Answer Set Programming with External Sources: Algorithms and Efficient Evaluation

PhD Defense

Christoph Redl

redl@kr.tuwien.ac.at

Supervisors: O. Univ.-Prof. Dipl.-Ing. Dr. techn. Thomas Eiter
Associate Prof. Dipl.-Ing. Dr. techn. Stefan Woltran

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

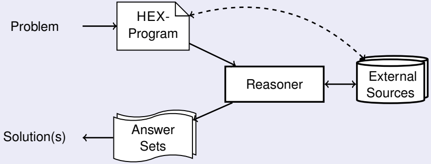
Knowledge-Based Systems Group

May 28, 2014

Motivation

HEX-Programs

- Extend ASP by external sources:

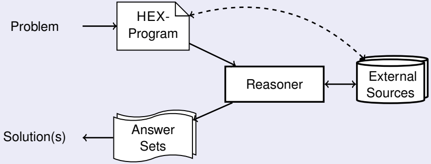


Example: $p(X, Y) \leftarrow url(U), \&rdf[U](X, Y, Z)$

Motivation

HEX-Programs

- Extend ASP by external sources:



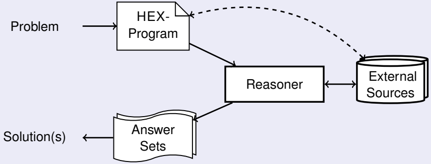
Example: $p(X,Y) \leftarrow url(U), \&rdf[U](X,Y,Z)$

- Traditional algorithm based on a translation to ASP $\Rightarrow$ natural, but ...

Motivation

HEX-Programs

- Extend ASP by external sources:



Example: $p(X,Y) \leftarrow url(U), \&rdf[U](X,Y,Z)$

- Traditional algorithm based on a translation to ASP $\Rightarrow$ natural, but ...

Two Main Problems

- Limited scalability due to blind guessing
- Restrictive syntactic conditions due to use of an ordinary ASP grounder

Motivation

Applications of HEX-Programs

- Multi-context Systems
- DL-programs – Integration of ASP with ontologies
- Constraint ASP solving
- ACTHEX – Programs with action atoms
- Physics simulation (e.g. AngryBirds agent)
- Route planning
- ...

Contributions

- New genuine evaluation algorithms based on conflict-driven algorithms
  $\Rightarrow$ significantly better scalability
- New safety concept and grounding algorithm
  $\Rightarrow$ more convenience for the user

Outline

Outline

HEX-Programs

HEX-programs extend ordinary ASP programs by external sources

Definition (HEX-programs)

A HEX-program consists of rules of form

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

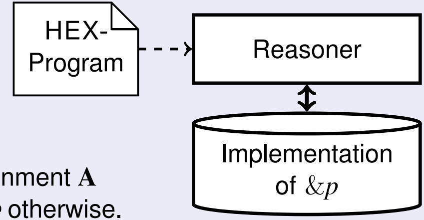
with classical literals a_i, and classical literals or an external atoms b_j.

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 $\mathbf{A}$
if $f_{\&p}(\mathbf{A}, q_1, \ldots, q_k, t_1, \ldots, t_l) = 1$ and *false* otherwise.

HEX-Programs

Definition (Answer Set)

An interpretation $\mathbf{A}$ is an answer set of program Π if it is a subset-minimal model of the FLP-reduct $f\Pi^{\mathbf{A}} = \{r \in \Pi \mid \mathbf{A} \models B(r)\}$ [Faber et al., 2011].

HEX-Programs

Definition (Answer Set)

An interpretation $\mathbf{A}$ is an answer set of program Π if it is a subset-minimal model of the FLP-reduct $f\Pi^{\mathbf{A}} = \{r \in \Pi \mid \mathbf{A} \models B(r)\}$ [Faber et al., 2011].

Example

$\&\mathit{diff}[p,q](X)$: all elements X, which are in the extension of p but not of q:

$$\begin{array}{rcl} \mathit{dom}(X) & \leftarrow & \#\mathit{int}(X) \\ n(X) & \leftarrow & \mathit{dom}(X), \&\mathit{diff}[\mathit{dom}, s](X) \\ s(X) & \leftarrow & \mathit{dom}(X), \&\mathit{diff}[\mathit{dom}, n](X) \\ & \leftarrow & s(X1), s(X2), s(X3), X1 \neq X2, X1 \neq X3, X2 \neq X3 \end{array}$$

Traditional Evaluation Method

Evaluating Program Π

1. Replace external atoms $\&g[\vec{y}](\vec{x})$ by ordinary ones $e_{\&g[\vec{y}]}(\vec{x})$ and guess their values $\Rightarrow$ guessing program $\hat{\Pi}$
2. For each candidate, check if it is a compatible set
3. If yes: check if it is also a subset-minimal model of the reduct

Definition (Compatible Set)

A compatible set of a program Π is an assignment $\hat{\mathbf{A}}$

(i) which is an answer set [Gelfond and Lifschitz, 1991] of $\hat{\Pi}$; and

(ii) $f_{\&g}(\hat{\mathbf{A}}, \vec{y}, \vec{x}) = 1$ if $\mathbf{T}e_{\&g[\vec{y}]}(\vec{x}) \in \hat{\mathbf{A}}$ and $f_{\&g}(\hat{\mathbf{A}}, \vec{y}, \vec{x}) = 0$ otherwise for all external atoms $\&g[\vec{y}](\vec{x})$ in Π.

Traditional Evaluation Method

Example

HEX-Program Π:

$$p(c_1);\ dom(c_1);\ dom(c_2);\ dom(c_3)$$
$$p(X) \leftarrow dom(X), \&empty[p](X)$$

$\&empty[p](t)$ with $t = c_1$ ($t = c_2$) is true iff$_{\text{def}}$ extension of p is empty (not empty)

Guessing program $\hat{\Pi}$:

$$p(c_1);\ dom(c_1);\ dom(c_2);\ dom(c_3)$$
$$p(X) \leftarrow dom(X), e_{\&empty[p]}(X)$$
$$e_{\&empty[p]}(X) \vee \neg e_{\&empty[p]}(X) \leftarrow dom(X)$$

8 candidates, e.g.:

$$\{\mathbf{T}p(c_1), \mathbf{T}p(c_2), \mathbf{T}dom(c_1), \mathbf{T}dom(c_2), \mathbf{T}dom(c_3),$$
$$\mathbf{F}e_{\&empty[p]}(c_1), \mathbf{T}e_{\&empty[p]}(c_2), \mathbf{F}e_{\&empty[p]}(c_3)\}$$

Compatibility check: passed $\Rightarrow$ compatible set

Outline

Algorithm for Ground HEX-Program Evaluation

Conflict-Driven Nogood Learning in ASP

- Successful SAT-solving technique adopted to ASP [Drescher et al., 2008]
- Program $\hat{\Pi}$ is represented as a set of nogoods
- Derive additional nogoods from conflicts

Algorithm for Ground HEX-Program Evaluation

Conflict-Driven Nogood Learning in ASP

- Successful SAT-solving technique adopted to ASP [Drescher et al., 2008]
- Program $\hat{\Pi}$ is represented as a set of nogoods
- Derive additional nogoods from conflicts

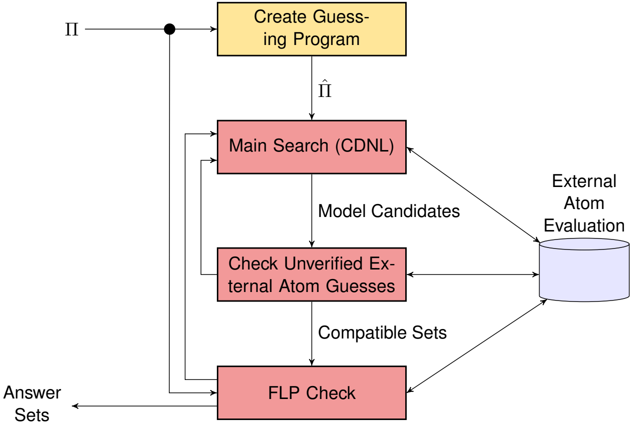
Idea

- Learn additional nogoods from external source evaluation
- Defined as learning functions of kind

$$\Lambda\colon\ \mathcal{E} \times 2^{\mathcal{D}(\Pi)} \mapsto 2^{2^{\mathcal{D}(\Pi)}}$$

  where
  $\mathcal{E}$ ...set of all external predicates with input list in Π
  $\mathcal{D}(\Pi)$ ...set of all signed literals
- Concept of correctness of learning functions wrt. a program

