Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence

Radial Restraint: A Semantically Clean Approach to Bounded Rationality for Logic Programs

Benjamin Grosof
Benjamin Grosof & Associates, LLC
Mercer Island, Washington, USA

Terrance Swift
CENTRIA
Universidade Nova de Lisboa, Lisboa Portugal

Abstract

Declarative logic programs (LP) based on the well-founded semantics (WFS) are widely used for knowledge representation (KR). Logical functions are desirable expressively in KR, but when present make LP inferencing become undecidable. In this paper, we present *radial restraint*: a novel approach to bounded rationality in LP. Radial restraint is parameterized by a norm that measures the syntactic complexity of a term, along with an abstraction function based on that norm. When a term exceeds a bound for the norm, the term is assigned the WFS's third truth-value of *undefined*. If the norm is finitary, radial restraint guarantees finiteness of models and decidability of inferencing, even when logical functions are present. It further guarantees soundness, even when non-monotonicity is present. We give a fixed-point semantics for *radially restrained well-founded models* which soundly approximate well-founded models. We also show how to perform correct inferencing relative to such models, via SLG_{ABS}, an extension of tabled SLG resolution that uses norm-based abstraction functions. Finally we discuss how SLG_{ABS} is implemented in the engine of XSB Prolog, and scales to knowledge bases with more than 10^8 rules and facts.

Introduction

Declarative logic programs (LP) based on the well-founded semantics (WFS) are widely used for knowledge representation (KR), e.g., in databases, business rules, and semantic web. They represent logical non-monotonicity, and offer much better scalability than answer-set programs (ASP) or first-order logic (FOL). Logical functions are desirable expressively in KR overall, and in particular in Rulelog, the logical extension of LP that has been developed and employed in the SILK project and system (SILK 2013). Functions are needed there to support three expressive features: hilog (Chen, Kifer, and Warren 1993); defeasibility via argumentation theories (Wan et al. 2009); and existentials in FOL-like *omniform* rules. These three features are in turn used for reasoning about causal processes/actions, qualitative reasoning, and text-based knowledge acquisition (KA) — in SILK's pilot application domain of cell biology at the first year college level. However, functions pose a fundamental computational complexity challenge in LP and thus Rulelog/SILK, as functions do also in FOL and ASP. LP inferencing is tractable (worst-case quadratic) in the size of the ground-instantiated rulebase. But when functions are present, the Herbrand universe is infinite, and LP inferencing is undecidable. The LP model might be infinite, and a single LP query might have an infinite set of answers. Today's commercially dominant systems for databases, business rules, and semantic web (e.g., SQL, SPARQL, production rules, and event-condition-action rules) avoid use of functions. We are thus motivated to seek a way to allow functions in LP yet to mitigate their impact on scalability — including, as a first step, to ensure finiteness of models and decidability of inferencing. Our approach builds on the idea of bounded rationality.

While AI bounded rationality research has largely focused on decision-theoretic optimization (e.g., (Russell and Subramanian 1995; Anderson and Oates 2007)), a strand has focused on limiting reasoning via deduction principles that derive some beliefs explicitly but leave others implicit (Konolige 1983; Levesque 1984; Fisher and Ghidini 1999; Grant, Kraus, and Perlis 2000; Fisher et al. 2007). To date, however, this strand has lacked much practical impact. For resource-limiting logic programming (i.e., LP), the main approach that has emerged in practice is to set (manually or heuristically) an inferencing engine parameter — for instance, a timeout or a term-depth bound in Prolog — and to treat as false any atom that is not inferred before the parameter bound is exceeded. However, incompleteness about an atom A can lead to unsoundness if another atom A' depends negatively on A. In addition, the results of such inferencing depend on the implementation code or session. Radial restraint overcomes both these shortcomings. It introduces (to our knowledge) the use of the truth value *undefined* to represent implicit deductions that have not been made explicit.

Our starting point is recent work on termination properties of logic programs with negation (default negation, a.k.a. negation-as-failure, a.k.a. weak negation). When tabled evaluation is extended with *subgoal abstraction*, first introduced in (Tamaki and Sato 1986), tabling can ensure termination to queries to safe normal programs that are *strongly bounded term size (SBTS)* (Riguzzi and Swift 2013b; 2013a). Such programs have well-founded models (van Gelder, Ross, and Schlipf 1991) that are representable via finite sets of true and undefined ground atoms, and have

been shown to properly include the finitely ground programs of (Calimeri et al. 2008), a class motivated by the needs of ASP grounders (cf. (Alviano, Faber, and Leone 2010)). Although these results are powerful, they do have drawbacks. It is not decidable whether a program is SBTS. In addition, while SBTS-programs are Turing-complete (shown for finitely ground programs in (Calimeri et al. 2008)), some natural programs are not SBTS: such as those that contain a predicate to determine list membership. From a theoretical level, this weakness can be addressed by defining program classes that terminate for various types of queries (cf. e.g., (Bonatti 2004)). However, membership in such classes is again not decidable.

The tabled evaluation method of (Riguzzi and Swift 2013a) is complete for SBTS-programs and queries. Building on this evaluation method, and making use of the *undefined* truth value as discussed above, we ensure that both the evaluations and the (sub-)models they produce are finite. We introduce aspects of the approach through the following example. to motivate the formalism that follows.

Example 1 *Consider the program P_{inf}:*

p(s(X)) ← p(X).
p(0).
q(0).

