Grounding HEX-Programs with Expanding Domains

Thomas Eiter, Michael Fink, Thomas Krennwallner, Christoph Redl

{eiter,fink,tkren,redl}@kr.tuwien.ac.at

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

Knowledge-Based Systems Group

GTTV'13, Sep 15, 2013

Motivation

HEX-Programs

- Extend ASP by external sources
- Traditional safety criteria not sufficient: value invention
- Strong safety is unnecessarily restrictive
- Liberal domain-expansion safe HEX program are more flexible, but no effective algorithms exist yet

Example

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

Motivation

HEX-Programs

- Extend ASP by external sources
- Traditional safety criteria not sufficient: value invention
- Strong safety is unnecessarily restrictive
- Liberal domain-expansion safe HEX program are more flexible, but no effective algorithms exist yet

Contribution

- New iterative grounding algorithm for liberal safety criteria
- Based on a grounder for ordinary ASP programs
- Avoids the worst case for the algorithm using program decomposition

HEX-Programs

HEX-programs extend ordinary ASP programs by external sources

Definition (HEX-programs)

A HEX-program consists of rules of form

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

with classical literals a_i, and classical literals or an external atoms b_j.

Definition (External Atoms)

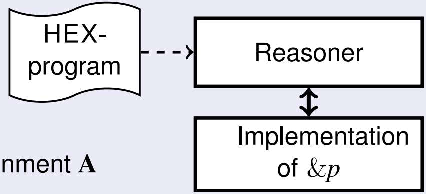
An external atom is of the form

$$\&p[q_1, \ldots, q_k](t_1, \ldots, t_l),$$

p . . . external predicate name

q_i . . . predicate names or constants

t_j . . . terms



Semantics:

$1 + k + l$-ary Boolean oracle function $f_{\&p}$:

$\&p[q_1, \ldots, q_k](t_1, \ldots, t_l)$ is true under assignment $\mathbf{A}$

iff $f_{\&p}(\mathbf{A}, q_1, \ldots, q_k, t_1, \ldots, t_l) = 1$.

Liberal Safety: Basic Concepts

Monotone Grounding Operator

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

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 instance of r under variable substitution $\theta\colon \mathcal{V} \to \mathcal{C}$.

Example

Program Π:

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

Least fixpoint $G^\infty_\Pi(\emptyset)$ of G_Π:

$$r'_1 : \ s(a). \quad r'_2 : \ dom(ax). \quad r'_3 : \ dom(axx).$$

Liberal Safety: Basic Concepts

Monotone Grounding Operator

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

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 instance of r under variable substitution $\theta : \mathcal{V} \to \mathcal{C}$.

Example

Program Π:

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

Least fixpoint $G^{\infty}_{\Pi}(\emptyset)$ of G_{Π}:

$$r'_1 : \; s(a). \quad r'_2 : \; dom(ax). \quad r'_3 : \; dom(axx).$$
$$r'_4 : \; s(ax) \leftarrow s(a), \&cat[a,x](ax), dom(ax).$$

Liberal Safety: Basic Concepts

Monotone Grounding Operator

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

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 instance of r under variable substitution $\theta \colon \mathcal{V} \to \mathcal{C}$.

Example

Program Π:

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

Least fixpoint $G^\infty_\Pi(\emptyset)$ of G_Π:

$$r'_1 : \; s(a). \quad r'_2 : \; dom(ax). \quad r'_3 : \; dom(axx).$$
$$r'_4 : \; s(ax) \leftarrow s(a), \&cat[a, x](ax), dom(ax).$$
$$r'_5 : s(axx) \leftarrow s(ax), \&cat[ax, x](axx), dom(axx).$$

Intuition: We call a program safe if this operator produces a finite grounding

Liberal Safety

Two concepts

- A term is bounded if $G_{\Pi}(\Pi')$ contains only finitely many substitutions for it
- An attribute is de-safe if $G_{\Pi}(\Pi')$ contains only finitely many values at this attribute position

Idea

Liberal Safety

Two concepts

- A term is bounded if $G_{\Pi}(\Pi')$ contains only finitely many substitutions for it
- An attribute is de-safe if $G_{\Pi}(\Pi')$ contains only finitely many values at this attribute position