Overall Algorithm



Algorithm Hex-Eval

Input: A HEX-program Π
Output: All answer sets of Π

$\hat{\Pi} \leftarrow \Pi$ with all external atoms $\&[\vec{y}](\vec{x})$ replaced by $e_{\&[\vec{y}]}(\vec{x})$
Add guessing rules for all replacement atoms to $\hat{\Pi}$
$\nabla \leftarrow \emptyset$ // set of dynamic nogoods
$S \leftarrow \emptyset$ // set of all compatible sets
while $\hat{\mathbf{C}} \neq \bot$ **do**
- $\hat{\mathbf{C}} \leftarrow \bot$
- $inconsistent \leftarrow false$
- **while** $\hat{\mathbf{C}} = \bot$ *and* $inconsistent = false$ **do**
  - $\hat{\mathbf{A}} \leftarrow$ Hex-CDNL$(\Pi, \hat{\Pi}, \nabla)$
  - **if** $\hat{\mathbf{A}} = \bot$ **then**
    - $inconsistent \leftarrow true$
  - **else**
    - $compatible \leftarrow true$
    - **for** *all external atoms with input list* $\&[\vec{y}]$ *in* Π **do**
      - Evaluate $\&[\vec{y}]$ under $\hat{\mathbf{A}}$
      - $\nabla \leftarrow \nabla \cup \Lambda(\&[\vec{y}], \hat{\mathbf{A}})$
      - Let $\hat{\mathbf{A}}^{\&[\vec{y}](\vec{x})} = 1 \Leftrightarrow \mathbf{T}e_{\&[\vec{y}](\vec{x})} \in \hat{\mathbf{A}}$
      - **if** $\exists \vec{x} : f_{\&g}(\hat{\mathbf{A}}, \vec{y}, \vec{x}) \neq \hat{\mathbf{A}}^{\&[\vec{y}](\vec{x})}$ **then**
        - Add $\hat{\mathbf{A}}$ to ∇
        - Set $compatible \leftarrow false$
    - **if** $compatible$ **then** $\hat{\mathbf{C}} \leftarrow \hat{\mathbf{A}}$
- **if** $inconsistent = false$ **then**
  - // $\hat{\mathbf{C}}$ is a compatible set of Π
  - $\nabla \leftarrow \nabla \cup \{\hat{\mathbf{C}}\}$
  - **if** *FLPCheck*$(\Pi, \hat{\mathbf{C}}, \nabla)$ **then**
    - $S \leftarrow S \cup \{\hat{\mathbf{C}}\}$

return $\{\{\mathbf{T}a \in \hat{\mathbf{C}} \mid a \in A(\Pi)\} \mid \hat{\mathbf{C}} \in S\}$

Algorithm Hex-CDNL

Input: A program Π, its guessing program $\hat{\Pi}$, a set of correct nogoods ∇ of Π
Output: An answer set of $\hat{\Pi}$ (candidate for a compatible set of Π) which is a solution to all nogoods $d \in \nabla$, or $\bot$ if none exists

$\mathbf{A} \leftarrow \emptyset$ // assignment over $A(\hat{\Pi}) \cup BA(\hat{\Pi}) \cup BA(sh(\hat{\Pi}))$
$dl \leftarrow 0$ // decision level
while *true* **do**
- $(\mathbf{A}, \nabla) \leftarrow Propagation(\hat{\Pi}, \nabla, \mathbf{A})$
- **if** $\delta \subseteq \mathbf{A}$ *for some* $\delta \in \Delta_{\hat{\Pi}} \cup \Theta_{sh(\hat{\Pi})} \cup \nabla$ **then**
  - **if** $dl = 0$ **then return** $\bot$
  - $(\epsilon, k) \leftarrow Analysis(\delta, \hat{\Pi}, \nabla, \mathbf{A})$
  - $\nabla \leftarrow \nabla \cup \{\epsilon\}$
  - $\mathbf{A} \leftarrow \mathbf{A} \setminus \{\sigma \in \mathbf{A} \mid k < dl(\sigma)\}$
  - $dl \leftarrow k$
- **else if** $\mathbf{A}^{\mathbf{T}} \cup \mathbf{A}^{\mathbf{F}} = A(\hat{\Pi}) \cup BA(\hat{\Pi}) \cup BA(sh(\hat{\Pi}))$ **then**
  - $U \leftarrow UnfoundedSet(\hat{\Pi}, \mathbf{A})$
  - **if** $U \neq \emptyset$ **then**
    - let $\delta \in \lambda_{\hat{\Pi}}(U)$ such that $\delta \subseteq \mathbf{A}$
    - **if** $\{\sigma \in \delta \mid 0 < dl(\sigma)\} = \emptyset$ **then return** $\bot$
    - $(\epsilon, k) \leftarrow Analysis(\delta, \hat{\Pi}, \nabla, \mathbf{A})$
    - $\nabla \leftarrow \nabla \cup \{\epsilon\}$
    - $\mathbf{A} \leftarrow \mathbf{A} \setminus \{\sigma \in \mathbf{A} \mid k < dl(\sigma)\}$
    - $dl \leftarrow k$
  - **else**
    - **return** $\mathbf{A}^{\mathbf{T}} \cap A(\hat{\Pi})$
- **else if** *Heuristic decides to evaluate* $\&[\vec{p}]$ **then**
  - Evaluate $\&[\vec{p}]$ under $\mathbf{A}$ and set $\nabla \leftarrow \nabla \cup \Lambda(\&[\vec{p}], \mathbf{A})$
- **else if** *Heuristic decides to do a UFS check* **then**
  - Let $\Pi' \subseteq \Pi$ s.t. $\hat{\mathbf{A}}$ is complete and compatible over $\hat{\Pi}'$
  - FLPCheck$(\Pi', \mathbf{A}, \nabla)$
- **else**
  - $\sigma \leftarrow Select(\hat{\Pi}, \nabla, \mathbf{A})$
  - $dl \leftarrow dl + 1$
  - $\mathbf{A} \leftarrow \mathbf{A} \circ (\sigma)$

Algorithm for Ground HEX-Program Evaluation

Restricting to learning functions that are correct for Π, the following results hold.

Proposition

If for input Π, $\hat{\Pi}$ and ∇, Algorithm Hex-CDNL returns

(i) *an interpretation $\mathbf{A}$, then $\mathbf{A}$ is an answer set of $\hat{\Pi}$ and a solution to ∇;*

(ii) *$\bot$, then Π has no compatible set that is a solution to ∇.*

Theorem (Soundness and Completeness of Algorithm Hex-Eval)

Algorithm Hex-Eval computes all answer sets of Π.

Concrete Learning Functions: Uninformed Learning

Idea: learn that input implies output

Definition

The learning function for a general external predicate with input list $\&g[\vec{p}]$ in program Π under assignment $\mathbf{A}$ is defined as

$$\Lambda_g(\&g[\vec{p}], \mathbf{A}) = \left\{ \mathbf{A}|_{\vec{p}} \cup \{\mathbf{F}e_{\&g[\vec{y}]}(\vec{x})\} \mid \vec{x} \in ext(\&g[\vec{y}], \mathbf{A}) \right\}$$

Example

$\&diff[p, q](X)$ with $ext(p, \mathbf{A}) = \{a, b\}$, $ext(q, \mathbf{A}) = \{a, c\}$
Learn: $\left\{ \mathbf{T}p(a), \mathbf{T}p(b), \mathbf{F}p(c), \mathbf{T}q(a), \mathbf{F}q(b), \mathbf{T}q(c), \mathbf{F}e_{\&diff[p,q]}(b) \right\}$

Lemma

For all assignments $\mathbf{A}$, the nogoods $\Lambda_g(\&g[\vec{p}], \mathbf{A})$ are correct wrt. Π.

