Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence

Liberal Safety for Answer Set Programs with External Sources*

Thomas Eiter and **Michael Fink** and **Thomas Krennwallner** and **Christoph Redl**
Institute of Information Systems, Vienna University of Technology
Favoritenstraße 9-11, A-1040 Vienna, Austria
{*eiter,fink,tkren,redl*}*@kr.tuwien.ac.at*

Abstract

Answer set programs with external source access may introduce new constants that are not present in the program, which is known as *value invention*. As naive value invention leads to programs with infinite grounding and answer sets, syntactic safety criteria are imposed on programs. However, traditional criteria are in many cases unnecessarily strong and limit expressiveness. We present liberal domain-expansion (de-) safe programs, a novel generic class of answer set programs with external source access that has a finite grounding and allows for value invention. De-safe programs use so-called term bounding functions as a parameter for modular instantiation with concrete—e.g., syntactic or semantic or both—safety criteria. This ensures extensibility of the approach in the future. We provide concrete instances of the framework and develop an operator that can be used for computing a finite grounding. Finally, we discuss related notions of safety from the literature, and show that our approach is strictly more expressive.

1 Introduction

Answer Set Programming (ASP) is a declarative programming approach which due to expressive and efficient systems like CLASP, DLV, and SMODELS, has been gaining popularity for many applications (Brewka, Eiter, and Truszczyński 2011). Current trends in computing, such as context awareness or distributed systems, raised the need for access to external sources in a program, which, e.g., on the Web ranges from light-weight data access (e.g., XML, RDF, or data bases) to knowledge-intensive formalisms (e.g., description logics).

To cater for this need, HEX-programs (Eiter et al. 2005) extend ASP with so-called external atoms, through which the user can couple any external data source with a logic program. Roughly, such atoms pass information from the program, given by predicate extensions, into an external source which returns output values of an (abstract) function that it computes. This convenient extension has been exploited for many different applications, including querying data and ontologies on the Web, multi-context reasoning, or e-government, to mention a few (Eiter et al. 2011). It is highly expressive as external sources may introduce new constants not present in the program, which is called *value invention*.

Naive support of value invention leads to programs with infinite groundings and answer sets. E.g., the program

$$\Pi = \left\{ \begin{array}{ll} r_1\colon t(a). & r_3\colon s(Y) \leftarrow t(X), \,\&\, cat[X,a](Y). \\ r_2\colon dom(aa). & r_4\colon t(X) \leftarrow s(X), dom(X). \end{array} \right\}$$

where $\&\, cat[X,a](Y)$, returning in Y, as expected, the string in X with a appended, has an infinite grounding. However, only rules using a and aa are relevant for program evaluation. Note that external sources are largely black boxes to the reasoner. Thus the set of relevant constants for grounding might be *intuitively* clear, but not *formally*. Predetermining is in general not possible. To ensure that a finite portion of the grounding of the program is sufficient, i.e., has the same answer sets (which we call *finite restrictability*), existing approaches impose strong syntactic safety restrictions on programs, as in Eiter et al. (2006) or Calimeri, Cozza, and Ianni (2007). As it turns out, they limit expressiveness often unnecessarily, i.e., programs may not fulfil traditional safety conditions while they are clearly finitely restrictable, as the program Π above.

Our overall objective is thus to introduce a more liberal notion of safety that still ensures finite restrictability. However, rather than to merely generalize an existing notion, we aim for a generic notion at a conceptual level that may incorporate besides syntactic also semantic information about sources. Briefly, our contributions are as follows:

- We introduce the notion of *liberal domain-expansion (de-) safety*, which is parameterized with *term bounding functions* (TBFs) (Section 3). The latter embody criteria which ensure that only finitely many ground instances of a term in a program matter. The notion provides a generic framework in which TBFs can be modularly exchanged and combined, which offers attractive flexibility and future extensibility.
- We then provide concrete TBFs, which exploit like traditional approaches *syntactic structure*, but also *semantic properties* of the program, hinging on cyclicity and meta-information (Section 4); this allows to cover the program Π above. Thanks to modularity, the TBFs can be fruitfully combined into a single, more powerful TBF.
- We provide an optimized operator which can be used for computing a finite grounding of de-safe programs (Section 5).

*This research has been supported by the Austrian Science Fund (FWF) project P20840, P20841, P24090, and by the Vienna Science and Technology Fund (WWTF) project ICT08-020.

- We implement de-safety in the open-source system DLVHEX and model, as a simple showcase, a pushdown automaton in a HEX-program exploiting de-safety. The program can be flexibly extended to perform parsing under additional constraints, as for instance in RNA string analysis (Section 6).
- Finally, we discuss related work and show that our approach is more liberal, and conclude the paper (Sections 7 and 8).

2 Preliminaries

We start with basic definitions and introduce HEX-programs; for details and background see Eiter et al. (2005). The signature consists of mutually disjoint sets $\mathcal{P}$ of predicates, $\mathcal{X}$ of external predicates, $\mathcal{C}$ of constants, and $\mathcal{V}$ of variables. Note that $\mathcal{C}$ may contain constants that do not occur explicitly in a HEX program and can even be infinite.

A *(signed) ground literal* is a positive or a negative formula $\mathbf{T}a$ resp. $\mathbf{F}a$, where a is a ground atom of form $p(c_1, \ldots, c_\ell)$, with predicate $p \in \mathcal{P}$ and constants $c_1, \ldots, c_\ell \in \mathcal{C}$, abbreviated $p(\mathbf{c})$. An *assignment* $\mathbf{A}$ is a consistent set of literals. We make the convention that if an assignment does not explicitly contain $\mathbf{T}a$ or $\mathbf{F}a$ for some atom a, i.e. the assignment is partial, then a is false w.r.t. $\mathbf{A}$.

Syntax. HEX-programs generalize (disjunctive) extended logic programs under the answer set semantics (Gelfond and Lifschitz 1991) with *external atoms* of the form $\&g[\mathbf{X}](\mathbf{Y})$, where $\&g \in \mathcal{X}$, $\mathbf{X} = X_1, \ldots, X_\ell$ and each $X_i \in \mathcal{P} \cup \mathcal{C} \cup \mathcal{V}$ is an *input parameter*, and $\mathbf{Y} = Y_1, \ldots, Y_k$ and each $Y_i \in \mathcal{C} \cup \mathcal{V}$ is an *output term*.

Each $p \in \mathcal{P}$ has arity $ar(p) \geq 0$ and each $\&g \in \mathcal{X}$ has input arity $ar_{\text{I}}(\&g) \geq 0$ and output arity $ar_{\text{O}}(\&g) \geq 0$. Each input argument i of $\&g$ ($1 \leq i \leq ar_{\text{I}}(\&g)$) has *type* $\mathbf{const}$ or $\mathbf{pred}$, denoted $\tau(\&g, i)$, where $\tau(\&g, i) = \mathbf{pred}$ if $X_i \in \mathcal{P}$ and $\tau(\&g, i) = \mathbf{const}$ otherwise.

A HEX-*program* (or *program*) consists of rules r of form

$$a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots, \text{not } b_n \ ,$$

where each a_i is an (ordinary) atom and each b_j is either an ordinary atom or an external atom, and $k + n > 0$.

The *head* of r is $H(r) = \{a_1, \ldots, a_n\}$ and the *body* is $B(r) = B^+(r) \cup \text{not } B^-$, where $B^+(r) = \{b_1, \ldots, b_m\}$ is the *positive body*, $B^-(r) = \{b_{m+1}, \ldots, b_n\}$ is the *negative body*, and $\text{not } S = \{\text{not } b \mid b \in S\}$. For any rule, set of rules O, etc., let $A(O)$ and $EA(O)$ be the set of all ordinary and external atoms occurring in O, respectively.