Idea

1. Start with empty set of bounded terms B_0 and de-safe attributes S_0
2. For all $n \geq 0$ until B_n and S_n do not change anymore
   a. Identify additional bounded terms $\Rightarrow B_{n+1}$
      (assuming that B_n are bounded and S_n are de-safe)
   b. Identify additional de-safe attributes $\Rightarrow S_{n+1}$
      (assuming that B_{n+1} are bounded and S_n are de-safe)

Liberal Safety

Two concepts

- A term is bounded if $G_{\Pi}(\Pi')$ contains only finitely many substitutions for it
- An attribute is de-safe if $G_{\Pi}(\Pi')$ contains only finitely many values at this attribute position

Idea

1. Start with empty set of bounded terms B_0 and de-safe attributes S_0
2. For all $n \geq 0$ until B_n and S_n do not change anymore
   a. Identify additional bounded terms $\Rightarrow B_{n+1}$ (assuming that B_n are bounded and S_n are de-safe)
   b. Identify additional de-safe attributes $\Rightarrow S_{n+1}$ (assuming that B_{n+1} are bounded and S_n are de-safe)

Identification of bounded terms in Step 2a by term bounding functions (TBFs)
Concrete safety criteria can be plugged in by specific TBF $b(\Pi, r, S, B)$

$\Rightarrow$ TBFs are a flexible means that however must fulfill certain conditions

Liberal Safety

Range of an attribute ... set of terms which occur in the position of the attribute.

Definition (Term Bounding Function (TBF))

Function: $b(\Pi, r, S, B)$, where

- Π ... Program
- r ... rule in Π
- S ... set of already safe attributes
- B ... set of already bounded terms in r

Returns an enlarged set of bounded terms $b(\Pi, r, S, B) \supseteq B$, s.t. every $t \in b(\Pi, r, S, B)$ has finitely many substitutions in $G^{\infty}_{\Pi}(\emptyset)$ if

(i) the attributes S have a finite range in $G^{\infty}_{\Pi}(\emptyset)$ and

(ii) each term in $terms(r) \cap B$ has finitely many substitutions in $G^{\infty}_{\Pi}(\emptyset)$.

Liberal Safety

Range of an attribute ... set of terms which occur in the position of the attribute.

Definition (Term Bounding Function (TBF))

Function: $b(\Pi, r, S, B)$, where

- Π ... Program
- r ... rule in Π
- S ... set of already safe attributes
- B ... set of already bounded terms in r

Returns an enlarged set of bounded terms $b(\Pi, r, S, B) \supseteq B$, s.t. every $t \in b(\Pi, r, S, B)$ has finitely many substitutions in $G^{\infty}_{\Pi}(\emptyset)$ if

(i) the attributes S have a finite range in $G^{\infty}_{\Pi}(\emptyset)$ and

(ii) each term in $terms(r) \cap B$ has finitely many substitutions in $G^{\infty}_{\Pi}(\emptyset)$.

Concrete TBFs based on (i) syntactic criteria, (ii) semantic properties (malign cycles in the attribute dependency graph or meta-information like finite domain and finite fiber), or (iii) composed TBFs.

Grounding Algorithm

Definition (Liberal Domain-expansion Safety Relevance)

A set R of external atoms is relevant for liberal de-safety of a program Π, if $\Pi|_R$ is liberally de-safe and $var(r) = var(r|_R)$, for all $r \in \Pi$.

Definition (Input Auxiliary Rule)

For HEX-program Π and $\&g[\mathbf{Y}](\mathbf{X})$, construct $r_{inp}^{\&g[\mathbf{Y}](\mathbf{X})}$:

- The head is $H(r_{inp}^{\&g[\mathbf{Y}](\mathbf{X})}) = \{g_{inp}(\mathbf{Y})\}$, where g_{inp} is a fresh predicate; and
- The body $B(r_{inp}^{\&g[\mathbf{Y}](\mathbf{X})})$ contains each $b \in B^+(r) \setminus \{\&g[\mathbf{Y}](\mathbf{X})\}$ such that $\&g[\mathbf{Y}](\mathbf{X})$ joins b, and b is de-safety-relevant if it is an external atom.