P_{inf} is not SBTS as the sets both of true atoms of its well-founded model, $true(WFM^{P_{inf}})$, and the false atoms, $false(WFM^{P_{inf}})$, are infinite. However, an approximation of the answers to p(X) *can be made by restraining inferencing. For instance, if a depth norm were used with a level of 4, then the answers* p(0)*,* p(s(0))*, and* p(s(s(0))) *would be derived. However the answer* p(s(s(s(0)))) *would be* abstracted *to* p(s(s(s(X)))) *and all ground atoms unifying with this answer would be assigned the truth value of* undefined*. This infinite set of answers is represented by the sentence* ∀X. p(s(s(s(X)))). *By allowing the set of atoms in $WFM^{P_{inf}}$ whose truth value is undefined (*undef*($WFM^{P_{inf}}$)) to be represented by such sentences in addition to atoms, a finite representation of the* radially restrained model *is constructed. In addition, because the default negation of any undefined atom is itself undefined in the well-founded semantics, the answer abstraction preserves the soundness of negation. For instance the literal* not p(s(s(s(s(0))))) *is also assigned undefined.*

Intuitively, atoms that are true or false in the well-founded model of a program remain true or false in the radially restrained model as long as they don't exceed a bound defined by a norm on atoms. Those atoms that exceed the bound are abstracted, and have the truth value *undefined*. In this way, radially restrained models are sound approximations to the well-founded model. Further, because variables in the abstraction are regarded as universally quantified, the resulting model can be represented using finite sets of true and undefined atoms.

This paper thus explores radially restrained models along with their efficient query evaluation. Specifically,

- We define the radially restrained well-founded model of a normal program P as parameterized by an abstraction function $abs(\cdot)$. We show that such a model soundly approximates the well-founded model of P, and that if $abs(\cdot)$ is replaced by a weaker abstraction, the approximation of the radially restrained model becomes tighter.
- By extending SLG resolution with subgoal abstraction (Riguzzi and Swift 2013a) to incorporate an abstraction function for answers, we introduce SLG$_{\mathsf{ABS}}$, which correctly evaluates queries with respect to radially restrained models. Given a finitary abstraction function, SLG$_{\mathsf{ABS}}$ terminates with an asymptotic complexity that is equal to the best complexity that is known.
- Finally, based on the the SLG-WAM of XSB (Swift and Warren 2012), we describe an implementation of SLG$_{\mathsf{ABS}}$ that is declarative, efficient and scalable.

Background

We assume a general knowledge of logic programming terminology, including tabled resolution and the well-founded semantics. In addition we make use of the following terminology and assumptions.

Throughout this paper we restrict our attention to normal programs, and to queries and subgoals that are atoms. We also assume a fixed strategy for selecting literals in a clause: without loss of generality we assume the selection strategy is left-to-right. In accordance with this strategy, a normal rule has the form

$$r = A_0 \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n$$

where $A_0, \ldots, A_n$ are atoms. A program P is *safe* if each rule r in P is such that every variable in r occurs in a positive literal in the body of r. Our attention is also restricted to three-valued (partial) interpretations and models, such as the well-founded model. Each such interpretation is represented as a pair of true and false atoms: $\langle true(\mathcal{I}); false(\mathcal{I})\rangle$. For two interpretations, $\mathcal{I}$ and $\mathcal{J}$, $\mathcal{I} \subseteq \mathcal{J}$ iff $true(\mathcal{I}) \subseteq true(\mathcal{J})$ and $false(\mathcal{I}) \subseteq false(\mathcal{J})$. Alternatively, a three-value interpretation can be represented as a set of literals.

Symbols within a term may be represented through *positions* which are members of the set Π. A *position* in a term is either the empty string Λ that reaches the root of the term, or the string $\pi.i$ that reaches the ith child of the term reached by π, where π is a position and i an integer. For a term t we denote the symbol at position π in t by t_π or alternatively by $t|_\pi$. For example, $p(a, f(X))_{2.1} = X$. We assume that a program P is defined over a language $\mathcal{L}$, containing a finite set $\mathcal{FN}$ of predicate and function symbols, and a countable set of variables from the set $\mathcal{V} \cup \hat{\mathcal{V}}$. Elements of the set $\mathcal{V}$ are referred to as *program variables*. Elements of the set $\hat{\mathcal{V}}$, called *position variables*, are of the form X_π, where π is a position. These variables are used when it is convenient to mark certain positions of interest in a term. The Herbrand Universe of $\mathcal{L}$ is denoted $\mathcal{H}_\mathcal{L}$, or as $\mathcal{H}_P$ if $\mathcal{L}$ consists of the predicate and function symbols in P; similarly the Herbrand Base is denoted as $\mathcal{B}_\mathcal{L}$ or as $\mathcal{B}_P$. Throughout the paper variant terms are considered to be equal.

Dynamic Stratification One of the most important formulations of stratification is that of *dynamic* stratification. (Przymusinski 1989) shows that a program has a 2-valued

well-founded model iff it is dynamically stratified, so that it is the weakest notion of stratification that is consistent with the well-founded semantics. As presented in (Przymusinski 1989), dynamic stratification computes strata via operators on interpretations of the form $\langle Tr; Fa\rangle$, where Tr and Fa are subsets of $\mathcal{H}_P$.

Definition 1 *For a normal program P, sets Tr and Fa of ground atoms and a 3-valued interpretation I (sometimes called a pre-interpretation):*

$True_I^P(Tr) = \{A | A$ *is not true in I; and there is a clause $B \leftarrow L_1, ..., L_n$ in P, a grounding substitution θ such that $A = B\theta$ and for every $1 \leq i \leq n$ either $L_i\theta$ is true in I, or $L_i\theta \in Tr\}$;*

$False_I^P(Fa) = \{A | A$ *is not false in I; and for every clause $B \leftarrow L_1, ..., L_n$ in P and grounding substitution θ such that $A = B\theta$ there is some i $(1 \leq i \leq n)$ such that $L_i\theta$ is false in I or $L_i\theta \in Fa\}$.*

(Przymusinski 1989) shows that $True_I^P$ and $False_I^P$ are both monotonic, and defines $\mathcal{TR}_I^P$ as the least fixed point of $True_I^P(\emptyset)$ and $\mathcal{FA}_I^P$ as the greatest fixed point of $False_I^P(\mathcal{H}_P)$. In words, the operator $\mathcal{TR}_I^P$ extends the interpretation I to add the new atomic facts that can be derived from P knowing I; $\mathcal{FA}_I^P$ adds the new negations of atomic facts that can be shown false in P by knowing I (via the uncovering of unfounded sets). An iterated fixed point operator builds up dynamic strata by constructing successive partial interpretations as follows.