Semantics. The semantics of a HEX-program Π is defined via its grounding $grnd(\Pi)$ (over $\mathcal{C}$) as usual, where the value of a ground external atom $\&g[\mathbf{p}](\mathbf{c})$ w.r.t. an interpretation $\mathbf{A}$ is given by the value $f_{\&g}(\mathbf{A}, \mathbf{p}, \mathbf{c})$ of a $k{+}l{+}1$-ary Boolean *oracle function* $f_{\&g}$ (Eiter et al. 2005). The input parameter $p_i \in \mathbf{p}$ is *monotonic* if, $f_{\&g}(\mathbf{A}, \mathbf{p}, \mathbf{c}) \leq f_{\&g}(\mathbf{A}', \mathbf{p}, \mathbf{c})$ whenever $\mathbf{A}'$ increases $\mathbf{A}$ only by literals $\mathbf{T}a$ where a has predicate p_i; otherwise, p_i is *nonmonotonic*.

Satisfaction of (sets of) literals, rules, programs etc. O w.r.t. $\mathbf{A}$ (denoted $\mathbf{A} \models O$, i.e., $\mathbf{A}$ is a *model* of O) extends to HEX in the obvious way. An *answer set* of Π is any model $\mathbf{A}$ of the *FLP-reduct* (Faber, Leone, and Pfeifer 2011) $\Pi^{\mathbf{A}} = \{r \in grnd(\Pi) \mid \mathbf{A} \models B(r)\}$ of Π w.r.t. $\mathbf{A}$ whose positive part (i.e., $\{\mathbf{T}a \in \mathbf{A}\}$) is subset-minimal; the set of all answer sets of Π is denoted by $\mathcal{AS}(\Pi)$.

Safety. A program is *safe*, if each variable in a rule r occurs also in a positive body atom in $B^+(r)$. However, due to external atoms we need additional safety criteria.

Example 1 *Let $\&cat[X, a](Y)$ be true iff Y is the string catenation of X and a. Then $\Pi = \{s(a).\ s(Y) \leftarrow s(X), \&cat[X, a](Y)\}$ is safe but not finitely groundable.* □

Thus the notion of *strong safety* was introduced in Eiter et al. (2006), which limits the output of cyclic external atoms.

Definition 1 (External Atom Dependencies) *Let Π be a* HEX*-program and $a = \&g[\mathbf{X}](\mathbf{Y})$ be an external atom in Π.*

- *If $b = p(\mathbf{Z}) \in \bigcup_{r \in \Pi} H(r)$, then a* depends external monotonically (resp. nonmonotonically) on *b, denoted $a \rightarrow^e_m b$ (resp. $a \rightarrow^e_n b$), if $X_i = p$ for some monotonic (resp. nonmonotonic) parameter $X_i \in \mathbf{X}$ ($= X_1, .., X_\ell$).*
- *If $\{a,\ p(\mathbf{Z})\} \subseteq B^+(r)$, some $X_i \in \mathbf{X}$ occurs in $\mathbf{Z}$, and $\tau(\&g, i) = \mathbf{const}$, then $\&g[\mathbf{X}](\mathbf{Y}) \rightarrow^e_m p(\mathbf{Z})$.*
- *If $\{a,\ \&h[\mathbf{V}](\mathbf{U})\} \subseteq B^+(r)$, some $X_i \in \mathbf{X}$ occurs in $\mathbf{U}$, and $\tau(\&g, i) = \mathbf{const}$, then $\&g[\mathbf{X}](\mathbf{Y}) \rightarrow^e_m \&h[\mathbf{V}](\mathbf{U})$.*

Definition 2 (Atom Dependencies) *For a program Π and (ordinary or external) atoms a, b, we say:*

(i) a depends monotonically on *b, denoted $a \rightarrow_m b$, if:*
- *some rule $r \in \Pi$ has $a \in H(r)$ and $b \in B^+(r)$; or*
- *there are rules $r_1, r_2 \in \Pi$ such that $a \in B(r_1)$ and $b \in H(r_2)$ and a unifies with b; or*
- *some rule $r \in \Pi$ has $a \in H(r)$ and $b \in H(r)$.*

(ii) a depends nonmonotonically on *b, denoted $a \rightarrow_n b$, if there is some rule $r \in \Pi$ such that $a \in H(r)$ and $b \in B^-(r)$.*

The following definition represents these dependencies.

Definition 3 (Atom Dependency Graph) *The* atom dependency graph *$ADG(\Pi) = (V_A, E_A)$ of a program Π has as nodes V_A the (nonground) atoms occurring in non-facts r (i.e., $k \neq 1$ or $n > 0$) of Π and as edges E_A the dependency relations $\rightarrow_m, \rightarrow_n, \rightarrow^e_m, \rightarrow^e_n$ between these atoms in Π.*

This allows us to introduce strong safety as follows.

Definition 4 (Strong Safety) *An atom $b = \&g[\mathbf{X}](\mathbf{Y})$ in a rule r of a* HEX*-program Π is* strongly safe *w.r.t. r and Π, if either there is no cyclic dependency over b in $ADG(\Pi)$, or every variable in $\mathbf{Y}$ occurs also in a positive ordinary atom $a \in B^+(r)$ not depending on b in $ADG(\Pi)$.*

A program Π is strongly safe, if every external atom in a rule $r \in \Pi$ is strongly safe w.r.t. r in Π.

Example 2 (cont'd) *In the program Π of Example 1, $\&cat[X, a](Y)$ is not strongly safe because it is in a cycle and no ordinary body atom contains Y. To make the program strongly safe, one can add a domain predicate as in $\Pi' = \{s(a); s(Y) \leftarrow s(X), \&cat[X, a](Y), dom(Y)\}$.* □

We can now introduce *strong domain-expansion safety*.

Definition 5 (Strong Domain-expansion Safety) *A program Π is* strongly domain-expansion safe*, if it is safe and each external atom in a rule r is strongly safe w.r.t. r and Π.*

Strong domain-expansion safety of a program Π guarantees that for some finite subset $\Pi' \subseteq grnd(\Pi)$, the answer sets of Π' and Π coincide on positive literals (in symbols, $\Pi' \equiv^{pos} \Pi$); we call this property *finite restrictability* of Π.

3 Liberal Safety Criteria

Strong domain-expansion safety is overly restrictive, as it also excludes programs that clearly *are* finitely restrictable.

Example 3 *Reconsider the program Π from Section 1. It is not strongly domain-expansion safe because Y in r_3 does not occur in an ordinary body atom that does not depend on $\&cat[X,a](Y)$. However, Π is finitely restrictable as the cycle is "broken" by $dom(X)$ in r_4.* □

In this section, we introduce a new notion of *liberal domain-expansion safety* which incorporates both syntactic and semantic properties of the program at hand. In the following, *domain-expansion safety* (*de-safety*) refers to liberal domain-expansion safety, unless we explicitly say strong domain-expansion safety. Compared to the latter, this gives us a larger class of programs which are guaranteed to have a finite grounding that preserves all answer sets. Unlike strong de-safety, liberal de-safety is not a property of entire atoms but of *attributes*, i.e., pairs of predicates and argument positions. Intuitively, an attribute is de-safe, if the number of different terms in an answer-set preserving grounding (i.e. a grounding which has the same answer sets if restricted to the positive atoms as the original program) is finite. A program is de-safe, if all its attributes are de-safe.