Grounding Algorithm

Definition (External Atom Guessing Rule)

For HEX-program Π and $\&g[\mathbf{Y}](\mathbf{X})$, construct $r_{guess}^{\&g[\mathbf{Y}](\mathbf{X})}$:

- The head is $H(r_{guess}^{\&g[\mathbf{Y}](\mathbf{X})}) = \{e_{r,\&g[\mathbf{Y}]}(\mathbf{X}), ne_{r,\&g[\mathbf{Y}]}(\mathbf{X})\}$
- The body $B(r_{guess}^{\&g[\mathbf{Y}](\mathbf{X})})$ contains
  - (i) each $b \in B^{+}(r) \setminus \{\&g[\mathbf{Y}](\mathbf{X})\}$ such that $\&g[\mathbf{Y}](\mathbf{X})$ joins b and b is de-safety-relevant if it is an external atom; and
  - (ii) $g_{inp}(\mathbf{Y})$.

- Based on this, we devised a grounding algorithm GroundHEX for liberally domain-expansion safe HEX programs
- Uses an iterative grounding approach

Grounding Algorithm GroundHEX

Input: A liberally de-safe HEX-program Π
Output: A ground HEX-program Π_g s.t. $\Pi_g \equiv \Pi$
Choose a set R of *de-safety-relevant* external atoms in Π
$\Pi_p := \Pi \cup \{r_{inp}^{\&g[\mathbf{Y}](\mathbf{X})} \mid \&g[\mathbf{Y}](\mathbf{X}) \text{ in } r \in \Pi\} \cup \{r_{guess}^{\&g[\mathbf{Y}](\mathbf{X})} \mid \&g[\mathbf{Y}](\mathbf{X}) \notin R\}$
Replace all external atoms $\&g[\mathbf{Y}](\mathbf{X})$ in all rules r in Π_p by $e_{r,\&g\mathbf{Y}}(\mathbf{X})$
repeat
    $\Pi_{pg} :=$ GroundASP(Π_p) `/* partial grounding */`
    `/* evaluate all de-safety-relevant external atoms */`
    for $\&g[\mathbf{Y}](\mathbf{X}) \in R$ *in a rule* $r \in \Pi$ **do**
        $\mathbf{A}_{ma} := \{\mathbf{T}p(\mathbf{c}) \mid a(\mathbf{c}) \in A(\Pi_{pg}), p \in \mathbf{Y}_m\} \cup \{\mathbf{F}p(\mathbf{c}) \mid a(\mathbf{c}) \in A(\Pi_{pg}), p \in \mathbf{Y}_a\}$
        `/* do this under all relevant assignments */`
        for $\mathbf{A}_{nm} \subseteq \{\mathbf{T}p(\mathbf{c}), \mathbf{F}p(\mathbf{c}) \mid p(\mathbf{c}) \in A(\Pi_{pg}), p \in \mathbf{Y}_n\}$ *s.t.* $\nexists a : \mathbf{T}a, \mathbf{F}a \in \mathbf{A}_{nm}$ **do**
            $\mathbf{A} := (\mathbf{A}_{ma} \cup \mathbf{A}_{nm} \cup \{\mathbf{T}a \mid a \leftarrow \in \Pi_{pg}\}) \setminus \{\mathbf{F}a \mid a \leftarrow \in \Pi_{pg}\}$
            for $\mathbf{y} \in \{\mathbf{c} \mid r_{inp}^{\&g[\mathbf{Y}](\mathbf{X})}(\mathbf{c}) \in A(\Pi_{pg})\}$ **do**
                Let $O = \{\mathbf{x} \mid f_{\&g}(\mathbf{A} \cup \mathbf{A}_{nm}, \mathbf{y}, \mathbf{x}) = 1\}$
                `/* add the respective ground guessing rules */`
                $\Pi_p := \Pi_p \cup \{e_{r,\&g[\mathbf{y}]}(\mathbf{x}) \vee ne_{r,\&g[\mathbf{y}]}(\mathbf{x}) \leftarrow \mid \mathbf{x} \in O\}$

until Π_{pg} *did not change*
Remove input auxiliary rules and external atom guessing rules from Π_{pg}
Replace all $e_{\&g[\mathbf{y}]}(\mathbf{x})$ in Π by $\&g[\mathbf{y}](\mathbf{x})$
return Π_{pg}