Definition 2 (Iterated Fixed Point and Dynamic Strata) *For a normal program P let*

$$\begin{aligned} WFM_0 &= \langle\emptyset;\emptyset\rangle; \\ WFM_{\alpha+1} &= WFM_\alpha \cup \langle \mathcal{TR}^P_{WFM_\alpha}; \mathcal{FA}^P_{WFM_\alpha}\rangle; \\ WFM_\alpha &= \bigcup_{\beta<\alpha} WFM_\beta, \text{ for limit ordinal } \alpha. \end{aligned}$$

$WFM(P)$ denotes the fixed point interpretation WFM_δ, where δ is the smallest (countable) ordinal such that both sets $\mathcal{TR}^P_{WFM_\delta}$ and $\mathcal{FA}^P_{WFM_\delta}$ are empty. The stratum *of atom A, is the least ordinal β such that $A \in WFM_\beta$.*

(Przymusinski 1989) shows that $WFM(P)$ is in fact the well-founded model and that any undefined atoms of the well-founded model do not belong to any stratum – i.e. they are not added to WFM_δ for any ordinal δ. Thus, a program is *dynamically stratified* if every atom belongs to a stratum.

Above in the notation, we sometimes left implicit an argument (e.g., P) when it is clear in context. We also do so throughout this paper henceforth.

Radially Restrained Models

Norms and Abstractions

Abstraction functions may be understood with respect to norms, which can specify families of abstraction functions. Typically, if the norm of an atom A is greater than a given integer bound, A is abstracted.

A *norm* $N(\cdot)$ is a function from terms to non-negative integers such that

1. $N(t) = 0$ iff $t = \Lambda$ (the empty term)
2. t subsumes t' implies $N(t) \leq N(t')$

A norm is *finitary* iff for any finite non-negative integer k, the cardinality of the set $\{t | t \in \mathcal{H}_\mathcal{L} \wedge N(t) < k\}$ is finite.

An *abstraction* of a term t, denoted $abs(t)$, may replace subterms of t by position variables: formally, $abs(t)$ is a term such that if $abs(t)|_\pi \in (\mathcal{FN} \cup \mathcal{V})$, then $abs(t)|_\pi = t|_\pi$. For instance $p(f(g(X_{1.1.1}), X_{1.2}), X_2)$ is an abstraction of $p(f(g(a), X), X)$. It is easy to see that $abs(t)$ subsumes t, so for any norm $N(\cdot)$, $N(abs(t)) \leq N(t)$. An abstraction $abs(\cdot)$ is finitary if the cardinality of $\{abs(t) | t \in \mathcal{H}_\mathcal{L}\}$ is finite. Given two abstractions, $abs_1(\cdot) \leq abs_2(\cdot)$ if for all terms t, $abs_1(t)$ subsumes $abs_2(t)$. Note that if $abs_1(\cdot) \leq abs_2(\cdot)$, then $\{abs(t) | t \in \mathcal{H}_\mathcal{L}\} \subseteq \{abs(t) | t \in \mathcal{H}_\mathcal{L}\}$. Norms and abstractions are applied to atoms by taking those atoms as terms, and to rules by applying the operation to each atom underlying a literal in the rule.

Example 2 *A* depth norm*, $depth(\cdot)$, maps a term t to the maximal* depth *of any position in t, where the depth of the outermost function symbol of t is 1 and the depth of a position $\pi.i$ is the depth of π plus 1 if $t|_{\pi.i}$ is a not a position variable, and is the depth of π otherwise. For a positive integer k, a* depth-k abstraction *is an abstraction that maps t to itself if $depth(t)$ is less than or equal to k; and otherwise to the abstraction of t with depth k that is maximal with respect to subsumption. It is easy to see that such a maximal depth-k abstraction of t must be unique. Within the atom $A = p(a, f(b, g(c)))$ the depth of c is 4. The depth 3 abstraction of A is $p(a, f(b, g(X_{2.2.1})))$, and the depth 2 abstraction of A is $p(a, f(X_{2.1}, X_{2.2}))$. Both the depth norm and the family of depth-k abstractions (for positive integer k) are finitary.*

Depth-k abstractions are simple to understand and to implement. However the number of terms whose depth is less than k may grow exponentially (in $|\mathcal{FN}|$ and thus in $|P|$). Thus, other abstractions, based on the size of a term, or that weigh the occurrence of certain types of function symbols over others (e.g., lists) can be practically useful. Finally, note that the identity function on terms, $Id(\cdot)$, is an abstraction function, but is not finitary for languages that contain function symbols whose arity is 1 or greater. In fact, $Id(\cdot)$ is the maximal abstraction function.

Radially Restrained Models

Definition 3 *For a normal program P, abstraction function $abs(\cdot)$, sets Tr and Fa of ground atoms, and a 3-valued interpretation I (sometimes called a pre-interpretation):*

$True_I^P(abs, Tr) = \{A |$ *there is a clause $B \leftarrow L_1, ..., L_n$ in P, a grounding substitution θ such that $A = B\theta = abs(B\theta)$, and for every $1 \leq i \leq n$ either $L_i\theta$ is true in I, or $L_i\theta \in Tr\}$;*

$False_I^P(abs, Fa) = \{A |$ *for every clause $B \leftarrow L_1, ..., L_n$ in P and grounding substitution θ such that $A = B\theta = abs(B\theta)$ and there is some i $(1 \leq i \leq n)$ such that $L_i\theta$ is false in I or $L_i\theta \in Fa\}$.*

Unlike Definition 1, Definition 3 requires that $abs(B\theta) = B\theta$ in order for an atom to be considered either true or false. Clearly both $True_I^P$ and $False_I^P$ are monotonic

in their second arguments; and as with the well-founded model, we define $\mathcal{TR}^P_I(abs)$ as the least fixed point of $True^P_I(abs, \emptyset)$ and $\mathcal{FA}^P_I(abs)$ as the greatest fixed point of $False^P_I(abs, \mathcal{H}_P)$.