Our notion of liberal de-safety is designed in an extensible fashion, i.e., such that several safety criteria can be easily integrated. For this we parameterize our definition of de-safety by a *term bounding function* (TBF), which identifies variables in a rule that are ensured to have only finitely many instantiations in the answer set preserving grounding. Finiteness of the overall grounding follows then from the properties of TBFs. Concrete syntactic and semantic properties are realized in our definitions of concrete TBFs (cf. Section 4).

For an ordinary predicate $p \in \mathcal{P}$, let $p{\restriction}i$ be the *i-th attribute of p* for all $1 \leq i \leq ar(p)$. For an external predicate $\&g \in \mathcal{X}$ with input list $\mathbf{X}$ in rule r, let $\&g[\mathbf{X}]_r{\restriction}_T i$ with $T \in \{\text{I}, \text{O}\}$ be the *i-th input resp. output attribute of $\&g[\mathbf{X}]$ in r* for all $1 \leq i \leq ar_T(\&g)$. For a ground program P, the *range* of an attribute is, intuitively, the set of ground terms which occur in the position of the attribute. Formally, for an attribute $p{\restriction}i$ we have $range(p{\restriction}i, \Pi) = \{t_i \mid p(t_1, \ldots, t_{ar(p)}) \in A(\Pi)\}$; for an attribute $\&g[\mathbf{X}]_r{\restriction}_T i$ we have $range(\&g[\mathbf{X}]_r{\restriction}_T i, \Pi) = \{x_i^T \mid \&g[\mathbf{x}^{\text{I}}](\mathbf{x}^{\text{O}}) \in EA(\Pi)\}$, where $\mathbf{x}^s = x_1^s, \ldots, x_{ar_s(\&g)}^s$.

Example 4 *Some attributes of the program Π from Section 1 are $t{\restriction}1$, $\&cat[X,a]_{r_3}{\restriction}_{\text{I}}2$ and $\&cat[X,a]_{r_3}{\restriction}_{\text{O}}1$. Furthermore, $range(t{\restriction}1, \Pi) = \{a\}$.*

We use the following monotone operator to compute by fixpoint iteration a finite subset of $grnd(\Pi)$ for a program Π:

$$G_\Pi(\Pi') = \bigcup_{r \in \Pi} \{r\theta \mid \exists \mathbf{A} \subseteq \mathcal{A}(\Pi'), \mathbf{A} \not\models \bot, \mathbf{A} \models B^+(r\theta)\},$$

where $\mathcal{A}(\Pi') = \{\mathbf{T}a, \mathbf{F}a \mid a \in A(\Pi')\} \setminus \{\mathbf{F}a \mid a \leftarrow . \in \Pi\}$ and $r\theta$ is the ground instance of r under variable substitution $\theta\colon \mathcal{V} \to \mathcal{C}$. Note that in this definition, $\mathbf{A}$ might be partial, but by convention we assume that all atoms which are not explicitly assigned to true are false. That is, G_Π takes a ground program Π' as input and returns all rules from $grnd(\Pi)$ whose positive body is satisfied under some assignment over the atoms of Π'. Intuitively, the operator iteratively extends the grounding by new rules if they are possibly relevant for the evaluation, where relevance is in terms of satisfaction of the positive rule body under some assignment constructible over the atoms which are possibly derivable so far. Obviously, the least fixpoint $G_\Pi^\infty(\emptyset)$ of this operator is a subset of $grnd(\Pi)$; we will show that it is finite if Π is de-safe according to our new notion. Moreover, we will show that this grounding preserves all answer sets because all rule instances which are not added have unsatisfied bodies anyway.

Example 5 *Consider the following program Π:*

$$r_1\colon s(a). \quad r_2\colon dom(ax). \quad r_3\colon dom(axx).$$
$$r_4\colon s(Y) \leftarrow s(X), \&cat[X,x](Y), dom(Y).$$

The least fixpoint of G_Π is the following ground program:

$$r_1'\colon \quad s(a). \quad r_2'\colon dom(ax). \quad r_3'\colon dom(axx).$$
$$r_4'\colon \quad s(ax) \leftarrow s(a), \&cat[a,x](ax), dom(ax).$$
$$r_5'\colon s(axx) \leftarrow s(ax), \&cat[ax,x](axx), dom(axx).$$

Rule r_4' is added in the first iteration and rule r_5' in the second.

Towards a definition of de-safety, we say that a term in a rule is *bounded*, if the number of substitutions in $G_\Pi^\infty(\emptyset)$ for this term is finite. This is abstractly formalized using *term bounding functions* (TBFs).

Definition 6 (Term Bounding Function (TBF)) *A* term bounding function $b(\Pi, r, S, B)$ *maps a program Π, a rule $r \in \Pi$, a set S of already safe attributes, and a set B of already bounded terms in r to an enlarged set of bounded terms $b(\Pi, r, S, B) \supseteq B$, such that every $t \in b(\Pi, r, S, B)$ has finitely many substitutions in $G_\Pi^\infty(\emptyset)$ if (i) the attributes S have a finite range in $G_\Pi^\infty(\emptyset)$ and (ii) each term in $terms(r) \cap B$ has finitely many substitutions in $G_\Pi^\infty(\emptyset)$.*

Intuitively, a TBF receives a set of already bounded terms and a set of attributes that are already known to be de-safe. Taking the program into account, the TBF then identifies and returns further terms which are also bounded.

Our concept yields de-safety of attributes and programs from the boundedness of variables according to a TBF. We provide a mutually inductive definition that takes the empty set of de-safe attributes $S_0(\Pi)$ as its basis. Then, each iteration step $n \geq 1$ defines first the set of bounded terms $B_n(r, \Pi, b)$ for all rules r, and then an enlarged set of de-safe attributes $S_n(\Pi)$. The set of de-safe attributes in step $n+1$ thus depends on the TBF, which in turn depends on the domain-expansion safe attributes from step n.

Definition 7 ((Liberal) Domain-expansion Safety) *Let b be a TBF. The set of* bounded terms *$B_n(r, \Pi, b)$ in a rule $r \in \Pi$ in step $n \geq 1$ is defined as $B_n(r, \Pi, b) = \bigcup_{j \geq 0} B_{n,j}(r, \Pi, b)$ where $B_{n,0}(r, \Pi, b) = \emptyset$ and for $j \geq 0$, $B_{n,j+1}(r, \Pi, b) = b(\Pi, r, S_{n-1}(\Pi), B_{n,j})$.*

The set of domain-expansion safe attributes $S_\infty(\Pi) = \bigcup_{i\geq 0} S_i(\Pi)$ *of a program* Π *is iteratively constructed with* $S_0(\Pi) = \emptyset$ *and for* $n \geq 0$*:*

- $p{\restriction}i \in S_{n+1}(\Pi)$ *if for each* $r \in \Pi$ *and atom* $p(t_1, \ldots, t_{ar(p)}) \in H(r)$, $t_i \in B_{n+1}(r, \Pi, b)$*, i.e.,* t_i *is* bounded*;*
- $\&g[\mathbf{X}]_r{\restriction}_\mathrm{I} i \in S_{n+1}(\Pi)$ *if each* $\mathbf{X}_i$ *is a* bounded *variable, or* $\mathbf{X}_i$ *is a predicate input parameter* p *and* $p{\restriction}1, \ldots, p{\restriction}ar(p) \in S_n(\Pi)$*;*
- $\&g[\mathbf{X}]_r{\restriction}_\mathrm{O} i \in S_{n+1}(\Pi)$ *if and only if* r *contains an external atom* $\&g[\mathbf{X}](\mathbf{Y})$ *such that* $\mathbf{Y}_i$ *is bounded, or* $\&g[\mathbf{X}]_r{\restriction}_\mathrm{I} 1, \ldots, \&g[\mathbf{X}]_r{\restriction}_\mathrm{I} ar_\mathrm{I}(\&g) \in S_n(\Pi)$.