Concrete Learning Functions: Monotonicity

Idea: learn that parts of the input imply output

Definition

The learning function for an external predicate $\&g$ with input list $\vec{p}$ in program Π under assignment $\mathbf{A}$, such that $\&g$ is monotonic in $\vec{p_m} \subseteq \vec{p}$, is defined as

$$\Lambda_m(\&g[\vec{p}], \mathbf{A}) = \left\{ \{\mathbf{T}a \in \mathbf{A}|_{\vec{p_m}}\} \cup \mathbf{A}|_{\vec{p_n}} \cup \{\mathbf{F}e_{\&g[\vec{y}]}(\vec{x})\} \mid \vec{c} \in ext(\&g[\vec{y}], \mathbf{A}) \right\}$$

Example

$\&diff[p,q](X)$ with $ext(p, \mathbf{A}) = \{a, b\}$, $ext(q, \mathbf{A}) = \{a, c\}$, monotonic in p
Learn: $\left\{ \mathbf{T}p(a), \mathbf{T}p(b), \mathbf{T}q(a), \mathbf{F}q(b), \mathbf{T}q(c), \mathbf{F}e_{\&diff[p,q]}(b) \right\}$

Lemma

For all assignments $\mathbf{A}$, the nogoods $\Lambda_m(\&g[\vec{p}], \mathbf{A})$ are correct wrt. Π.

Minimality of Answer Sets

Recall

An interpretation $\mathbf{A}$ is an answer set of program Π if it is a subset-minimal model of the FLP reduct $f\Pi^{\mathbf{A}} = \{r \in \Pi \mid \mathbf{A} \models B(r)\}$.

2-Step Algorithm

1. Compute a compatible set (=answer set candidate) [Eiter et al., 2012]
2. Check minimality

Traditional Approach

1. Given a compatible set
2. Explicitly construct the reduct
3. Search for smaller models

Minimality of Answer Sets

Example

Let Π be the following program:

$$
\begin{aligned}
&dom(a); dom(b)\\
p(a) &\leftarrow dom(a), \&g[p](a)\\
p(b) &\leftarrow dom(b), \&g[p](b)
\end{aligned}
$$

Let $\&g$ implement the mapping: $\emptyset \mapsto \{b\}; \{a\} \mapsto \{a\}; \{b\} \mapsto \emptyset; \{a,b\} \mapsto \{a,b\}$

Guessing program $\hat{\Pi}$:

$$
\begin{aligned}
&dom(a); dom(b)\\
p(a) &\leftarrow dom(a), e_{\&g[p]}(a)\\
p(b) &\leftarrow dom(b), e_{\&g[p]}(b)\\
e_{\&g[p]}(a) \vee \neg e_{\&g[p]}(a) &\leftarrow\\
e_{\&g[p]}(b) \vee \neg e_{\&g[p]}(b) &\leftarrow
\end{aligned}
$$

4 candidates, e.g.: $\left\{\mathbf{T}dom(a), \mathbf{T}dom(b), \mathbf{T}p(a), \mathbf{T}e_{\&g[p]}(a)\right\}$

Minimality of Answer Sets

Example

Let Π be the following program:

$$
\begin{array}{rcl}
dom(a); dom(b) & & \\
p(a) & \leftarrow & dom(a), \&g[p](a) \\
p(b) & \leftarrow & dom(b), \&g[p](b)
\end{array}
$$

Let $\&g$ implement the mapping: $\emptyset \mapsto \{b\}; \{a\} \mapsto \{a\}; \{b\} \mapsto \emptyset; \{a,b\} \mapsto \{a,b\}$

Guessing program $\hat{\Pi}$:

$$
\begin{array}{rcl}
dom(a); dom(b) & & \\
p(a) & \leftarrow & dom(a), e_{\&g[p]}(a) \\
p(b) & \leftarrow & dom(b), e_{\&g[p]}(b) \\
e_{\&g[p]}(a) \vee \neg e_{\&g[p]}(a) & \leftarrow & \\
e_{\&g[p]}(b) \vee \neg e_{\&g[p]}(b) & \leftarrow &
\end{array}
$$

4 candidates, e.g.: $\left\{\mathbf{T}dom(a), \mathbf{T}dom(b), \mathbf{T}p(a), \mathbf{T}e_{\&g[p]}(a)\right\}$

Minimality of Answer Sets

Example

Let Π be the following program:

$$
\begin{array}{rcl}
dom(a); dom(b) & & \\
p(a) & \leftarrow & dom(a), \&g[p](a) \\
p(b) & \leftarrow & dom(b), \&g[p](b)
\end{array}
$$

Let $\&g$ implement the mapping: $\emptyset \mapsto \{b\}; \{a\} \mapsto \{a\}; \{b\} \mapsto \emptyset; \{a, b\} \mapsto \{a, b\}$

Guessing program $\hat{\Pi}$:

$$
\begin{array}{rcl}
dom(a); dom(b) & & \\
p(a) & \leftarrow & dom(a), e_{\&g[p]}(a) \\
p(b) & \leftarrow & dom(b), e_{\&g[p]}(b) \\
e_{\&g[p]}(a) \vee \neg e_{\&g[p]}(a) & \leftarrow & \\
e_{\&g[p]}(b) \vee \neg e_{\&g[p]}(b) & \leftarrow &
\end{array}
$$

4 candidates, e.g.: $\{\mathbf{T}dom(a), \mathbf{T}dom(b), \mathbf{T}p(a), \mathbf{T}e_{\&g[p]}(a)\}$
$\rightarrow$ passes compatibility check but fails minimality check

Using Unfounded Sets [Faber, 2005]

Definition (Unfounded Set)

A set of atoms U is an unfounded set of Π wrt. (partial) assignment $\mathbf{A}$, if for all $a \in U$ and all $r \in \Pi$ with $a \in H(r)$ at least one of the following holds:

1. $\mathbf{A} \not\models B(r)$
2. $\mathbf{A} \,\dot{\cup}\, \neg.U \not\models B(r)$
3. $\mathbf{A} \models h$ for some $h \in H(r) \setminus U$

(where $\mathbf{A} \,\dot{\cup}\, \neg.U = \big(\mathbf{A} \setminus \{\mathbf{T}a \mid a \in U\}\big) \cup \{\mathbf{F}a \mid a \in U\}$)

Definition (Unfounded-Free Assignments)

An assignment $\mathbf{A}$ is unfounded-free wrt. program Π, if there is no unfounded set U of Π wrt. $\mathbf{A}$ such that $\mathbf{T}a \in \mathbf{A}$ for some $a \in U$.

Theorem

A model $\mathbf{A}$ of a program Π is is an answer set iff it is unfounded-free.