Definition 4 (Radially Restrained Model) *For a normal program P and abstraction function $abs(\cdot)$*

$$\begin{array}{rcl} WFM_0(abs) & = & \langle \emptyset; \emptyset \rangle; \\ WFM_{\alpha+1}(abs) & = & WFM_\alpha(abs) \cup \\ & & \langle \mathcal{TR}^P_{WFM_\alpha}(abs); \mathcal{FA}^P_{WFM_\alpha}(abs) \rangle; \\ WFM_\alpha(abs) & = & \bigcup_{\beta<\alpha} WFM_\beta(abs), \\ & & \textit{for limit ordinal } \alpha. \end{array}$$

The radially restrained model $WFM(abs, P)$ denotes the fixed point interpretation WFM_δ, where δ is the smallest ordinal such that both sets $\mathcal{TR}^P_{WFM_\delta}(abs)$ and $\mathcal{FA}^P_{WFM_\delta}(abs)$ are empty.

The following statement follows directly from Definition 3. Since the language of P has a finite number of function symbols and predicates, and since $abs(\cdot)$ is finitary, $True^P_I(abs, Tr)$ can only produce a finite number of grounded rules, even if I or Tr were infinite [1].

Proposition 1 *For a program P and finitary abstraction function $abs(\cdot)$ let*

$$WFM(abs, P) = \langle TrueAtoms; FalseAtoms \rangle.$$

The cardinality of $TrueAtoms$ is finite.

Because $\mathcal{TR}(abs)$ is monotonic, due to Proposition 1 it must reach fixed point for some finite ordinal. Accordingly, if $abs(\cdot)$ is finitary, $WFM(abs, P)$ will also reach fixed point at some finite ordinal.

Theorem 1 *Given a program P and finitary abstraction function $abs(\cdot)$, then $WFM(abs, P) = WFM(abs, P)_\delta$ for some finite ordinal δ.*

The main theorem about radially restrained well-founded models is as follows.

Theorem 2 *Let $abs_1(\cdot), abs_2(\cdot)$ be abstraction functions such that $abs_1(\cdot) \leq abs_2(\cdot)$. Then for any program P, $WFM(abs_1, P) \subseteq WFM(abs_2, P)$.*

Since the identity function, $Id(\cdot)$ is the maximal abstraction function, and since $WFM(Id, P) = WFM(P)$, Theorem 2 implies:

Corollary 1 *For a program P and abstraction function $abs(\cdot)$, $WFM(abs, P) \subseteq WFM(P)$.*

For any program P, Theorem 2 also implies that a chain of abstraction functions $abs_1(\cdot), abs_2(\cdot), \ldots$ such that for $i \leq j$, $abs_i(\cdot) \leq abs_j(\cdot)$ is associated with a chain of models: $WFM(abs_1, P), WFM(abs_2, P), \ldots, WFM(abs_j, P) \ldots$ such that for $i \leq j$, $WFM(abs_i, P) \subseteq WFM(abs_j, P)$. Thus, families of finitary abstraction functions, based on depth, size or other measures, provide successively more powerful finite approximations of the well-founded model.

[1] Proofs of all results, along with a full presentation of SLG$_{\mathsf{ABS}}$ (introduced in the next section) are available at http://www.cs.sunysb.edu/~tswift/webpapers/radial.pdf.

Tabled Resolution for Bounded Rationality

SLG$_{\mathsf{ABS}}$ is a tabled resolution method that correctly evaluates queries to radially restrained models of programs. SLG$_{\mathsf{ABS}}$ strictly extends SLG evaluation (Chen and Warren 1996) which models well-founded computation at an operational level, ensuring goal-directedness, termination and optimal complexity for a normal programs. SLG evaluation, along with numerous extensions of it, are well-described in the literature. Accordingly in this section we present only those extensions used in SLG$_{\mathsf{ABS}}$, after a brief review of the terminology required by the extensions.

Terminology Used

In the forest-of-trees model of SLG (Swift 1999), an evaluation is a possibly transfinite sequence of forests (sets) of trees in which each tree corresponds to a subgoal that has been encountered in an evaluation. When a new tabled subgoal S is encountered, a tree with root $S \leftarrow |S$ is added to the current forest by a NEW SUBGOAL operation, and children of the root are added through PROGRAM CLAUSE RESOLUTION operations. Other positive selected literals are resolved through the POSITIVE RETURN operation; while ground negative selected subgoals are resolved through the NEGATIVE RETURN operation, or their resolution may be delayed through the DELAYING operation. These delayed literals may later be evaluated through SIMPLIFICATION or ANSWER COMPLETION operations. The need to delay some literals arises because modern Prolog engines rely on a fixed order for selecting literals in a rule. However, well-founded computations cannot be performed using a fixed-order literal selection function. When it is determined that no more resolution may be performed for non-delayed literals in nodes of trees for a mutually dependent set of subgoals, the trees are marked as *complete* using the COMPLETION operation. If a subgoal S has been marked as *complete* and S has no answers, literals of the form $not\, S$ can be resolved away by the NEGATIVE RETURN operation.

More specifically, the nodes in each tree have the form

$$Ans \leftarrow Delays|Goals \qquad \text{or} \qquad \textit{fail}.$$

In the first form, Ans is an atom while $Delays$ and $Goals$ are sequences of literals. The second form is called a *failure node*. *Goals* represents the sequence of literals left to be examined, while *Delays* represents those literals that have been examined, but their resolution delayed. A node N is an *answer* when it is a leaf node for which $Goals$ is empty. If the $Delays$ of an answer is empty, it is termed an *unconditional answer*, otherwise, it is a *conditional answer*.

