathscr{S}_\alpha}$. The proof used induction on $N$ to show that the minimal dynamic stratum for $U_{\mathscr{S}_\alpha}$ cannot occur for any $N$, which in turn will show that $\mathscr{S}_\alpha$ cannot be a flummoxed system.

- Base case, $N = 1$. $F_{J_0}$ consists of atom $H$ such that for all rules $H \leftarrow Body$, *Body* contains a failing prefix consisting solely of positive literals. Thus, there are no negative dependencies encountered in evaluating $H$. If $H$ were flummoxed, Theorem 3.3, would be contradicted. A similar argument can be made for elements of $T_{J_0}$.
- $N$ is a successor ordinal. Assume that the statement holds for all dynamic strata less than $N$ to show that the statements also holds at stratum $N$. We prove this case by contradiction: assume *Subgoal* is an element $U_{\mathscr{S}}$ whose dynamic stratum is $N$.
  - We first show that *Subgoal* cannot be an element of $T_{J_N}$. Consider that by Defintion 2.5 if *Subgoal* has dynamic stratum $N$, then it has a derivation consisting of subgoals with strata $N - 1$ or less, possibly along with subgoals whose dynamic stratum is $N$ and that are in $T_{J_N}$. By the induction hypothesis any subgoals having dynamic strata less than $N$ that *Subgoal* depends on must be completed, and by Lemma 4.1 they must be completed correctly. This leaves only subgoals in $T_{J_N}$ that *Subgoal* may depend on. At this point let $U_{\mathscr{S}}^{T_{J_N}}$ represent those elements of $U_{\mathscr{S}}$ which are in $T_{J_N}$. Then one of these elements, $S_{\text{prim}}$, has a derivation which depends on no other element in $U_{\mathscr{S}}^{T_{J_N}}$ for it to be in $T_{J_N}$ (otherwise the set of subgoals in $U_{\mathscr{S}}^{T_{J_N}}$ would be unfounded, and *Subgoal* would not be in $T_{J_N}$). However in this case, $S_{\text{prim}}$ is completely evaluated (via the early completion condition of Definition 3.9). If the element were completely evaluated, it could not be in $U_{\mathscr{S}}$. Using Theorem 3.3, it is straightforward to see that $U_{\mathscr{S}}^{T_{J_N}}$ must be empty and as a result, *Subgoal* cannot be in $T_{J_N}$.
  - Next, we show that no element of $U_{\mathscr{S}}$ can be in $F_{J_N}$. Consider that each element in $U_{\mathscr{S}}$ must have in its dependency graph a subgoal with a suspended X-rule –

by Theorem 3.3. Consider now an element *Subgoal* with associated suspended X-rule $C$. *Subgoal* is suspended on another goal $Subgoal_{\mathrm{neg}}$ in $U_{\mathscr{S}}$ so that there is a negative dependency from *Subgoal* to $Subgoal_{\mathrm{neg}}$ in $U_{\mathscr{S}}$, and $\neg Subgoal_{\mathrm{neg}}$ is the selected literal for $C$. *Subgoal* cannot be in $F_{J_N}^{\downarrow 1}$. To see this, note that Defintion 2.1 requires $Subgoal_{\mathrm{neg}}$ either to be false in $J_N$ or to be in $F_{J_N}^{\downarrow 1}$. The first case is impossible since if $Subgoal_{\mathrm{neg}}$ were in a lower stratum, it would be (correctly) completed due to the induction hypothesis. The second case cannot hold as is trivial since a negative literal cannot be in $F_{J_N}^{\downarrow 1}$. Thus no element of $U_{\mathscr{S}}$ can be in $F_{J_N}$.

- $N$ is a limit ordinal. Given the assumption that the statement holds for all dynamic strata less than $N$, this step is trivial since for limit ordinals, $J_N$ is directly defined as the interpretation union of all $J_M$ such that $M < N$. □

## References

[1] J.J. Alferes, C.V. Damásio, L.M. Pereira, A logic programming system for non-monotonic reasoning, J. Automat. Reasoning 14(1) (1995) 93–147.

[2] N. Bidoit, C. Froidevaux, Negation by default and unstratifiable logic programs, Theoret. Comput. Sci. 78(1) (1991) 85–112.

[3] R. Bol, L. Degerstedt, Tabulated resolution for the well founded semantics, J. Logic Programming 34(2) (1998) 67–110.

[4] W. Chen, M. Kifer, D.S. Warren, HiLog: a foundation for higher-order logic programming, J. Logic Programming 15(3) (1993) 187–230.

[5] W. Chen, D.S. Warren, Tabled evaluation with delaying for general logic programs, J. ACM 43(1) (1996) 20–74.

[6] J. Dix, Classifying semantics of logic programs, in: A. Nerode, W. Marek, V.S. Subrahmanian (Eds.), Logic Programming and Mon-Monotonic Reasoning, Proc. 1st Internat. Workshop, The MIT Press, Washington D.C., July 1991, pp. 166–180.

[7] E.A. Emerson, C.-L. Lei, Efficient model checking in fragments of the propositional mu-calculus, in: Proc. IEEE Symp. on Logic in Computer Science, IEEE Computer Society, Cambridge, MA, June 1986, pp. 267–278.