Grounding Algorithm

Example

Program Π:

$$f : d(a).\ d(b).\ d(c). \qquad \begin{aligned} r_1 &: s(Y) \leftarrow \&diff[d,n](Y), d(Y). \\ r_2 &: n(Y) \leftarrow \&diff[d,s](Y), d(Y). \\ r_3 &: c(Z) \leftarrow \&count[s](Z). \end{aligned}$$

Grounding Algorithm

Example

Program Π:

$$f : d(a).\ d(b).\ d(c). \qquad \begin{aligned} r_1 &: s(Y) \leftarrow \&\mathit{diff}[d,n](Y), d(Y). \\ r_2 &: n(Y) \leftarrow \&\mathit{diff}[d,s](Y), d(Y). \\ r_3 &: c(Z) \leftarrow \&\mathit{count}[s](Z). \end{aligned}$$

Π_p at the beginning of the first iteration:

$$\begin{aligned} f &: d(a).\ d(b).\ d(c). & r_1 &: s(Y) \leftarrow e_1(Y), d(Y). \\ g_1 &: e_1(Y) \vee ne_1(Y) \leftarrow d(Y). & r_2 &: n(Y) \leftarrow e_2(Y), d(Y). \\ g_2 &: e_2(Y) \vee ne_2(Y) \leftarrow d(Y). & r_3 &: c(Z) \leftarrow e_3(Z). \end{aligned}$$

($e_1(Y)$, $e_2(Y)$, $e_3(Z)$ short for $e_{r_1,\&\mathit{diff}[d,n]}(Y)$, $e_{r_2,\&\mathit{diff}[d,s]}(Y)$, $e_{r_3,\&\mathit{count}[s]}(Z)$, resp.)

Evaluates $\&\mathit{count}[s](Z)$ under all $\mathbf{A} \subseteq \{s(a), s(b), s(c)\}$

Adds rules $\{e_3(Z) \vee ne_3(Z) \leftarrow \mid\ Z \in \{0,1,2,3\}\}$

Program Decomposition

Traditional HEX-algorithms

1. Program decomposition sometimes necessary
2. Intuition: Program is split whenever value invention may occur

Example

Program Π:

$$f : d(a).\ d(b).\ d(c). \qquad \begin{array}{l} r_1 : s(Y) \leftarrow \&\mathit{diff}[d,n](Y), d(Y). \\ r_2 : n(Y) \leftarrow \&\mathit{diff}[d,s](Y), d(Y). \\ r_3 : c(Z) \leftarrow \&\mathit{count}[s](Z). \end{array}$$

needs to be partitioned into evaluation units

1. $u_1 = \{f, r_1, r_2\}$
2. $u_2 = \{r_3\}$

where u_1 depends nonmonotonically on u_2

Program Decomposition

New Grounding Algorithm GreedyGEG

Now: Program decomposition not necessary
But: Sometimes useful

Program Decomposition

New Grounding Algorithm GreedyGEG

Now: Program decomposition not necessary
But: Sometimes useful

Input: A liberally de-safe HEX-program Π
Output: A generalized evaluation graph $\mathcal{E} = \langle V, E \rangle$ for Π
Let V be the set of (subset-maximal) strongly connected components of $G = \langle \Pi, \rightarrow_m \cup \rightarrow_n \rangle$
Update E
while V *was modified* **do**
  for $u_1, u_2 \in V$ *such that* $u_1 \neq u_2$ **do**
    if *there is no indirect path from* u_1 *to* u_2 *(via some* $u' \neq u_1, u_2$*) or vice versa* **then**
      if *no de-relevant* $\&g[\mathbf{y}](\mathbf{x})$ *in some* u_2 *has a nonmonotonic predicate input from* u_1 **then**
        $V := (V \setminus \{u_1, u_2\}) \cup \{u_1 \cup u_2\}$
        Update E

return $\mathcal{E} = \langle V, E \rangle$

Implementation and Evaluation

#	w. domain predicates			w/o domain predicates		
	wall clock	ground	solve	wall clock	ground	solve
15	0.59	0.28	0.08	0.49	0.23	0.06
25	5.78	4.67	0.33	2.94	1.90	0.35
35	36.99	33.99	1.00	14.02	11.30	0.95
45	161.91	155.40	2.18	53.09	47.19	2.22
55	—	—	n/a	171.46	158.58	5.74
65	—	—	n/a	—	—	n/a