SLG resolution is used to resolve an answer A against a node N.

Definition 5 (SLG Resolution) *Let N be a node $A \leftarrow D|L_1, ..., L_n$, where $n > 0$. Let $Ans = A' \leftarrow D'|$ be an answer whose variables are disjoint from N. If $\exists i$, $1 \leq i \leq n$, such that L_i and A' are unifiable with mgu θ, then the* resolvent *of N and Ans on L_i has the form:*

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

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

$$(A \leftarrow D, L_i|L_1, ..., L_{i-1}, L_{i+1}, ..., L_n)\theta.$$

Example 3 below further illustrates the foregoing concepts within an SLG_{ABS} evaluation.

Definition 6 relates SLG forests to interpretations, and is used for the statement of correctness in Theorem 4.

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

- *A (ground) atom $A \in true(\mathcal{I}_{\mathcal{F}})$ iff A is in the ground instantiation of an unconditional answer $Ans \leftarrow |$ in $\mathcal{F}$.*
- *A (ground) atom $A \in false(\mathcal{I}_{\mathcal{F}})$ iff A is in the ground instantiation of a subgoal whose tree in $\mathcal{F}$ is marked as* complete*, and A is not in the ground instantiation of any answer in a tree in $\mathcal{F}$.*

An atom S is successful *(resp.* failed*) in $\mathcal{F}$ if S' is in $true(\mathcal{I}_{\mathcal{F}})$ ($false(\mathcal{I}_{\mathcal{F}})$) for every S' in the ground instantiation of S. A non-ground subgoal $not\ S$ succeeds (fails) if S fails (succeeds). Given an interpretation $\mathcal{J}$ and forest $\mathcal{F}$, the* restriction of $\mathcal{J}$ to $\mathcal{F}$, *$\mathcal{J}|_{\mathcal{F}}$ is the interpretation such that $true(\mathcal{J}|_{\mathcal{F}})$ ($false(\mathcal{J}|_{\mathcal{F}})$) consists of those atoms in $true(\mathcal{J})$ ($false(\mathcal{J})$) that are in the ground instantiation of some subgoal whose tree is in $\mathcal{F}$*

SLG_{ABS}

SLG_{ABS} extends SLG to use abstraction both when creating a tree for a new subgoal, and when deriving an answer.

Definition 7 (Subgoal Abstraction (Riguzzi and Swift 2013a)) NEW SUBGOAL*: Let $abs(\cdot)$ be an abstraction function, and let a forest $\mathcal{F}_n$ contain a tree with non-root node*

$$N = Ans \leftarrow Delays|G, Goals$$

where S is the underlying subgoal of the literal G. Assume $\mathcal{F}_n$ contains no tree with root $abs(S)$. Then add the tree $abs(S) \leftarrow |abs(S)$ to $\mathcal{F}_n$.

Abstaction is also used when an answer Ans is derived; If the abstraction is non-trivial, i.e., if $Ans \neq abs(Ans)$, then a special atom $undefined_{abs}$ is added to the $Delays$ of Ans.

Definition 8 (Answer Abstraction) POSITIVE RETURN*: Let $abs(\cdot)$ be an abstraction function, and let a forest $\mathcal{F}_n$ contain a tree with non-root node N whose selected literal S is positive. Let Ans be an answer for S in $\mathcal{F}_n$ and $N' = A \leftarrow Delays|Goals$ be the SLG resolvent of N and Ans on S.*

- *If $Goals$ is non-empty, then $N_{child} = N'$;*
- *Otherwise, if $abs(N') = N'$, then $N_{child} = N'$;*
- *Otherwise, if $abs(N') \neq N'$, $N_{child} = abs(A \leftarrow Delays, undefined_{abs})|$.*

If N does not have a child N_{child} in $\mathcal{F}_n$, then add N_{child} as a child of N.

For SLG_{ABS} to be correct with respect to radially restrained models of normal programs, negation must be extended to handle the lack of safety that is introduced by abstraction. The following example shows how this can occur, and illustrates the SLG and SLG_{ABS} terminology used so far.

Example 3 *Figure 1 shows the SLG_{ABS} evaluation of the query* r(X) *against the safe program $P_{abs\text{-}unsafe}$:*

p(s(X)) ← p(X).
r(X) ← p(X),not q(X).
p(0).
q(0).

where a depth-2 abstraction function is used (local scheduling is assumed for this evaluation, cf. (Swift and Warren 2012)). The evaluation begins in a manner identical to SLG evaluation. The initial forest consists simply of node 0. Children of root nodes are created by PROGRAM CLAUSE RESOLUTION*, which creates node 1. The selected (leftmost) literal of node 1 is* p(X)*, which is new at this point of the evaluation. A* NEW SUBGOAL *operation creates node 2, (although an abstraction is applied, it is trivial), and* PROGRAM CLAUSE RESOLUTION *creates node 3, an unconditional answer. Reapplication of* PROGRAM CLAUSE RESOLUTION *also creates node 4, whose selected literal is not new to the evaluation. There is already an answer for* p(X) *so that* POSITIVE RETURN *is applicable to this node; repeated applications of* POSITIVE RETURN *produce nodes 5 and 6. Although abstraction is performed for all answers, it is trivial except when producing node 6. Once node 6 is produced, the tree for* p(X) *is completely evaluated, and a* COMPLETION *operation marks it* complete*. Another* POSITIVE RETURN *operation produces node 7 which has a selected negative literal. Evaluation of the subgoal* q(0) *shows that* q(0) *is successful, and a* NEGATIVE RETURN *operation creates a failure node as child 10. The evaluation proceeds until finally the conditional answer, node 6 is resolved against the selected literal of node 14. Because the answer was conditional, the selected literal* p(s(s(X)) *is moved to the $Delays$ after resolution (Definition 5). Because of the abstraction used to produce node 6, the next selected literal* not q(s(s(X))) *is non-ground. Nonetheless, the atom* q(s(s(X))) *becomes failed (Definition 6), once its tree is completed with no answers (step 15a). Because* q(s(s(X))) *is failed, a* NEGATIVE RETURN *operation resolves the selected literal away, leading to the conditional answer node 16.*