[8] J. Freire, T. Swift, D.S. Warren, Beyond depth-first strategies: Improving tabled logic programs through alternative scheduling, J. Funct. Logic Programming 1998(3) (1998).

[9] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: R.A. Kowalski, K.A. Bowen (Eds.), Proc. 5th Internat. Conf. and Symp. on Logic Programming, The MIT Press, Seattle, August 1988, pp. 1081–1086.

[10] X. Liu, C.R. Ramakrishnan, S.A. Smolka, Fully local and efficient evaluation of alternating fixed points, in: B. Steffen (Ed.), Proc. TACAS-98: Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science, Vol. 1384, Springer, Lisbon, Portugal, March/April 1998, pp. 5–19.

[11] J.W. Lloyd, Foundations of Logic Programming, 2nd ed., Springer, Berlin, 1987.

[12] Medicine Rules, Inc. Diagnostica, Technical Report, 1998. http://medicinerules.com.

[13] S. Morishita, An extension of Van Gelder's alternating fixpoint to magic programs, J. Comput. System Sci. 52(3) (1996) 506–521.

[14] I. Niemelä, P. Simons, Smodels – an implementation of the stable model and well-founded semantics for normal LP, in: J. Dix, U. Furbach, A. Nerode (Eds.), Proc. 4th Internat. Conf. on Logic Programming and Non-Monotonic Reasoning, Lecture Notes in Artificial Intelligence, Vol. 1265, Springer, Dagstuhl Castle, Germany, July 1997, pp. 420–429.

[15] H. Przymusinska, T.C. Przymusinski, Weakly stratified logic programs, Fund. Inform. 13(1) (1990) 51–65.

[16] T.C. Przymusinski, On the declarative semantics of deductive databases and logic programs, in: J. Minker (Ed.), Foundations of Deductive Databases and Logic Programming, Morgan-Kaufmann, Los Altos, 1988, pp. 193–216.

[17] T.C. Przymusinski, Every logic program has a natural stratification and an iterated least fixed point model, in: Proc. 8th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, ACM Press, Philadelphia, Pennsylvania, March 1989, pp. 11–21.

[18] Y.S. Ramakrishna, C.R. Ramakrishnan, I.V. Ramakrishnan, S.A. Smolka, T. Swift, D.S. Warren, Efficient model checking using tabled resolution, in: O. Grumberg (Ed.), Proc. 9th Internat. Conf. on Computer-Aided Verification, Lecture Notes in Computer Science, Vol. 1254, Springer, Haifa, Israel, July 1997, pp. 143–154.

[19] I.V. Ramakrishnan, P. Rao, K. Sagonas, T. Swift, D.S. Warren, Efficient access mechanisms for tabled logic programs, J. Logic Programming 38(1) (1999) 31–54.

[20] R. Ramakrishnan, D. Srivastava, S. Sudarshan, Controlling the search in bottom-up evaluation, in: K. Apt (Ed.), Proc. Joint Internat. Conf. and Symposium on Logic Programming, The MIT Press, Washington DC, October 1992, pp. 273–287.

[21] R. Ramakrishnan, D. Srivastava, S. Sudarshan, CORAL – control, relations, and logic, in: Proc. 18th Conf. on Very Large Data Bases, Morgan-Kaufmann, Vancouver, Canada, August 1992, pp. 238–249.

[22] P. Rao, K. Sagonas, T. Swift, D.S. Warren, J. Freire, XSB: a system for efficiently computing WFS, in: J. Dix, U. Furbach, A. Nerode (Eds.), Proc. 4th Internat. Conf. on Logic Programming and Non-Monotonic Reasoning, Lecture Notes in Artificial Intelligence, Vol. 1265, Springer, Dagstuhl Castle, Germany, July 1997, pp. 430–440.

[23] K.A. Ross, A procedural semantics for well-founded negation in logic programs, J. Logic Programming 13(1) (1992) 1–22.

[24] K.A. Ross, Modular stratification and magic sets for Datalog programs with negation, J. ACM 41(6) (1994) 1216–1266.

[25] K. Sagonas, T. Swift, An abstract machine for tabled execution of fixed-order stratified logic programs, ACM Trans. Programming Languages Systems 20(3) (1998) 586–634.

[26] K. Sagonas, T. Swift, D.S. Warren, An abstract machine for computing the well-founded semantics, in: M. Maher (Ed.), Proc. Joint Internat. Conf. and Symp. on Logic Programming, The MIT Press, Bonn, Germany, September 1996, pp. 274–288.

[27] P.J. Stuckey, S. Sudarshan, Well-founded ordered search: goal-directed bottom-up evaluation of well-founded models, J. Logic Programming 32(3) (1997) 171–206.

[28] A. Van Gelder, The alternating fixpoint of logic programs with negation, J. Comput. System Sci. 47(1) (1993) 185–221.

[29] A. Van Gelder, K.A. Ross, J.S. Schlipf, The well-founded semantics for general logic programs, J. ACM 38(3) (1991) 620–650.