Using Unfounded Sets

Encode the search for unfounded sets as SAT instance

Unfounded Set Search Problem – Basic Encoding

Nogood Set $\Gamma_{\Pi,\mathbf{A}} = \Gamma^N_{\Pi,\mathbf{A}} \cup \Gamma^O_{\Pi,\mathbf{A}}$ over atoms $A(\hat{\Pi}) \cup \{h_r, l_r \mid r \in \Pi\}$ consisting of a necessary part $\Gamma^N_{\Pi,\mathbf{A}}$ and an optimization part $\Gamma^O_{\Pi,\mathbf{A}}$

- $\Gamma^R_{r,\mathbf{A}} = \Gamma^H_{r,\mathbf{A}} \cup \Gamma^C_{r,\mathbf{A}}$
- $\Gamma^H_{r,\mathbf{A}} = \big\{\{\mathbf{T}h_r\} \cup \{\mathbf{F}h \mid h \in H(r)\}\big\} \cup \big\{\{\mathbf{F}h_r, \mathbf{T}h\} \mid h \in H(r)\big\}$
- $\Gamma^C_{r,\mathbf{A}} = \begin{cases} \big\{\{\mathbf{T}h_r\} \cup \\ \quad \{\mathbf{F}a \mid a \in B^+_o(r), \mathbf{A} \models a\} \cup \{\mathbf{t}a \mid a \in B_e(\hat{r})\} \cup \\ \quad \{\mathbf{T}h \mid h \in H(r), \mathbf{A} \models h\}\big\} & \mathit{if}\, \mathbf{A} \models B(r), \\ \emptyset & \mathit{otherwise} \end{cases}$

Intuition: solutions of $\Gamma_{\Pi,\mathbf{A}}$ correspond to potential unfounded sets of Π wrt. $\mathbf{A}$

Using Unfounded Sets

Each unfounded set corresponds to a solution of $\Gamma_{\Pi,\mathbf{A}}$

Using Unfounded Sets

Each unfounded set corresponds to a solution of $\Gamma_{\Pi,\mathbf{A}}$

Formally:

Proposition

Let U be an unfounded set of a program Π wrt. assignment $\mathbf{A}$ such that $\mathbf{A}^{\mathbf{T}} \cap U \neq \emptyset$. Then $I_\Gamma(U, \Gamma_{\Pi,\mathbf{A}}, \Pi, \mathbf{A})$ is a solution to $\Gamma_{\Pi,\mathbf{A}}$.

where $I_\Gamma(U, \Gamma_{\Pi,\mathbf{A}}, \Pi, \mathbf{A})$ is defined as follows:

Definition (Induced Assignment of an Unfounded Set wrt. $\Gamma_{\Pi,\mathbf{A}}$)

Let U be an unfounded set of a program Π wrt. assignment $\mathbf{A}$. The *assignment induced by U wrt.* $\Gamma_{\Pi,\mathbf{A}}$, denoted $I_\Gamma(U, \Gamma_{\Pi,\mathbf{A}}, \Pi, \mathbf{A})$, is

$$I_\Gamma(U, \Gamma_{\Pi,\mathbf{A}}, \Pi, \mathbf{A}) = I_\Gamma^0(U, \Pi, \mathbf{A}) \cup \{\mathbf{F}a \mid a \in A(\Gamma_{\Pi,\mathbf{A}}), \mathbf{T}a \notin I_\Gamma^0(U, \Pi, \mathbf{A})\},$$

where

$$\begin{aligned} I_\Gamma^0(U, \Pi, \mathbf{A}) = \quad & \{\mathbf{T}a \mid a \in U\} \cup \{\mathbf{T}h_r \mid r \in \Pi, H(r) \cap U \neq \emptyset\} \cup \\ & \{\mathbf{T}e_{\&g[\vec{y}]}(\vec{x}) \mid e_{\&g[\vec{y}]}(\vec{x}) \in A(\hat{\Pi}), \mathbf{A} \mathbin{\dot\cup} \neg.U \models \&g[\vec{y}](\vec{x})\}. \end{aligned}$$

Using Unfounded Sets

Not every solution of $\Gamma_{\Pi,\mathbf{A}}$ corresponds to an unfounded set, but ...

Using Unfounded Sets

Not every solution of $\Gamma_{\Pi,\mathbf{A}}$ corresponds to an unfounded set, but ...

Proposition

Let S be a solution to $\Gamma_{\Pi,\mathbf{A}}$ such that

(a) $\mathbf{T}e_{\&g[\vec{y}]}(\vec{x}) \in S$ *and* $\mathbf{A} \not\models \&g[\vec{p}](\vec{c})$ *implies* $\mathbf{A} \dot\cup \neg.U \models \&g[\vec{y}](\vec{x})$*; and*

(b) $\mathbf{F}e_{\&g[\vec{y}]}(\vec{x}) \in S$ *and* $\mathbf{A} \models \&g[\vec{p}](\vec{c})$ *implies* $\mathbf{A} \dot\cup \neg.U \not\models \&g[\vec{y}](\vec{x})$*,*

where $U = \{a \mid a \in A(\Pi), \mathbf{T}a \in S\}$*. Then U is an unfounded set of Π wrt. $\mathbf{A}$.*

Using Unfounded Sets

Not every solution of $\Gamma_{\Pi,\mathbf{A}}$ corresponds to an unfounded set, but ...

Proposition

Let S be a solution to $\Gamma_{\Pi,\mathbf{A}}$ such that

(a) $\mathbf{T}e_{\&g[\vec{y}]}(\vec{x}) \in S$ *and* $\mathbf{A} \not\models \&g[\vec{p}](\vec{c})$ *implies* $\mathbf{A} \dot\cup \neg.U \models \&g[\vec{y}](\vec{x})$*; and*

(b) $\mathbf{F}e_{\&g[\vec{y}]}(\vec{x}) \in S$ *and* $\mathbf{A} \models \&g[\vec{p}](\vec{c})$ *implies* $\mathbf{A} \dot\cup \neg.U \not\models \&g[\vec{y}](\vec{x})$*,*

where $U = \{a \mid a \in A(\Pi), \mathbf{T}a \in S\}$*. Then U is an unfounded set of Π wrt. $\mathbf{A}$.*

Therefore:

Our Approach

1. Compute a solution S of $\Gamma_{\Pi,\mathbf{A}}$
2. Check if post-conditions (a) and (b) hold for S
3. If yes: S represents an unfounded set
   If no: goto 1 and continue with next solution of $\Gamma_{\Pi,\mathbf{A}}$

Optimization and Learning

Optimization

- Generate additional nogoods $\Gamma^{O}_{\Pi,\mathbf{A}}$ to prune search space
- Advanced encoding Ω_{Π} which is reusable for all compatible sets

Learning

- Nogood exchange: search for compatible sets $\leftrightarrow$ UFS search
- Learn nogoods from detected unfounded sets

Decision Criterion

- Ordinary ASP solver has already performed a UFS-check
- For some programs this is sufficient
- Criterion: positive cycles through external atoms (e-cycles)

Outline

Liberal Safety: Motivation

Safety in HEX-Programs

- Ordinary safety not sufficient due to value invention
- Current notion of strong safety: no value invention by cyclic external atoms
- But: unnecessarily restrictive

Liberal Safety: Motivation

Safety in HEX-Programs

- Ordinary safety not sufficient due to value invention
- Current notion of strong safety: no value invention by cyclic external atoms
- But: unnecessarily restrictive

Example

$$\Pi = \{p(hexhex);\ p(Y) \leftarrow p(X), \&tail[X](Y)\}$$

Liberal Safety: Motivation

Safety in HEX-Programs

- Ordinary safety not sufficient due to value invention
- Current notion of strong safety: no value invention by cyclic external atoms
- But: unnecessarily restrictive

Example

$$\Pi = \{p(hexhex);\ p(Y) \leftarrow p(X), \&tail[X](Y)\}$$

Contribution

- New more liberal safety criteria
- Still guarantee finite groundability
- Based on a modular framework $\Rightarrow$ extensibility of the approach

Liberal Safety

Monotone Grounding Operator

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

where $\mathcal{A}(\Pi') = \left\{ \mathbf{T}a, \mathbf{F}a \mid a \in A(\Pi') \right\} \setminus \{ \mathbf{F}a \mid a \leftarrow . \in \Pi \}$
and $r\theta$ is the instance of r under variable substitution $\theta\colon \mathcal{V} \to \mathcal{C}$.

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

Liberal Safety

Monotone Grounding Operator

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

where $\mathcal{A}(\Pi') = \left\{ \mathbf{T}a, \mathbf{F}a \mid a \in A(\Pi') \right\} \setminus \{\mathbf{F}a \mid a \leftarrow . \in \Pi\}$
and $r\theta$ is the instance of r under variable substitution $\theta\colon\ \mathcal{V} \to \mathcal{C}$.