A program Π *is* (liberally) domain-expansion safe*, if it is safe and all its attributes are domain-expansion safe.*

An example is delayed until we have introduced concrete TBFs in Section 4. However, the intuition is as follows. In each step, the TBF first provides further terms that are bounded (given the information assembled in previous iterations), exploiting e.g. syntactic or semantic criteria. This possibly makes additional attributes de-safe (cf. the conditions for $S_n(\Pi)$ in Definition 7 above), which in turn may cause further terms to become bounded in the next iteration step.

One can show that $S_\infty(\Pi)$ is finite, thus the inductive definition can be used for computing $S_\infty(\Pi)$: the iteration can be aborted after finitely many steps. We first note some desired properties.

Proposition 1 *The set* $S_\infty(\Pi)$ *is finite.*

Proof (Sketch). There are only finitely many ordinary and external predicates with finite (input and output) arity. □

Moreover, de-safe attributes have a finite range in $G^\infty_\Pi(\emptyset)$.

Proposition 2 *For every TBF* b *and* $n \geq 0$*, if* $\alpha \in S_n(\Pi)$*, then the range of* α *in* $G^\infty_\Pi(\emptyset)$ *is finite.*

Proof (Sketch). The proof uses a double induction, following the mutual dependency of $S_n(\Pi)$ and $B_n(r, \Pi, b)$. The outer induction shows that $S_{n+1}(\Pi)$ are de-safe, using as (outer) induction hypothesis that $S_n(\Pi)$ are de-safe. This allows for proving that for all $t \in B_{n+1}(r, \Pi, b)$ the set of ground instances of r in $G^\infty_\Pi(\emptyset)$ contains only finitely many different substitutions for t. This is again done by induction: We show for each $j \geq 0$, if $t \in B_{n+1,j+1}(r, \Pi, b)$, then there are only finitely many substitutions for t in $G^\infty_\Pi(\emptyset)$, assuming that it already holds for all $t \in B_{n+1,j}(r, \Pi, b)$ (inner induction hypothesis). This essentially follows from the properties of TBFs. For the induction step of the outer induction, the different cases in the definition of liberal de-safety are exploited to show the desired property. □

Corollary 1 *If* $\alpha \in S_\infty(\Pi)$*, then* $range(\alpha, G^\infty_\Pi(\emptyset))$ *is finite.*

This means that such attributes occur with only finitely many arguments in the grounding computed by G_Π. This result implies that also the whole grounding $G^\infty_\Pi(\emptyset)$ is finite.

Corollary 2 *If* Π *is a de-safe program, then* $G^\infty_\Pi(\emptyset)$ *is finite.*

Proof (Sketch). Since Π is de-safe by assumption, $a \in S_\infty(\Pi)$ for all attributes a of Π. Then by Corollary 1, the range of all attributes of Π in $G^\infty_\Pi(\emptyset)$ is finite. But then there exists also only a finite number of ground atoms in $G^\infty_\Pi(\emptyset)$. Therefore the grounding is finite. □

As follows from these propositions, $S_\infty(\Pi)$ is also finitely constructible. Note that the propositions hold independently of a concrete TBF, because the properties of TBFs are sufficiently strong. This allows for a modular exchange or combination of the TBFs, as long as the preconditions of TBFs are satisfied, without changing the definition of de-safety.

4 Concrete Term Bounding Functions

We now introduce concrete term bounding functions that exploit syntactic and semantic properties of external atoms to guarantee boundedness of variables. By our previous result, this ensures also finiteness of the ground program given by $G^\infty_\Pi(\emptyset)$.

1. Syntactic Criteria. We first identify syntactic properties that can be exploited for our purposes.

Definition 8 (Syntactic Term Bounding Function) *We define* $b_{syn}(\Pi, r, S, B)$ *such that* $t \in b_{syn}(\Pi, r, S, B)$ *iff*

(i) t *is a constant in* r*; or*

(ii) there is an ordinary atom $q(s_1, \ldots, s_{ar(q)}) \in B^+(r)$ *such that* $t = s_j$*, for some* $1 \leq j \leq ar(q)$ *and* $q{\restriction}j \in S$*; or*

(iii) for some external atom $\&g[\mathbf{X}](\mathbf{Y}) \in B^+(r)$*, we have that* $t = Y_i$ *for some* $Y_i \in \mathbf{Y}$*, and for each* $X_i \in \mathbf{X}$*,*

$$\begin{cases} X_i \in B, & \text{if } \tau(\&g, i) = \mathbf{const}, \\ X_i{\restriction}1, \ldots, X_i{\restriction}ar(X_i) \in S, & \text{if } \tau(\&g, i) = \mathbf{pred}. \end{cases}$$

Intuitively, (i) a constant is trivially bounded because it is never substituted by other terms in the grounding. Case (ii) states that terms at de-safe attribute positions are bounded; more spcifically, the fact that an attribute $q{\restriction}j$ (where $1 \leq j \leq ar(q)$) is de-safe, and thus has a finite range in $G^\infty_\Pi(\emptyset)$, implies that the term at this attribute position is bounded. Case (iii) essentially expresses that if the input to an external atom is finite, then also its output is finite.

Lemma 3 *Function* $b_{syn}(\Pi, r, S, B)$ *is a TBF.*

Proof (Sketch). If t is in the output of $b_{syn}(\Pi, r, S, B)$, then one of the the conditions holds. If condition (i) holds, then t is a constant, thus there is only one instance. If condition (ii) holds, then t occurs as value for $q{\restriction}j$, which has a finite range by assumption. If condition (iii) holds, then t is output of an external atom such that its input is finite. Thus there are only finitely many oracle calls with finite output each. □

Example 6 (cont'd) *Consider* Π *from Example 5. We get* $S_1(\Pi) = \{dom{\restriction}1, \&cat[X, x]_{r_4}{\restriction}_\mathrm{I} 2\}$*, as* $B_1(r_2, \Pi, b_{syn}) = \{ax\}$*,* $B_1(r_3, \Pi, b_{syn}) = \{axx\}$ *and* $B_1(r_4, \Pi, b_{syn}) = \{x\}$ *(by item (i) in Definition 8), i.e., the derived terms in all rules that have* $dom{\restriction}1$ *in their head are known to be bounded. In the next iteration, we get* $B_2(r_4, \Pi, b_{syn}) = \{Y\}$ *(by item (ii) in Definition 8) as* $dom{\restriction}1$ *is already known to be de-safe. Since we also have* $B_2(r_1, \Pi, b_{syn}) = \{a\}$*, the terms derived by* r_1 *and* r_4 *are bounded, hence* $s{\restriction}1 \in S_2(\Pi)$*. Moreover,* $\&cat[X, x]_{r_4}{\restriction}_\mathrm{O} 1 \in S_2(\Pi)$ *because* Y *is bounded. The third iteration yields* $\&cat[X, x]_{r_4}{\restriction}_\mathrm{I} 1 \in S_3(\Pi)$ *because* $X \in B_3(r_4, \Pi, b_{syn})$ *due to item (ii) in Definition 8. Thus, all attributes are de-safe.* □

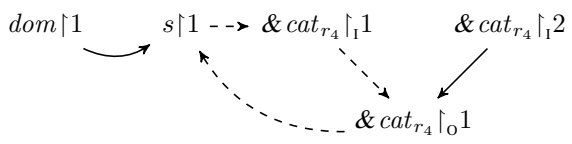

Figure 1: Attribute dependency graph for Π in Example 5

2. Semantic Properties. We now define a TBF exploiting meta-information about external sources in three properties.

The first property is based on *malign cycles* in *attribute dependency graphs*, which are the source of any infinite value invention. For space reasons, we omit here lengthy formal definitions and focus on an intuitive understanding.