Thus SLG_{ABS} has the following extensions over SLG:

1. Abstraction is used both when creating new trees (in the NEW SUBGOAL operation), and when adding an answer (in the POSITIVE RETURN operation);
2. A special atom $undefined_{abs}$ is added to the $Delays$ of each non-trivially abstracted answer A (in the POSITIVE RETURN operation). The truth value of $undefined_{abs}$ is always *undefined*, so it can never be removed from the $Delays$ of A, forcing A to have a truth value of *undefined* as well; and
3. NEGATIVE RETURN is defined so that literal $not\ A$ can be resolved away in a forest $\mathcal{F}$ if A is failed in $\mathcal{F}$, regardless of whether A is ground.

Of course, NEW SUBGOAL and POSITIVE RETURN in SLG_{ABS} can be reduced to the classical definitions of SLG by setting the $abs(\cdot)$ to the identity function.

If a finitary abstraction function is used in SLG_{ABS}, then any forest has a finite number of trees and answers. This fact together with other tabling properties ensures the following.

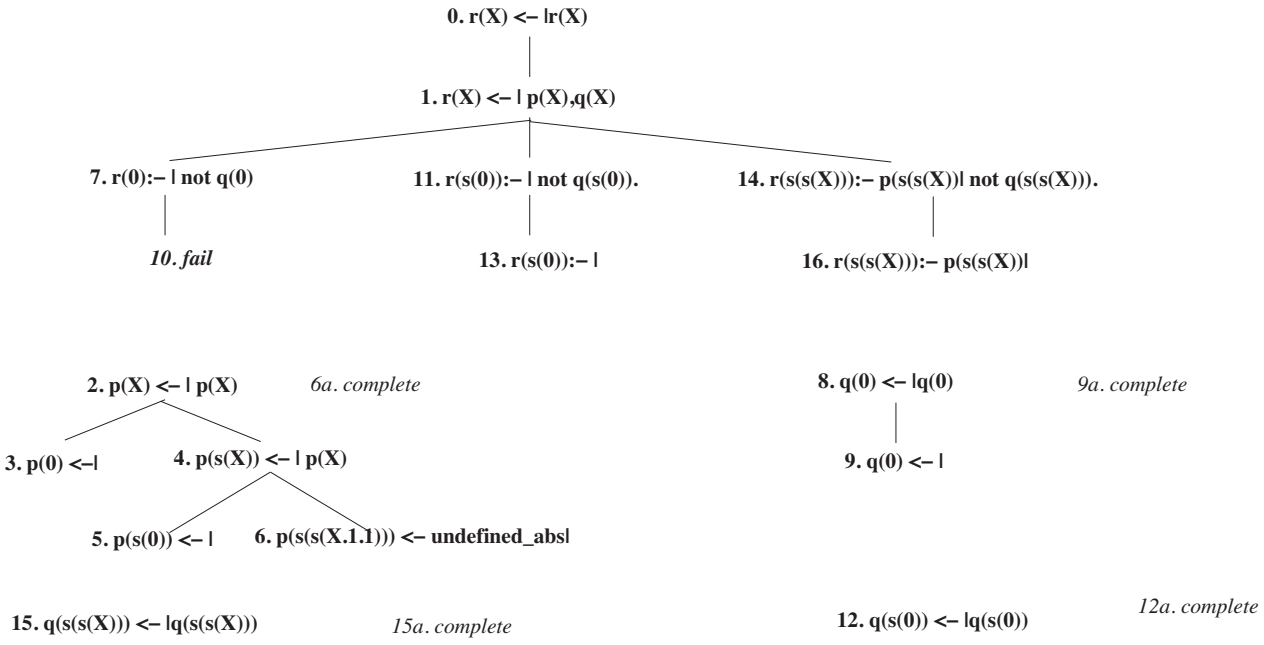

Figure 1: Final forest for the query $r(X)$ to $P_{abs\text{-}unsafe}$.

Theorem 3 *Let Q be a query to a normal program P, and let $abs(\cdot)$ be a finitary abstraction function. Then any SLG$_{\mathsf{ABS}}$ evaluation $\mathcal{E}$ of Q against P reaches a final forest $\mathcal{F}_{fin}$ after a finite number of steps.*

Regardless of whether $abs(\cdot)$ is finitary, a SLG$_{\mathsf{ABS}}$ evaluation is complete with respect to a model that is restrained by the same abstraction function. If P is unsafe, SLG$_{\mathsf{ABS}}$ may derive truth values that are not in $WFM(abs, P)$, but that are in $WFM(P)$. This occurs if SLG$_{\mathsf{ABS}}$ derives a non-ground answer A for which $A = abs(A)$, and for some atom A' in the ground instantiation of A, $A' \neq abs(A')$. In this case A' is undefined in $WFM(abs, P)$, although A is true in the interpretation induced by the final forest of the SLG$_{\mathsf{ABS}}$ evaluation ($\mathcal{I}_{\mathcal{F}_{fin}}$).

Theorem 4 *Let $\mathcal{E}$ be an SLG$_{\mathsf{ABS}}$ evaluation of a query Q to a normal program P using abstraction function $abs(\cdot)$, such that $\mathcal{E}$ has a final forest $\mathcal{F}_{fin}$. Then*

$$WFM(abs, P)|_{\mathcal{F}_{fin}} \subseteq \mathcal{I}_{\mathcal{F}_{fin}} \subseteq WFM(P)|_{\mathcal{F}_{fin}}.$$

Complexity of SLG$_{\mathsf{ABS}}$

The best currently known bound on worst case complexity for computing the well-founded semantics of a program P is $size(P) \times |atoms(P)|$ (van Gelder, Ross, and Schlipf 1991). In order to relate the complexity of SLG$_{\mathsf{ABS}}$ to this result, we extend the cost model of (Riguzzi and Swift 2013b).