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

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 of G_Π:

Liberal Safety

Monotone Grounding Operator

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

where $\mathcal{A}(\Pi') = \left\{ \mathbf{T}a, \mathbf{F}a \mid a \in A(\Pi') \right\} \setminus \{ \mathbf{F}a \mid a \leftarrow . \in \Pi \}$
and $r\theta$ is the instance of r under variable substitution $\theta\colon\ \mathcal{V} \to \mathcal{C}$.

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

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 of G_{Π}:

$$r_1':\ \ s(a); \quad r_2':\ dom(ax); \quad r_3':\ dom(axx)$$

Liberal Safety

Monotone Grounding Operator

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

where $\mathcal{A}(\Pi') = \left\{ \mathbf{T}a, \mathbf{F}a \mid a \in A(\Pi') \right\} \setminus \{ \mathbf{F}a \mid a \leftarrow . \in \Pi \}$
and $r\theta$ is the instance of r under variable substitution $\theta\colon\ \mathcal{V} \to \mathcal{C}$.

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

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 of G_Π:

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

Liberal Safety

Monotone Grounding Operator

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

where $\mathcal{A}(\Pi') = \left\{ \mathbf{T}a, \mathbf{F}a \mid a \in A(\Pi') \right\} \setminus \{ \mathbf{F}a \mid a \leftarrow . \in \Pi \}$
and $r\theta$ is the instance of r under variable substitution $\theta\colon\ \mathcal{V} \to \mathcal{C}$.

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

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 of G_Π:

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

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)

Step 2a realized by term bounding functions (TBFs) $b(\Pi, r, S, B)$

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)

Step 2a realized by term bounding functions (TBFs) $b(\Pi, r, S, B)$

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

Liberal Safety: Modularity

Modular composition of TBFs:

Proposition

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.*

Liberal Safety: Modularity

Modular composition of TBFs:

Proposition

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.

Operator G is a witness for finite groundability:

Proposition

If Π is a de-safe program, then $G_\Pi^\infty(\emptyset)$ is finite.

Theorem (Finite Restrictability of DE-Safe Programs)

Let Π be a de-safe program. Then Π is finitely restrictable and $G_\Pi^\infty(\emptyset) \equiv^{pos} \Pi$.

The results hold for any TBF!

Liberal Safety: Relations to Other Notions

We defined a TBF $b_{synsem} = b_{syn}(\Pi, r, S, B) \cup b_{sem}(\Pi, r, S, B)$

Using TBF b_{synsem}, de-safety is strictly more general than many other approaches:

Theorem

Every strongly de-safe [Eiter et al., 2006] program is de-safe.

Theorem

Every VI-restricted program [Calimeri et al., 2007] is de-safe.

Theorem

If Π is ω-restricted [Syrjänen, 2001], then it corresponds to a rewritten program $F(\Pi)$ which is de-safe.

Liberal Safety: Relations to Other Notions

We defined a TBF $b_{synsem} = b_{syn}(\Pi, r, S, B) \cup b_{sem}(\Pi, r, S, B)$

Using TBF b_{synsem}, de-safety is strictly more general than many other approaches:

Theorem

Every strongly de-safe [Eiter et al., 2006] program is de-safe.

Theorem

Every VI-restricted program [Calimeri et al., 2007] is de-safe.

Theorem

If Π is ω-restricted [Syrjänen, 2001], then it corresponds to a rewritten program $F(\Pi)$ which is de-safe.

Further techniques can often be integrated by customized TBFs!

Grounding Algorithm

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

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

Program Decomposition

Traditional HEX-Algorithms

- Program decomposition sometimes necessary
- Intuition: program is split whenever value invention may occur

Example

Program Π:

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

needs to be partitioned into evaluation units

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

where u_2 depends on u_1

Program Decomposition

New Evaluation Heuristics GreedyGEG

Now: program decomposition not necessary
But: worst-case of grounder can sometimes be avoided by splitting

Program Decomposition

New Evaluation Heuristics GreedyGEG

Now: program decomposition not necessary
But: worst-case of grounder can sometimes be avoided by splitting

Splitting might be good for the grounder but bad for the solver
$\Rightarrow$ Aim at two contrary goals

Program Decomposition

New Evaluation Heuristics GreedyGEG

Now: program decomposition not necessary
But: worst-case of grounder can sometimes be avoided by splitting

Splitting might be good for the grounder but bad for the solver
$\Rightarrow$ Aim at two contrary goals

Algorithm GreedyGEG

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 $\&g[\vec{y}](\vec{x})$ in some u_2 has a nonmonotonic predicate input from u_1* **then**
        $V \leftarrow (V \setminus \{u_1, u_2\}) \cup \{u_1 \cup u_2\}$
        Update E

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

Outline

Implementation

DLVHEX

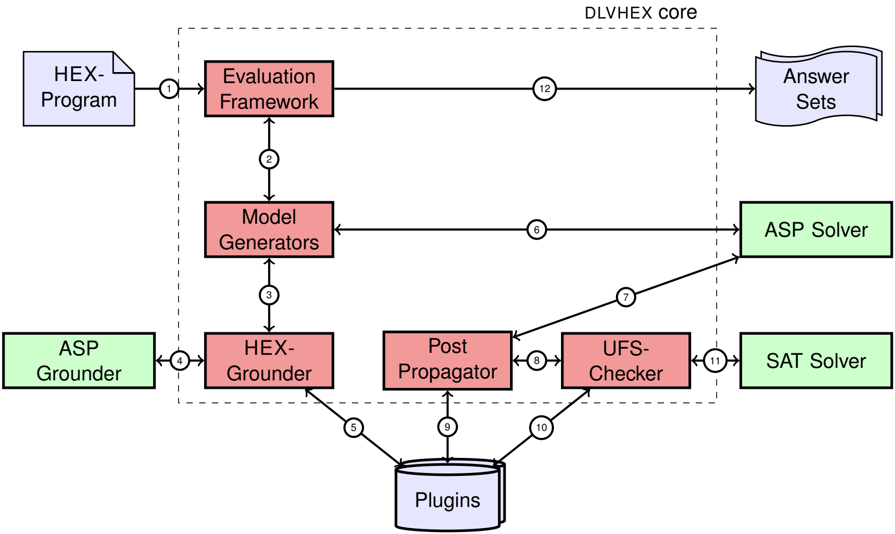
Implementation

- Platform-independent (Linux, OS X, MS Windows)
- Written in C++ (core: 130k lines of code)
- External sources loaded via plugin interface

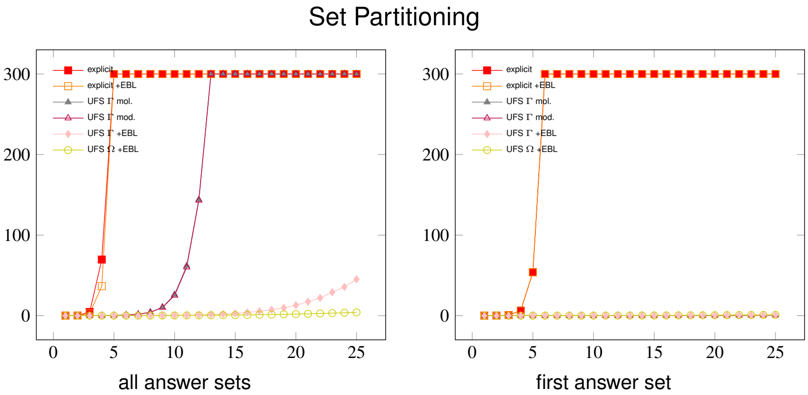
Technology

- Backend: GRINGO and CLASP from Potassco
- CLASP serves also as SAT solver for UFS search
- Alternatively: self-made grounder and solver built from scratch