The *attribute dependency graph* $G_A(\Pi)$ has as nodes the attributes of Π and its edges model the information flow between the attributes. For instance, if for rule r we have $p(\mathbf{X}) \in H(r)$ and $q(\mathbf{Y}) \in B^+(r)$ such that $X_i = Y_j$ for some $X_i \in \mathbf{X}$ and $Y_j \in \mathbf{Y}$, then we have a flow from $q\restriction j$ to $p\restriction i$.

Example 7 (cont'd) *The attribute dependency graph $G_A(\Pi)$ of program Π from Example 5 is shown in Figure 1.*

Definition 9 (Benign and Malign Cycles) *A cycle K in $G_A(\Pi)$ is* benign *w.r.t. a set of safe attributes S, if there exists a well-ordering $\leq_{\mathcal{C}}$ of $\mathcal{C}$, such that for every $\&g[\mathbf{X}]_r\restriction_{\mathrm{O}} j \notin S$ in the cycle, $f_{\&g}(\mathbf{A}, x_1, \ldots, x_m, t_1, \ldots t_n) = 0$ whenever*

- *some x_i for $1 \leq i \leq m$ is a predicate parameter, $\&g[\mathbf{X}]_r\restriction_{\mathrm{I}} i \notin S$ is in K, $(s_1, \ldots, s_{ar(x_i)}) \in ext(\mathbf{A}, x_i)$, and $t_j \not\leq_{\mathcal{C}} s_k$ for some $1 \leq k \leq ar(x_i)$; or*
- *some x_i for $1 \leq i \leq m$ is a constant input parameter, $\&g[\mathbf{X}]_r\restriction_{\mathrm{I}} i \notin S$ is in K, and $t_j \not\leq_{\mathcal{C}} x_i$.*

A cycle in $G_A(\Pi)$ is called malign *w.r.t. S if it is not benign.*

Intuitively, a cycle is benign if external atoms never deliver larger values w.r.t. to their yet unsafe cyclic input. As there is a least element, this ensures a finite grounding.

Example 8 (cont'd) *The cycle in $G_A(\Pi)$ (dashed lines in Figure 1) is malign w.r.t. $S = \emptyset$ because there is no well-ordering as required by Definition 9. Intuitively, this is because the external atom infinitely extends the string.*

If we replace $\&cat[X, x](Y)$ in Π by $\&tail[X](Y)$, i.e., we compute the string Y from X with the first character removed, then the cycle in the adapted attribute dependency graph becomes benign using $<$ over the string lengths as well-ordering. □

Two other properties involve meta-information that directly ensures an output attribute of an external source is finite.

Definition 10 (Finite Domain) *An external predicate $\&g \in \mathcal{X}$ has the* finite domain property w.r.t. output argument $i \in \{1, \ldots, ar_{\mathrm{O}}(\&g)\}$, *if $\{y_i \mid \mathbf{x} \in (\mathcal{P} \cup \mathcal{C})^{ar_{\mathrm{I}}(\&g)}, \mathbf{y} \in \mathcal{C}^{ar_{\mathrm{O}}(\&g)}, f_{\&g}(\mathbf{A}, \mathbf{x}, \mathbf{y}) = 1\}$ is finite for all assignments $\mathbf{A}$.*

Here, the provider of the external source explicitly states that the output at a certain position in the output tuple is finite. This is perhaps the most direct way to ensure boundedness of the respective term.

Example 9 *An external atom $\&md5[S](Y)$ computing the MD5 hash value Y of a string S is finite domain w.r.t. the (single) output element, as its domain is finite (yet very large).* □

While the previous properties derive boundedness of an output term of an external atom from finiteness of its input, we now reverse the direction. An external atom may have the property that only a finite number of different inputs can yield a certain output, which is formalized as follows.

Definition 11 (Finite Fiber) *An external predicate $\&g \in \mathcal{X}$ has the* finite fiber property, *if $\{\mathbf{x} \mid \mathbf{x} \in (\mathcal{P} \cup \mathcal{C})^{ar_{\mathrm{I}}(\&g)}, f_{\&g}(\mathbf{A}, \mathbf{x}, \mathbf{y}) = 1\}$ is finite for every $\mathbf{A}$ and $\mathbf{y} \in \mathcal{C}^{ar_{\mathrm{O}}(\&g)}$.*

Example 10 *Let $\&square[X](S)$ be an external atom that computes the square number S of X. Then for a given S, there are at most two distinct values for X.* □

The three properties above lead to the following TBF.

Definition 12 (Semantic Term Bounding Function) *We define $b_{sem}(\Pi, r, S, B)$ such that $t \in b_{sem}(\Pi, r, S, B)$ iff*

(i) t is captured by some attribute α in $B^+(r)$ that is not reachable from malign cycles in $G_A(\Pi)$ w.r.t. S, i.e., if $\alpha = p\restriction i$ then $t = t_i$ for some $p(t_1, \ldots, t_\ell) \in B^+(r)$, and if $\alpha = \&g[\mathbf{X}]_r\restriction_T i$ then $t = X_i^T$ for some $\&g[\mathbf{X}^{\mathrm{I}}](\mathbf{X}^{\mathrm{O}}) \in B^+(r)$ where $\mathbf{X}^T = X_1^T, \ldots, X_{ar(\&g)}^T$; or

(ii) $t = Y_i$ for some $\&g[\mathbf{X}](\mathbf{Y}) \in B^+(r)$, where $\&g$ has the finite domain property in i; *or*

(iii) $t \in \mathbf{X}$ for some $\&g[\mathbf{X}](\mathbf{Y}) \in B^+(r)$, where $U \in B$ for every $U \in \mathbf{Y}$ and $\&g$ has the finite fiber property.

This TBF is directly motivated by the properties introduced above.

Lemma 4 *Function $b_{sem}(\Pi, r, S, B)$ is a TBF.*

Proof (Sketch). If t is in the output of $b_{syn}(\Pi, r, S, B)$, then one of the the conditions holds. If condition (i) holds, then there is no information flow from malign cycles wrt. S to t. Such cycles are the only source of infinite groundings: the attributes in S have a finite domain by assumption. For the remaining attributes in the cycle, the the well-ordering guarantees that only finitely many different values can be produced. If condition (ii) holds, then the claim follows from finiteness of the domain of the external atom. If condition (iii) holds, there are only finitely many substitutions for t because the output of the respective external atom is bound by precondition of TBFs and the finite fiber ensures that there are only finitely many different inputs for each output. □

For an attractive framework it is important that a certain degree of flexibility is achieved in terms of composability of TBFs. The following proposition allows us to construct TBFs modularly from multiple TBFs and thus ensures future extensibility by, e.g., customized application-specific TBFs.

Proposition 3 *If $b_i(\Pi, r, S, B)$, $1 \leq i \leq \ell$, are TBFs, then $b(\Pi, r, S, B) = \bigcup_{1 \leq i \leq \ell} b_i(\Pi, r, S, B)$ is a TBF.*

Proof (Sketch). For $t \in b(\Pi, r, S, B)$, $t \in b_i(\Pi, r, S, B)$ for some $1 \leq i \leq \ell$. Then there are only finitely many substitution for t in $G_\Pi^\infty(\emptyset)$ because b_i is a TBF. □

In particular, a TBF which exploits syntactic and semantic properties simultaneously is

$$b_{s^2}(\Pi, r, S, B) = b_{syn}(\Pi, r, S, B) \cup b_{sem}(\Pi, r, S, B),$$

which we will use subsequently.