Table: Reachability

Implementation and Evaluation

#	w. domain predicates			w/o domain predicates		
	wall clock	ground	solve	wall clock	ground	solve
10	0.49	0.01	0.39	0.52	0.02	0.41
20	3.90	0.05	3.62	4.67	0.10	4.23
30	16.12	0.18	15.32	19.59	0.36	18.32
40	48.47	0.48	46.71	51.55	0.90	48.74
50	115.56	1.00	112.14	119.40	1.79	114.11
60	254.66	1.84	248.88	257.78	3.35	248.51

Table: Set Partitioning

Implementation and Evaluation

#	w. domain predicates			w/o domain predicates		
	wall clock	ground	solve	wall clock	ground	solve
5	0.06	$<$0.005	0.01	0.08	0.02	0.01
10	0.14	$<$0.005	0.08	1.32	1.12	0.10
11	0.27	$<$0.005	0.19	2.85	2.43	0.27
12	0.32	$<$0.005	0.23	6.05	5.53	0.26
13	0.69	0.01	0.60	12.70	11.76	0.61
14	0.66	$<$0.005	0.57	28.17	26.70	0.73
15	1.66	0.01	1.49	59.73	57.14	1.46
16	1.69	0.01	1.53	139.47	131.87	1.92
17	3.83	0.01	3.57	—	—	n/a
18	4.34	0.01	4.08	—	—	n/a
19	10.07	0.01	9.56	—	—	n/a
20	11.36	0.01	10.87	—	—	n/a
24	95.60	0.01	93.35	—	—	n/a
25	—	0.01	—	—	—	n/a

Table: Bird-penguin

Implementation and Evaluation

#	w. domain predicates			w/o domain predicates		
	wall clock	ground	solve	wall clock	ground	solve
5	0.22	0.04	0.10	0.10	0.01	0.04
6	1.11	0.33	0.54	0.10	0.01	0.04
7	9.84	4.02	4.42	0.11	0.01	0.05
8	115.69	61.97	42.30	0.12	0.01	0.05
9	—	—	n/a	0.14	0.01	0.07
10	—	—	n/a	0.15	0.08	0.01
15	—	—	n/a	0.23	0.14	0.01
20	—	—	n/a	0.47	0.35	0.02
25	—	—	n/a	1.90	1.58	0.06
30	—	—	n/a	4.11	3.50	0.12
35	—	—	n/a	20.98	18.45	0.51
40	—	—	n/a	61.94	54.62	1.46
45	—	—	n/a	144.22	133.99	2.26
50	—	—	n/a	—	—	n/a

Table: Merge Sort

Implementation and Evaluation

#	monolithic			greedy		
	wall clock	ground	solve	wall clock	ground	solve
4	0.57	0.11	0.38	0.25	0.01	0.18
5	2.12	0.67	1.26	0.44	0.01	0.37
6	18.93	7.45	10.86	0.88	0.01	0.80
7	237.09	170.12	65.12	1.65	0.01	1.57
8	—	—	n/a	3.13	0.01	3.05
9	—	—	n/a	7.41	0.02	7.31
10	—	—	n/a	15.92	0.02	15.81
11	—	—	n/a	31.19	0.02	31.05
12	—	—	n/a	63.16	0.02	62.95
13	—	—	n/a	172.75	0.03	172.38
14	—	—	n/a	256.60	0.01	256.44
15	—	—	n/a	290.01	$<$0.005	290.00

Table: Argumentation

Conclusion

ASP Programs with External Sources

- Ordinary safety criteria not enough because of value invention
- Traditional strong safety is unnecessarily restrictive
  $\Rightarrow$ liberal domain-expansion safety

New Grounding Algorithm

- Based on ordinary ASP grounders
- Can ground any liberally de-safe program without splitting
- But: splitting sometimes useful for performance reasons

Future Work

- Refine and extend concept of liberally de-safety
- Exploit further syntactic and semantic properties to improve grounding
- Extend research to avoid the worst case

References

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. (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*, pages 273–287. Springer.

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

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