System Architecture

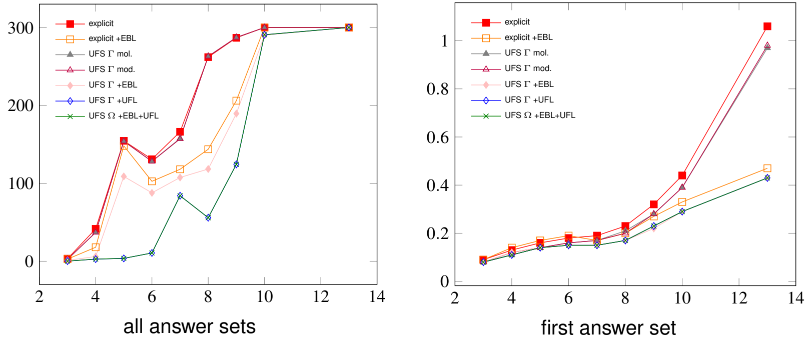


Evaluation of the Learning-Based Algorithm

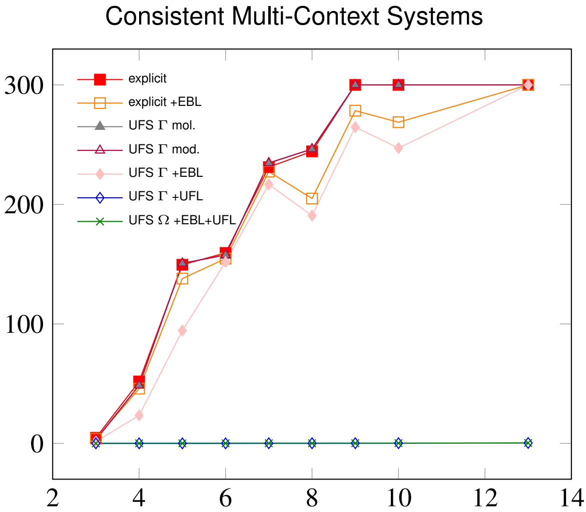


Evaluation of the Learning-Based Algorithm

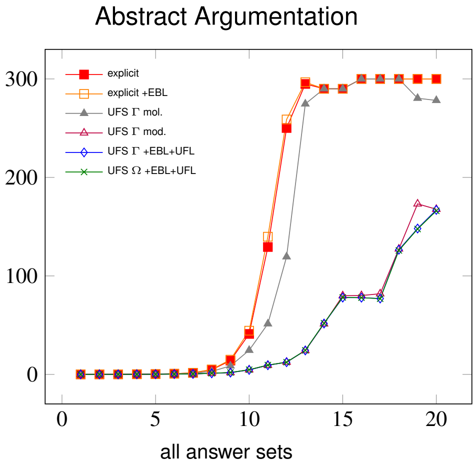
Inconsistent Multi-Context Systems



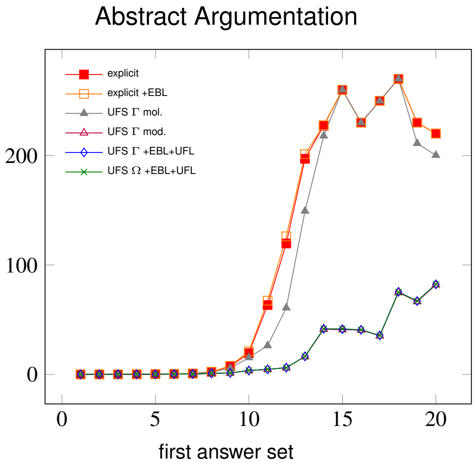
Evaluation of the Learning-Based Algorithm



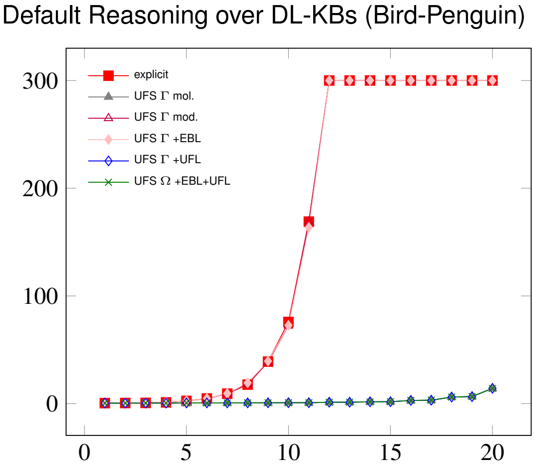
Evaluation of the Learning-Based Algorithm



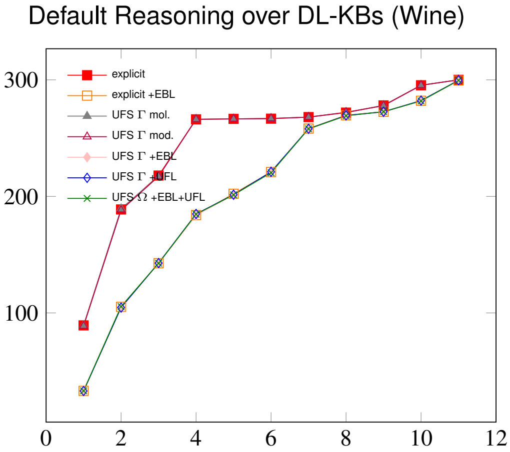
Evaluation of the Learning-Based Algorithm



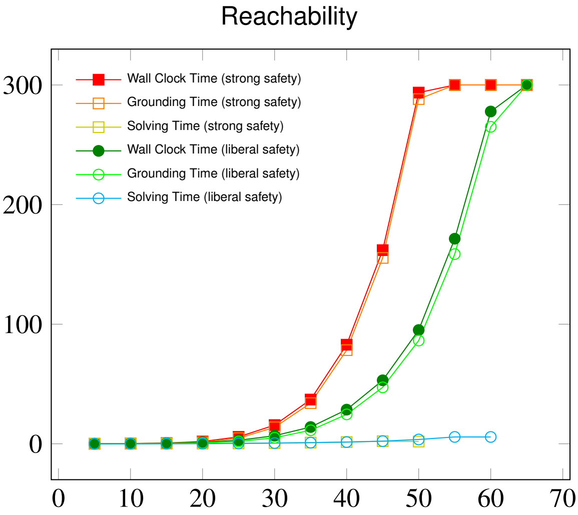
Evaluation of the Learning-Based Algorithm



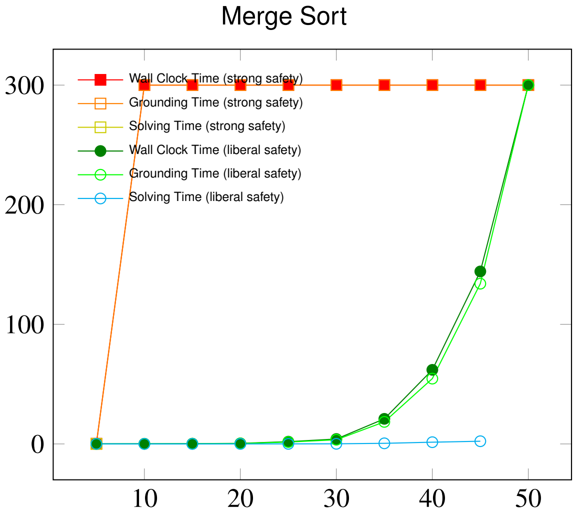
Evaluation of the Learning-Based Algorithm