The first aspect of our cost model, $\mathcal{C}_{SLG_{\mathsf{ABS}}}$, addresses the fact that evaluations may terminate on ground programs that are not finite. Let P be a ground (normal) program, and Q an atomic query to P (not necessarily ground). Then the *atomic search space* of Q, P_Q, consists of the union of all ground instantiations of Q in $\mathcal{B}_P$ together with all atoms reachable in the atom dependency graph of P from any ground instantiation of Q. By Theorem 3 a SLG$_{\mathsf{ABS}}$ evaluation $\mathcal{E}$ of Q against P that uses a finitary abstraction function will produce a final forest $\mathcal{F}_{fin}$ after a finite number of steps, and $\mathcal{F}_{fin}$ will itself be finite. It is evident that the set of subgoals corresponding to trees in $\mathcal{F}_{fin}$ ($subgoals(\mathcal{F}_{fin})$) is finite. Because $\mathcal{F}_{fin}$ may contain non-ground subgoals, it is not the case that $subgoals(\mathcal{F}_{fin}) \subseteq P_Q$; however if depth-$k$ abstraction is used, it can be shown that $|subgoals(\mathcal{F}_{fin})| \leq 2 \times |atoms(P_Q)|$.

Next, given the finite sequence $\mathcal{E}$, we can construct the set of (ground) rules that were used in some PROGRAM CLAUSE RESOLUTION operation and denote this set as $P_Q(\mathcal{E})$. It is evident that $P_Q(\mathcal{E}) \subseteq P_Q \subseteq P$, and that $P_Q(\mathcal{E})$ must always be finite. Define for a rule r, $size(r)$ as one plus the number of body literals in r. Extending this, $size(P_Q(\mathcal{E}))$ is defined as the sum of sizes of rules in $P_Q(\mathcal{E})$. $\mathcal{C}_{SLG_{\mathsf{ABS}}}$ thus does not consider the size of terms within an atom or literal.

Finally, $\mathcal{C}_{SLG_{\mathsf{ABS}}}$ determines the cost of each SLG$_{\mathsf{ABS}}$ operation. Note, since the scope of an abstraction function is an atom, the cost of applying an abstraction function is constant in $\mathcal{C}_{SLG_{\mathsf{ABS}}}$ [2]. Accordingly under $\mathcal{C}_{SLG_{\mathsf{ABS}}}$ the NEW SUBGOAL, PROGRAM CLAUSE RESOLUTION, POSITIVE RETURN, NEGATIVE RETURN, DELAYING, and SIMPLIFICATION operations each affect one goal or delay literal and are considered constant time. The COMPLETION operation, however, applies to a set of subgoals $\mathcal{S}$ in a forest $\mathcal{F}$ and its cost is proportional to the size of $\mathcal{S}$: in the worst case this is

[2] Of course a practical implementation of an abstraction function should have a low cost as a function of the actual size of an atom to which it is applied.

$|subgoals(\mathcal{F})|$. Similarly, the ANSWER COMPLETION operation must determine an unsupported set of answers and its worst case is $size(P_Q(\mathcal{E}))$.

The cost model $\mathcal{C}_{SLG_{\mathsf{ABS}}}$ thus consists of

1. The definition of $subgoals(\mathcal{F})$ which is finite, and is $\mathcal{O}(atoms(P))$ if $atoms(P)$ is finite;
2. The definition of $size(P_Q(\mathcal{E}))$ which finite and is $\mathcal{O}(size(P))$ if $size(P)$ is finite; and
3. Costs for each individual $\text{SLG}_{\mathsf{ABS}}$ operation.

Theorem 5 *Let P be a ground normal program, Q a ground query, and $\mathcal{E}$ a terminating $\text{SLG}_{\mathsf{ABS}}$ evaluation of Q against P that uses depth-k abstraction, and with final forest $\mathcal{F}_{fin}$. Then under the cost mode $\mathcal{C}_{SLG_{\mathsf{ABS}}}$, the cost of $\mathcal{E}$ is $\mathcal{O}(|subgoals(\mathcal{F}_{fin})| \times size(P_Q(\mathcal{E})))$.*

Implementation, Performance and Scalability

$\text{SLG}_{\mathsf{ABS}}$ is implemented using depth-k abstraction in version 3.3.7 (publicly available) of XSB (XSB 2013), based in part on a prior implementation of subgoal abstraction. From the programmer's perspective, depth-k abstraction is not used by default, but can be invoked using different values of k on a predicate basis. Answer abstraction is performed in the tabling engine of XSB, the SLG-WAM, during the check/insert step which checks whether an answer exists in a given table, and inserts the answer into the table if not. A counter maintains the current depth of the answer $Ans \leftarrow Delays|$ being traversed; if the depth of Ans is greater than k then the current subterm is replaced by a free (position) variable. In addition, the atom $undefined_{abs}$, a reserved atom in XSB, is added to $Delays$ if it is not already included, indicating that Ans is *undefined*. The overhead of answer abstraction is thus the cost of maintaining the depth-counter, along with that of copying $undefined_{abs}$ into $Delays$ if the depth bound is exceeded.

If no answer abstraction function is specified (so that answers will not be abstracted) the overhead consists solely of the cost of maintaining the the depth counter within the answer check/insert operation. For various forms of linear recursion, we measured this overhead at $0 - 4\%$ based on the ratio of answers to subgoals in a given benchmark.

A series of independent studies have shown XSB to be highly scalable (OpenRuleBench 2011). In addition, recent work with trace-based analysis in XSB has performed sophisticated analysis on trace logs with 10^7 to 10^8 and more events, where each event corresponds to a Prolog fact that is dynamically loaded for the analysis. This scalability has not been affected by the extension to $\text{SLG}_{\mathsf{ABS}}$ in the SLG-WAM.