5 Finite Restrictability

We now make use of the results from above to show that de-safe programs are finitely restrictable in an effective manner.

Proposition 4 *Let Π be a de-safe program. Then Π is finitely restrictable and $G^\infty_\Pi(\emptyset) \equiv^{pos} \Pi$.*

Proof (Sketch). We construct the grounding $grnd_C(\Pi)$ as the least fixpoint $G^\infty_\Pi(\emptyset)$ of $G_\Pi(X)$, which is finite by Corollary 2. The set C is then given by the set of constants appearing in $grnd_C(\Pi)$. It remains to show that indeed $grnd_C(\Pi) \equiv^{pos} grnd_{\mathcal{C}}(\Pi)$. We show the more general proposition $grnd_C(\Pi) \equiv^{pos} grnd_{C'}(\Pi)$ for any $C' \supseteq C$.
($\Rightarrow$) One can show that for any $\mathbf{A} \in \mathcal{AS}(grnd_C(\Pi))$, the bodies of all additional rules in $r \in grnd_{C'}(\Pi)$ are unsatisfied, thus $\mathbf{A}$ is a model of $grnd_{C'}(\Pi)$. Moreover, $fgrnd_C(\Pi)^{\mathbf{A}} = fgrnd_{C'}(\Pi)^{\mathbf{A}'}$, thus it is an answer set.
($\Leftarrow$) For any $\mathbf{A} \in \mathcal{AS}(grnd_{C'}(\Pi))$, its restriction $\mathbf{A}'$ to the atoms in $grnd_C(\Pi)$ is an answer set of $grnd_C(\Pi)$. If $\mathbf{A}'$ is again extended to $grnd_{C'}(\Pi)$ by setting all additional atoms to false, this yields another model of $grnd_{C'}(\Pi)$. Then minimality of answer sets implies $\mathbf{A} = \mathbf{A}'$. □

This proposition holds independently of a concrete term bounding function. However, too liberal functions are excluded by the preconditions in the definition of TBFs.

Although the design of an algorithm which actually computes a finite grounding is ongoing work and our of the scope of this paper, we briefly sketch optimizations of the grounding operator as basis for an efficient algorithm. The operator G_Π is exponential in the number of ground atoms as it considers all assignments $\mathbf{A} \subseteq \mathcal{A}(\Pi')$ in every step. As this compromises efficiency, a better alternative is

$$R_\Pi(\Pi') = \bigcup_{r \in \Pi} \left\{ r\theta \mid \{\mathbf{T}a \mid a \in A(\Pi')\} \models B^+(r\theta) \right\} \quad .$$

Intuitively, instead of enumerating exponentially many assignments it simply maximizes the output of external atoms by setting all input atoms to true, which is possible due to monotonicity.

Proposition 5 *Let Π be a de-safe program such that each nonmonotonic input parameter to external atoms occurs only in facts. Then $G^\infty_\Pi(\emptyset) = R^\infty_\Pi(\emptyset)$.*

Proof (Sketch). The proposition follows from monotonicity of all external atoms in all atoms except facts because the assignment $\mathbf{A}'$ maximizes the external atom output. □

Thus, for such programs we may compute a sufficient finite subset of $grnd(\Pi)$ using instead G_Π the more efficient R_Π.

Example 11 (cont'd) *In Example 5, $\&cat[X,x](Y)$ is monotonic, hence we can use R_Π for restricted grounding.* □

The operator can also be optimized in a different way. External atoms that are not relevant for de-safety can be removed from the fixpoint iteration without affecting correctness of the grounding. For each $r \in \Pi$, let $\bar{r} = H \leftarrow B$ be any rule such that $r = H \leftarrow b_1, \ldots, b_h, B$ where $b_1, \ldots, b_h \in EA(r)$, i.e., $\bar{r}$ results from r by possibly dropping external atoms from $B^+(r)$, and let $\bar{\Pi} = \{\bar{r} \mid r \in \Pi\}$.

We then define the following monotone operator:

$$Q_\Pi(\Pi') = \bigcup_{r \in \Pi} \{r\theta \mid \exists \mathbf{A} \subseteq \mathcal{A}(\Pi'), \mathbf{A} \not\models \bot, \mathbf{A} \models B^+(\bar{r}\theta)\}.$$

The intuition is that removing atoms from rule bodies makes rule applicability (possibly) more frequent, which may result in a larger (but still finite) grounding. As this grounding is a superset of the one computed by $G_\Pi(\Pi')$, it is still answer set preserving.

Proposition 6 *For every program Π, $Q^\infty_\Pi(\emptyset) \subseteq grnd(\Pi)$ is finite and $Q^\infty_\Pi(\emptyset) \equiv^{pos} G^\infty_\Pi(\emptyset)$.*

Proof (Sketch). Since $\bar{\Pi}$ is still de-safe by assumption, $G^\infty_{\bar{\Pi}}(\emptyset)$ is still finite by Proposition 4. Moreover, since $Q^\infty_\Pi(\emptyset) \supseteq G^\infty_\Pi(\emptyset)$, $\mathcal{AS}(Q^\infty_\Pi(\emptyset)) \supseteq \mathcal{AS}(G^\infty_\Pi(\emptyset))$. □

Example 12 (cont'd) *In the program in Example 5, the external atom $\&cat[X,x](Y)$ is not needed to establish de-safety, hence we might drop it during fixpoint iteration.* □

The *combination* of the optimizations is especially valuable. One can first eliminate external atoms with nonmonotonic input other than facts and check then rule body satisfaction as in R_Π. If an external atom b is strongly safe w.r.t. to the according rule and the program, then it is very often (as in Example 5) not necessary for establishing de-safety and b is a candidate for being removed. That is, the traditional strong safety criterion is now used as a *weak criterion*, which is not strictly necessary but may help reduce grounding time.

6 Implementation and Application

We implemented the framework and the concrete TBFs from above in the open-source solver DLVHEX. The system provides an interface to external source developers, which allows for specifying the semantics of the source, and meta-information which is used for deciding de-safety.

We stress that the main goal of this paper and the implementation was not performance enhancement of the grounding procedure, but checking liberal de-safety of programs. Thus, in context of this work the only reasonable benchmark experiment would measure the overhead introduced by the checking algorithm, which we disregard because the costs are neglectable compared to grounding and solving. The development of an actual grounding algorithm is ongoing work, but is based on (an optimization of) the operators introduced in this paper.

Obvious applications that need recursion through external atoms and thus benefit from our result are, e.g., recursive access of web resources and recursive query processing. We discuss here a different one in advanced parsing.

Example: Pushdown Automaton. We model a *pushdown automaton* in a HEX-program, which can be of use if context-free languages must be parsed under further constraints that cannot be easily expressed in the production rules; the HEX-program may be extended accordingly, where the declarative nature of HEX is versatile for parsing and constraint checking in parallel as opposed to a generate-and-filter approach.

For instance, consider RNA sequences over the alphabet $\{a, g, c, u\}$ and suppose we want to accept all sequences ww' such that w' is the complementary string of w, where (a, u)

and (g, c) are complementary pairs. Because complementary strings within one sequence influence the secondary structure of an RNA molecule, this duality is important for its proper function (Zuker and Sankoff 1984). This language is easily expressed by the production rules

$$\{S \to aSu, S \to uSa, S \to gSc, S \to cSg, S \to \epsilon\}$$

with start symbol S. Now suppose that we want to check in addition the occurrence of certain subsequences, e.g., because they have a known function. A concrete example would be a *promotor sequence*, which identifies the starting location of a new gene and might be used to separate coding and non-coding sequences. Modeling this in the production rules makes the grammar considerably more complex. Moreover, we might want to keep the grammar independent of concrete subsequences but import them from a database. Then it might be useful to model the basic language as automaton in a logic program and check side conditions by additional constraints.