Evaluation of the Grounding Algorithm

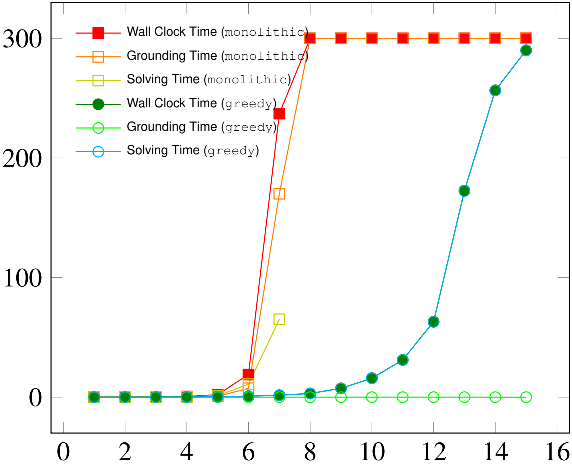


Evaluation of the Grounding Algorithm

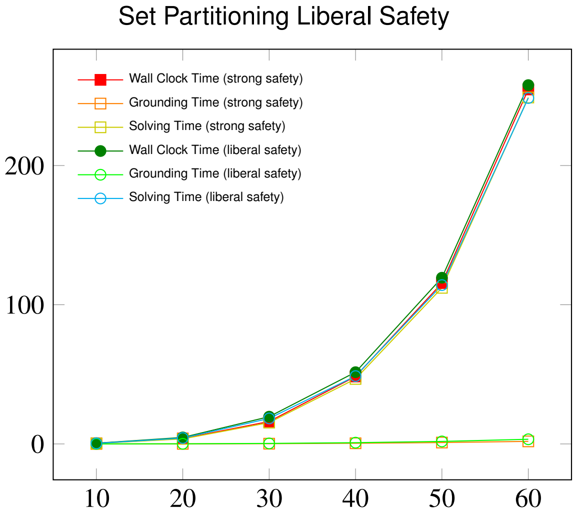


Evaluation of the Grounding Algorithm

Argumentation with Subsequent Processing

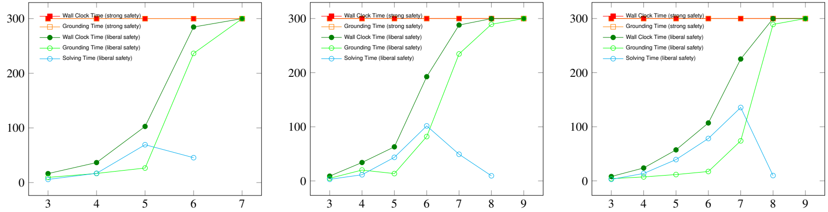


Evaluation of the Grounding Algorithm



Evaluation of the Grounding Algorithm

Route Planning



full map | subway and tram | subway only

Outline

Conclusion

HEX-Programs – ASP Programs with External Sources

- Useful for many applications:
  multi-context systems, semantic web, route planning, etc.
- But:
  traditional algorithms suffer both scalability and expressiveness limitations

Contributions

- Learning-based algorithm based on customizable learning functions
- Tight integration with UFS-based minimality-checking algorithm
- Decision criterion which sometimes allows for skipping the minimality check
- New notion of safety based on a flexible framework $\Rightarrow$ liberal de-safety
- Grounding algorithm for liberally de-safe HEX-programs

$\Rightarrow$ Significant improvement of efficiency and expressiveness

Outlook

Future Work

- Further improvement of efficiency and user's convenience
  - Application-specific learning functions
  - Pruning of the search space in minimality check
  - Improvement of the grounding algorithm
  - Refine and extend existing term bounding functions
- Integration of language extensions,
  e.g. ASP extensions and modularity techniques
- Long term goal: tight integration of grounding and solving

References I

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.

Drescher, C., Gebser, M., Grote, T., Kaufmann, B., König, A., Ostrowski, M., and Schaub, T. (2008).
Conflict-driven disjunctive answer set solving.
In *Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR 2008)*, pages 422–432. AAAI Press.

Eiter, T., Fink, M., Krennwallner, T., and Redl, C. (2012).
Conflict-driven ASP solving with external sources.
Theory and Practice of Logic Programming: Special Issue Twenty-Eighth International Conference on Logic Programming (ICLP 2012), 12(4–5):659–679.
Published online: September 5, 2012.

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 Kaelbling, L. P. and Saffiotti, A., editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 90–96, Denver, USA. Professional Book Center.

References II

Eiter, T., Ianni, G., Schindlauer, R., and Tompits, H. (2006).
Effective integration of declarative rules with external evaluations for semantic-web reasoning.
In *Proceedings of the Third European Conference on Semantic Web (ESWC 2006)*, volume 4011 of *LNCS*, pages 273–287. Springer.

Faber, W. (2005).
Unfounded sets for disjunctive logic programs with arbitrary aggregates.
In *LPNMR*, volume 3662, pages 40–52. 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., Ostrowski, M., and Schaub, T. (2009).
Constraint answer set solving.
In Hill, P. and Warren, D., editors, *Proceedings of the Twenty-Fifth International Conference on Logic Programming (ICLP 2009)*, volume 5649 of *Lecture Notes in Computer Science*, pages 235–249. Springer.

Gelfond, M. and Lifschitz, V. (1991).
Classical negation in logic programs and disjunctive databases.
New Generation Computing, 9(3–4):365–386.

References III

Syrjänen, T. (2001).
Omega-restricted logic programs.
In *Sixth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2011)*, pages 267–279. Springer.

Concrete Learning Functions: Functionality

Idea: multiple output tuples exclude each other

Definition

The learning function for a functional external predicate $\&g$ with input list $\vec{p}$ in program Π under assignment $\mathbf{A}$ is defined as

$$\Lambda_f(\&g[\vec{p}], \mathbf{A}) = \left\{ \{\mathbf{T}e_{\&g[\vec{y}]}(\vec{x}), \mathbf{T}e_{\&g[\vec{y}]}(\vec{x'})\} \mid \vec{x} \neq \vec{x'} \right\}$$

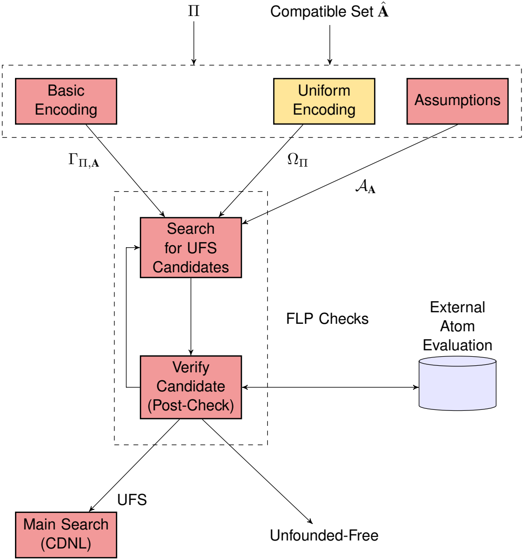
Example

$\&concat[ab, c](X)$
Learn: $\left\{ \mathbf{T}e_{\&concat[ab,c]}(abc), \mathbf{T}e_{\&concat[ab,c]}(ab) \right\}$

Lemma

For all assignments $\mathbf{A}$, if $\&g$ is functional, the nogoods $\Lambda_f(\&g[\vec{p}], \mathbf{A})$ are correct wrt. Π.

Using Unfounded Sets



Exchanging Nogoods between UFS and Main Search

Definition

A nogood of the form $N = \{\mathbf{T}t_1, \dots, \mathbf{T}t_n, \mathbf{F}f_1, \dots, \mathbf{F}f_m, \circ e_{\&g[\vec{y}]}(\vec{x})\}$, where $\circ$ is $\mathbf{T}$ or $\mathbf{F}$, is a valid input-output-relationship, if for all assignments $\mathbf{A}$, $\mathbf{T}t_i \in \mathbf{A}$, for $1 \leq i \leq n$, and $\mathbf{F}f_i \in \mathbf{A}$, for $1 \leq i \leq m$, implies $\mathbf{A} \models \&g[\vec{y}](\vec{x})$ if $\circ = \mathbf{F}$, and $\mathbf{A} \not\models \&g[\vec{y}](\vec{x})$ if $\circ = \mathbf{T}$.

Definition (Nogood Transformation $\mathcal{T}$)

Nogood transformation $\mathcal{T}$ for a valid input-output relationship N:

$$\mathcal{T}(N, \mathbf{A}) = \begin{cases} \emptyset & \text{if } \mathbf{F}t_i \in \mathbf{A} \text{ for some } 1 \leq i \leq n, \\ \{\{\mathbf{F}t_1, \dots, \mathbf{F}t_n\} \cup \{\circ e_{\&g[\vec{y}]}(\vec{x})\}\} \cup & \\ \quad \{\mathbf{T}f_i \mid 1 \leq i \leq m, \mathbf{A} \models f_i\} & \text{otherwise.} \end{cases}$$

Proposition

Let N be a valid input-output relationship, and let U be an unfounded set wrt. Π and $\mathbf{A}$. Then $I(U, \Gamma^{\mathbf{A}}_{\Pi})$ is a solution to $\mathcal{T}(N, \mathbf{A})$.