Discussion and Current Work

This paper has shown how radially restrained well-founded models of a program approximate the well-founded model in a clear manner (Theorem 2). Queries to these models terminate correctly (Theorems 3 and 4) with low abstract complexity (Theorem 5). Tabled resolution for restrained models can be implemented with low overhead on performance, without impacting the scalability of query evaluation.

Current work is studying how this kind of bounded rationality is best exploited for practical KR. To date, this study has been primarily in the SILK project, which has sought to provide a framework for scalable logic-based KR. SILK's logic, Rulelog, derives its scalability in part from its reliance on the well-founded semantics, which offers a low computational complexity and supports top-down query evaluation. SILK has been used to evaluate sets of high-level rules about cell biology at the first-year college level. These rules are constructed by a team of knowledge engineers who, as an experiment, constructed rules directly from textual knowledge. While the heavy use of tabling makes evaluation of such rule sets possible, they are sometimes not well-behaved, so that knowledge engineers may be faced with "run-away" computations. Radial restraint bounds these computations so that they will terminate. Then the results of queries may be analyzed — via SILK's justification-based debugging and other introspection routines — and used to modify problematic rules.

Besides such application piloting, current work is also addressing several areas of mechanisms and theory around radial restraint. One area is formulating results on conditions that ensure computational tractability of LP and Rulelog. A second area is developing methods to coordinate radial restraint with temporal bounds (i.e., time-outs). A third area is developing analysis, such as estimated maximum number of answers, that is specific to various abstraction norms, e.g., term size, term depth, and list vs. non-list functions. A fourth area is developing justification mechanisms to inform users whether an atom was undefined because of restraint, versus because negation lacked stratifiability. A fifth area is developing methods for introspection during long-running queries (within which SILK permits interrupting and resuming computation), so that a knowledge engineer can obtain information about what *subqueries* in a current paused state of a computation have a truth value that is undefined due to radial restraint.

A final area of current work is developing techniques and theory for *non-radial* kinds of restraint. Restraint appears to be potentially a rich realm for work in the field of KR overall.

Acknowledgements

The authors would like to acknowledge support, while performing the work described in this paper, from: Vulcan, Inc., specifically, the SILK project within overall larger Project Halo; and FCT Project ERRO PTDC/EIACCO/121823/2010. The authors also thank Keith Goolsbey and Michael Kifer for inspiring and helpful discussions, the rest of the overall SILK team for their encouragement, and the anonymous reviewers for their stimulating comments.

References

Alviano, M.; Faber, W.; and Leone, N. 2010. Disjunctive ASP with functions: Decidable queries and effective computation. *Theory and Practice of Logic Programming* 10(4-6):497–512.

Anderson, M. L., and Oates, T. 2007. A review of recent research in metareasoning and metalearning. *AI Magazine* 28:7–16.

Bonatti, P. 2004. Reasoning with infinite stable models. *Artificial Intelligence* 156:75–111.

Calimeri, F.; Cozza, S.; Ianni, G.; and Leone, N. 2008. In *International Conference on Logic Programming*, 407–424.

Chen, W., and Warren, D. S. 1996. Tabled Evaluation with Delaying for General Logic Programs. *Journal of the ACM* 43(1):20–74.

Chen, W.; Kifer, M.; and Warren, D. S. 1993. HiLog: A foundation for higher-order logic programming. *Journal of Logic Prog.* 15(3):187–230.

Fisher, M., and Ghidini, C. 1999. Programming resource-bounded deliberative agents. In *International Joint Conference on Artificial Intelligence*, volume 16, 200–205.

Fisher, M.; Bordini, R. H.; Hirsch, B.; and Torroni, P. 2007. Computational logics and agents a roadmap of current technologies and future trends. *Computational Intelligence* 23(1):61–91.

Grant, J.; Kraus, S.; and Perlis, D. 2000. A logic for characterizing multiple bounded agents. *Autonomous Agents and Multi-Agent Systems* 3(4):351–387.

Konolige, K. 1983. A deductive model of belief. In *International Joint Conference on Artificial Intelligence*, 377–381.

Levesque, H. 1984. A logic of implicit and explicit belief. In *Proceedings of the Conference of the American Association for Artificial Intelligence*, 198–202.

OpenRuleBench. 2011. Openrulebench: Benchmarks for semantic web rule engines. rulebench.projects.semwebcentral.org, benchmark suites were tested in 2009, 2010, and 2011.

Przymusinski, T. 1989. Every logic program has a natural stratification and an iterated least fixed point model. In *ACM Principles of Database Systems*, 11–21. ACM Press.

Riguzzi, F., and Swift, T. 2013a. Termination of logic programs with finite three-valued models.

Riguzzi, F., and Swift, T. 2013b. Well-definedness and efficient inference for probabilistic logic programming under the distribution semantics. *Theory and Practice of Logic Programming* 13(2):279–302.

Russell, S., and Subramanian, D. 1995. Provably bounded-optimal agents. *Journal of Artificial Intelligence Research* 2.

SILK. 2013. SILK: Semantic Inferencing on Large Knowledge. http://silk.semwebcentral.org (project begun in 2008).

Swift, T., and Warren, D. 2012. XSB: Extending the power of Prolog using tabling. *Theory and Practice of Logic Programming* 12(1-2):157–187.

Swift, T. 1999. A new formulation of tabled resolution with delay. In *Progress in Artificial Intelligence*, 163–177.

Tamaki, H., and Sato, T. 1986. OLDT resolution with tabulation. In *International Conference on Logic Programming*, 84–98.

van Gelder, A.; Ross, K.; and Schlipf, J. 1991. Unfounded sets and well-founded semantics for general logic programs. *Journal of the ACM* 38(3):620–650.

Wan, H.; Grosof, B.; Kifer, M.; Fodor, P.; and Liang, S. 2009. Logic programming with defaults and argumentation theories. In *International Conference on Logic Programming*, 432–448.

XSB. 2013. XSB Prolog. http://xsb.sourceforge.net (first released in 1993).