Recall that a pushdown automaton is a finite-state machine with an additional stack, cf. Sipser (2012); formally, it is a tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z, F)$, where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite stack alphabet, $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \times Q \times \Gamma^*$ is the transition relation, $q_0 \in Q$ is the initial state, $Z \in \Gamma$ is the initial stack symbol, and $F \subseteq Q$ is the set of final states. The transition relation maps the current state, an input symbol and the topmost stack symbol to a successor state and a finite word over the stack alphabet, which is pushed onto the stack after removing the topmost symbol. We assume for simplicity that there are no ϵ-transitions, i.e., $\delta \subseteq Q \times \Sigma \times \Gamma \times Q \times \Gamma^*$; such an automaton is easily obtained from a normalized grammar, if one is not interested, as in our example, in the empty word.

We use the following external atoms:

- $\&\,car[S](H, T)$ splits S into first symbol H and rest T;
- $\&\,cat[A, B](C)$ joins A and B to C;
- $\&\,inc[I](I1)$ increments the integer I to $I1 = I{+}1$;
- $\&\,len[S](L)$ returns the length L of string S.

Then the automaton can be modelled as follows:

$$str(Word, 0) \leftarrow input(Word). \quad (1)$$

$$str(R, I1) \leftarrow str(W, I), \&\,car[W](C, R), \&\,inc[I](I1). \quad (2)$$

$$char(C, I) \leftarrow str(W, I), \&\,car[W](C, R). \quad (3)$$

$$in(start, z, 0). \quad (4)$$

$$in(NewState, NewStack, NewPos) \leftarrow \quad (5)$$
$$in(State, Stack, Pos), char(Pos, Char),$$
$$\&\,car[Stack](SChar, SRest),$$
$$transition(State, Char, SChar, NewState, Push),$$
$$\&\,cat[Push, SRest](NewStack), \&\,inc[Pos](NewPos).$$

$$accept \leftarrow input(W), \&\,len[W](L), in(S, z, L), final(S). \quad (6)$$

$$\leftarrow \text{not } accept. \quad (7)$$

An atom $in(state, stack, step)$ encodes that when processing symbol pos, the machine is in state $state$ with stack content $stack$. Rules (1)-(3) split the input string into characters, and the remaining ones model the automaton. The program starts in the initial state $start$ with the initial stack symbol z as stack content (fact (4)). Transition rule (5) splits the current stack content into its topmost symbol $SChar$ and its rest $SRest$ and uses the predicate $transition$ to (nondeterministically) determine the successor state and the string to push onto the stack. The rules (6)-(7) ensure that the input is accepted if eventually a final state is reached such that the input has been completely processed and the stack content is z. Side conditions can now be modeled, e.g., by additional constraints which restrict the stack content, or by additional body atoms in the transition rule.

The program is not strongly safe as all external atoms occur in cycles and their output is not bounded by ordinary atoms from outside. However, it is de-safe if we exploit semantical information. String splitting with $\&\,car$ yields ϵ or a shorter string, i.e., a well-ordering exists. Hence the output terms of $\&\,car$ are safe by Proposition 4 due to Definition 12(i). Each transition step pushes a finite word onto the stack, and only finitely many steps happen (as no ϵ-transitions occur); hence only finitely many stack contents are possible, i.e., $\&\,cat$ has a finite output domain. Thus the output terms are safe due to Definition 12(ii). The domain of $\&\,inc$ is finite for the same reason, which bounds $NewPos$. Hence, all variables are bounded and all attributes are domain-expansion safe.

Further Applications. We now briefly discuss other applications which exploit de-safety.

Declarative processing of recursive data structures, such as trees or lists, can easily violate traditional safety criteria. However, in a concrete program the use of the external sources may satisfy syntactic or semantic conditions such that finiteness of the grounding is still guaranteed. For instance, if a list is only subdivided but not recursively extended, then there is a well-ordering as defined above and the grounding may be finite. Additional application-specific safety criteria can be straightforwardly integrated into our framework by customized term bounding functions.

Another application is route planning. Importing a complete street map into the logic program a priori might be too expensive due to the large amount of data. The alternative is to query direct connections between nodes in a recursive fashion. But if the set of nodes is not known in advance, then such queries do not satisfy traditional strong safety. However, since maps are finite, our notion of de-safety helps ensuring the existence of a finite grounding.

7 Related Work

Our notion of liberal de-safety using b_{s^2} compares to the traditionally used strong de-safety and to other formalizations

Strong Safety. One can now show that (liberal) de-safety is strictly less restrictive than strong de-safety.

Proposition 7 *Every strongly de-safe program* Π *is de-safe.*

Proof (Sketch). One can show that all attributes a of a strongly safe program are de-safe. This is done by induction on the number j of malign cycles wrt. $\emptyset$ in $G_A(\Pi)$ from which a is reachable. One can then show both for the base case $j = 0$ and for the induction step $j \mapsto j{+}1$ that whenever one of the conditions of strong safety applies for an external atom, then the properties of syntactic and semantic TBFs guarantee that the corresponding attributes are de-safe. □

The converse does not hold, as there are de-safe programs that are not strongly safe, cf. Example 3.

VI-Restricted Programs. Calimeri et al. (2007) introduced the notion of *VI-restrictedness* for *VI programs*, which amount to the class of HEX-programs in which all input parameters to external atoms are of type **const**. Their notion of attribute dependency graph is related to ours, but our notion is more fine-grained for attributes of external predicates. While we use a node $\&g[\mathbf{X}]_r\restriction_T i$ for each external predicate $\&g$ with input list $\mathbf{X}$ in a rule r and $T \in \{\mathrm{I}, \mathrm{O}\}$, $1 \leq i \leq ar_T(\&g)$, Calimeri et al. use just one attribute $\&g\restriction i$ for each $i \in \{1, \ldots, ar_{\mathrm{I}}(\&g)+ar_{\mathrm{O}}(\&g)\}$ independent of $\mathbf{X}$. Thus, neither multiple occurrences of $\&g$ with different input lists in a rule, nor of the same attribute in multiple rules are distinguished; this collapses distinct nodes in our attribute dependency graph into one. Using b_{s^2}, we can show:

Proposition 8 *Every VI-restricted program Π is de-safe.*

Proof (Sketch). The key idea is to redefine the definitions of *blocking* and *savior attributes* inductively, such that the step in which an attribute becomes blocked or savior is identified by an integer; this is not the case in the original definitions in Calimeri, Cozza, and Ianni (2007). Then the rest of the proof is an induction on the step number in which an attribute becomes blocked or savior. One can then show that for each such attribute, the syntactic and semantic TBFs guarantee that it will be declared de-safe in finitely many steps. □

The converse does not hold, as there are de-safe VI-programs (due to semantic criteria) that are not VI-restricted.

Logic Programs with Function Symbols. Syrjänen (2001) defined ω-restricted logic programs, which allow function symbols under a level mapping to control the introduction of new terms with function symbols to ensure decidability. Calimeri et al. (2007) observe that such programs Π can be rewritten to VI-programs $F(\Pi)$ using special external predicates that compose/decompose terms from/into function symbols and a list of arguments, such that $F(\Pi)$ is VI-restricted. As every VI-restricted program, viewed as a HEX-program, is by Proposition 8 also de-safe, we obtain:

Proposition 9 *If Π is ω-restricted, then $F(\Pi)$ is de-safe.*

Proof (Sketch). By Theorem 7 in Calimeri, Cozza, and Ianni (2007) $F(\Pi)$ is VI-restricted, and thus by Proposition 8 also de-safe using $b_{synsem}(\Pi, r, S, B)$. □

Since de-safety is strictly more liberal than VI-restrictedness, it is also more liberal than ω-restrictedness.

More expressive variants of ω-restricted programs are λ-restricted (Gebser, Schaub, and Thiele 2007) and argument-restricted programs (Lierler and Lifschitz 2009). They can be captured within our framework as well, but argument-restricted programs Π exist such that $F(\Pi)$ is *not* de-safe w.r.t. b_{s^2}. The reason is that specific properties of the external atoms for term (de)composition are exploited (while our approach uses general external sources). However, tailored TBFs can be used (which shows the flexibility of our modular approach).

Similarly, i.e., by means of dedicated external atoms for (de)composing terms and a specialized TBF, so-called FD programs (Calimeri et al. 2008) map into our framework. Finitary programs (Bonatti 2004; 2002) and FG programs (Calimeri et al. 2008), however, differ more fundamentally from our approach and cannot be captured as de-safe w.r.t. appropriate TBFs, as they are not effectively recognizable (and the former, in general, not even finitely restrictable).

Term Rewriting Systems. A term rewriting system is a set of rules for rewriting terms to other terms. Termination is usually shown by proving that the right-hand side of every rule is strictly smaller than its left-hand side (Zantema 1994; 2001). Our notion of benign cycles is similar, but different from term rewriting systems the values do not need to *strictly* decrease. While terms that stay equal may prevent termination in term rewriting systems, they do not harm in our case because they cannot expand the grounding infinitely.

Other Notions of Safety. Related to our semantic properties are Sagiv and Vardi (1989), Ramakrishnan et al. (1987), and Krishnamurthy et al. (1996). They exploit finiteness of attributes (cf. item (ii) in Definition 12) in sets of Horn clauses and derive finiteness of further attributes using *finiteness dependencies*. This is related to item (iii) in Definition 12 and item (iii) in Definition 8. However, they do this not for model building but for query answering over infinite databases.

Less related to our work are Cabalar et al. (2009), Lee et al. (2008), and Bartholomew and Lee (2010), which extend safety, resp. argument restrictedness, to arbitrary first-order formulas with(out) function symbols under the stable model semantics, rather than generalizing the concepts.

8 Conclusion

We presented a framework for obtaining classes of HEX-programs (ASP programs with external sources) that allow for finite groundings sufficient for evaluation over an infinite domain (which arises by value invention in calls of external sources). It is based on term bounding functions (TBFs) and enables modular exchange and enhancement of such functions, and combining hitherto separate syntactic and semantic criteria into a single notion of *liberal domain expansion safety*. Our work pushes the classes of HEX-program with evaluation via finite grounding considerably, leading to strictly larger classes than available via well-known criteria for answer set programs over infinite domains. We provided two concrete TBFs that capture syntactic criteria similar to but more fine-grained than ones in Calimeri et al. (2007), and semantic criteria related to Sagiv and Vardi (1989) and Ramakrishnan et al. (1987), but targeting model generation (not query answering). An implementation for DLVHEX is available.

Issues for ongoing and future work are the identification of further TBFs and suitable well-orderings of domains in practice. Of particular interest are external atoms that provide built-in functions and simulate, in a straightforward manner, function symbols. On the implementation side, further refinement and optimizations are an issue, as well as a library of TBFs and a plugin architecture that supports creating customized TBFs, to make our framework more broadly usable.

References

Bartholomew, M., and Lee, J. 2010. A decidable class of groundable formulas in the general theory of stable models. In *12th International Conference on the Principles of Knowledge Representation and Reasoning (KR'10)*, 477–485. AAAI Press.

Bonatti, P. A. 2002. Reasoning with Infinite Stable Models II: Disjunctive Programs. In *18th International Conference on Logic Programming (ICLP'02)*, volume 2401 of *LNCS*, 333–346. Springer.

Bonatti, P. A. 2004. Reasoning with infinite stable models. *Artificial Intelligence* 156(1):75—111.

Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54(12):92–103.

Cabalar, P.; Pearce, D.; and Valverde, A. 2009. A revised concept of safety for general answer set programs. In *10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, volume 5753 of *LNCS*, 58–70. Springer.

Calimeri, F.; Cozza, S.; Ianni, G.; and Leone, N. 2008. Computable Functions in ASP: Theory and Implementation. In *24th International Conference on Logic Programming (ICLP'08)*, volume 5366 of *LNCS*, 407–424. Springer.

Calimeri, F.; Cozza, S.; and Ianni, G. 2007. External Sources of Knowledge and Value Invention in Logic Programming. *Annals of Mathematics and Artificial Intelligence* 50(3–4):333–361.

Eiter, T.; Ianni, G.; Schindlauer, R.; and Tompits, H. 2005. A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer-Set Programming. In *19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, 90–96. Professional Book Center.

Eiter, T.; Ianni, G.; Schindlauer, R.; and Tompits, H. 2006. Effective Integration of Declarative Rules with External Evaluations for Semantic-Web Reasoning. In *3rd European Semantic Web Conference (ESWC'06)*, volume 4011 of *LNCS*, 273–287. Springer.

Eiter, T.; Fink, M.; Ianni, G.; Krennwallner, T.; and Schüller, P. 2011. Pushing Efficient Evaluation of HEX Programs by Modular Decomposition. In *11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, volume 6645 of *LNCS*, 93–106. Springer.

Faber, W.; Leone, N.; and Pfeifer, G. 2011. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence* 175(1):278–298.

Gebser, M.; Schaub, T.; and Thiele, S. 2007. GrinGo: A New Grounder for Answer Set Programming. In *9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, volume 4483, 266–271. Springer.

Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing* 9(3–4):365–386.

Krishnamurthy, R.; Ramakrishnan, R.; and Shmueli, O. 1996. A framework for testing safety and effective computability. *Journal of Computer and System Sciences* 52(1):100–124.

Lee, J.; Lifschitz, V.; and Palla, R. 2008. Safe formulas in the general theory of stable models (preliminary report). In *24th International Conference on Logic Programming (ICLP'08)*, volume 5366 of *LNCS*, 672–676. Springer.

Lierler, Y., and Lifschitz, V. 2009. One more decidable class of finitely ground programs. In *25th International Conference on Logic Programming (ICLP'09)*, volume 5649 of *LNCS*, 489–493. Springer.

Ramakrishnan, R.; Bancilhon, F.; and Silberschatz, A. 1987. Safety of recursive horn clauses with infinite relations. In *6th Symposium on Principles of Database Systems (PODS'87)*, 328–339. ACM.

Sagiv, Y., and Vardi, M. Y. 1989. Safety of datalog queries over infinite databases. In *8th Symposium on Principles of Database Systems (PODS'89)*, 160–171. ACM.

Sipser, M. 2012. *Introduction to the Theory of Computation*. Course Technology, 3rd edition.

Syrjänen, T. 2001. Omega-restricted logic programs. In *6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01)*, volume 2173 of *LNCS*, 267–279. Springer.

Zantema, H. 1994. Termination of term rewriting: Interpretation and type elimination. *Journal of Symbolic Computation* 17(1):23–50.

Zantema, H. 2001. The termination hierarchy for term rewriting. *Applicable Algebra in Engineering, Communication and Computing* 12(1-2):3–19.

Zuker, M., and Sankoff, D. 1984. RNA secondary structures and their prediction. *Bulletin of Mathematical Biology* 46(4):591–621.