Exchanging Nogoods between UFS and Main Search

Example (Set Partitioning)

Consider the program Π:

$$\begin{aligned} s(a) &\leftarrow domain(a), \&diff[domain, n](a) \\ n(a) &\leftarrow domain(a), \&diff[domain, s](a) \\ domain(a) &\leftarrow \end{aligned}$$

Let

$$\mathbf{A}^{\mathbf{T}} = \{domain(a), s(a), e_{\&diff[n]}(a)\}$$

Suppose the main search has learned $N = \{\mathbf{T}domain(a), \mathbf{F}n(a), \mathbf{F}e_{\&diff[n]}(a)\}$.

Then $\mathcal{T}(N,\mathbf{A})=\{\{\mathbf{F}domain(a), \mathbf{F}e_{\&diff[n]}(a)\}\}$.

Learning Nogoods from Unfounded Sets

Example

Consider the program $\Pi = \{p \leftarrow \&id[p](); \ x_1 \vee x_2 \vee \ldots \vee x_k \leftarrow\}$.

Set $\{p\}$ is an unfounded set wrt. $\mathbf{A} = \{\mathbf{T}p, \mathbf{T}e_{\&id}()\}$, regarding just the first rule.

The same is true for any $\mathbf{A}' \supset \mathbf{A}$ regarding Π.

Proposition

If U is an unfounded set of Π wrt. $\mathbf{A}$, then every answer set of Π is a solution to the nogoods in $L(U, \Pi, \mathbf{A})$, where

$$L(U, \Pi, \mathbf{A}) = \{\{\sigma_0, \sigma_1, \ldots, \sigma_j\} \mid \sigma_0 \in \{\mathbf{T}a \mid a \in U\}, \sigma_i \in H_i \textit{ for all } 1 \leq i \leq j)\}$$

Learning Nogoods from Unfounded Sets

Example

Consider the program $\Pi = \{p \leftarrow \&id[p](); \; x_1 \vee x_2 \vee \ldots \vee x_k \leftarrow\}$.

Set $\{p\}$ is an unfounded set wrt. $\mathbf{A} = \{\mathbf{T}p, \mathbf{T}e_{\&id}()\}$, regarding just the first rule.

The same is true for any $\mathbf{A}' \supset \mathbf{A}$ regarding Π.

Proposition

If U is an unfounded set of Π wrt. $\mathbf{A}$, then every answer set of Π is a solution to the nogoods in $L(U, \Pi, \mathbf{A})$, where

$$L(U, \Pi, \mathbf{A}) = \{\{\sigma_0, \sigma_1, \ldots, \sigma_j\} \mid \sigma_0 \in \{\mathbf{T}a \mid a \in U\}, \sigma_i \in H_i \textit{ for all } 1 \leq i \leq j)\}$$

Example (cont'd)

Reconsider program $\Pi = \{p \leftarrow \&id[p](); \; x_1 \vee x_2 \vee \ldots \vee x_k \leftarrow\}$

Suppose the UFS $U = \{p\}$ wrt. $\mathbf{A} = \{\mathbf{T}p, \mathbf{T}x_1\} \cup \{\mathbf{F}a_i \mid 1 < i \leq k\}$ was found

The learned UFS nogood is: $L(U, \mathbf{A}, \Pi) = \{\mathbf{T}p\}$

Algorithm FLPCheck

Input: A program Π, a compatible set $\hat{\mathbf{A}}$, a set of correct nogoods ∇ of Π
Output: *true* iff $\mathbf{A}$ is an answer set of Π and *false* otherwise, learned nogoods added to ∇

if *encoding* Γ **then**
    SAT instance is $\Gamma_{\Pi}^{\mathbf{A}}$
    Let $\mathcal{T}(N) = \mathcal{T}_{\Gamma}(N)$

if *encoding* Ω **then**
    SAT instance is Ω_{Π} with assumptions $\mathcal{A}_{\mathbf{A}}$
    Let $\mathcal{T}(N) = \mathcal{T}_{\Omega}(N)$

for $C \in Comp$ **do**
    if *there is an e-cycle of* Π_C *under* $\rightarrow_p^d$ **then**
        while *SAT instance has more solutions* **do**
            Let S be the next solution of the SAT instance
            Let U be the unfounded set candidate encoded by S
            $isUFS \leftarrow true$
            for *all external atoms* $\&[\vec{y}](\vec{x})$ *in* Π_C **do**
                Evaluate $\&[\vec{y}]$
                $\nabla \leftarrow \nabla \cup \Lambda(\&[\vec{y}], \mathbf{A})$
                Add $\mathcal{T}(N)$ to the SAT instance for all $N \in \Lambda(\&[\vec{y}], \mathbf{A})$
                if $\mathbf{T}e_{\&[\vec{y}]}(\vec{x}) \in S, \mathbf{A} \not\models \&[\vec{y}](\vec{x})$ *and* $\mathbf{A} \dot{\cup} \neg.U \not\models \&[\vec{y}](\vec{x})$ **then**
                    $isUFS \leftarrow false$
                if $\mathbf{F}e_{\&[\vec{y}]}(\vec{x}) \in S, \mathbf{A} \models \&[\vec{y}](\vec{x})$ *and* $\mathbf{A} \dot{\cup} \neg.U \models \&[\vec{y}](\vec{x})$ **then**
                    $isUFS \leftarrow false$
            if $isUFS$ **then**
                Let $N \in L_1(U, \Pi_C, \mathbf{A})$ be a nogood learned from the UFS
                $\nabla \leftarrow \nabla \cup \{N\}$ **return** *false*

return *true*

Minimality Checking Algorithm

Theorem (Soundness and Completeness of Algorithm FLPCheck)

(i) *Algorithm FLPCheck returns* true *if and only if the restriction* $\mathbf{A}$ *of* $\hat{\mathbf{A}}$ *to ordinary atoms is an answer set of* Π*.*

(ii) *All answer sets of* Π *are solutions to all nogoods added to* ∇ *by FLPCheck.*

Decision Criterion for Necessity of Minimality Checking

Idea

- Ordinary ASP solver has already performed a (restricted) minimality check
- Thus many unfounded sets are already detected
- This is sufficient for many programs

Decision Criterion for Necessity of Minimality Checking

Idea

- Ordinary ASP solver has already performed a (restricted) minimality check
- Thus many unfounded sets are already detected
- This is sufficient for many programs

Criterion

- E-cycles:
  Only positive cycles which involve predicate parameters of external atoms require additional checks
- Criterion can be applied component-wise

Atom Dependency Graph

Definition (Positive Atom Dependencies)

For a ground program Π and ground atoms $p(\vec{c})$, $q(\vec{d})$, we say:

(i) $p(\vec{c})$ depends positively on $q(\vec{d})$ ($p(\vec{c}) \rightarrow_p q(\vec{d})$) if for some rule $r \in \Pi$ we have $p(\vec{c}) \in H(r)$ and $q(\vec{d}) \in B^+(r)$; and

(ii) $p(\vec{c})$ depends externally on $q(\vec{d})$ ($p(\vec{c}) \rightarrow_p^e q(\vec{d})$) if for some rule $r \in \Pi$ we have $p(\vec{c}) \in H(r)$ and there is a $\&g[q_1, \dots, q_n](\vec{d}) \in B^+(r) \cup B^-(r)$ with $q_i = q$ for some $i \in \{1, \dots, n\}$.

Example

$$\Pi = \{r \leftarrow \&id[r](); \quad p \leftarrow \&id[r](); \quad p \leftarrow q; \quad q \leftarrow p\}$$

$$\begin{tikzcd}
r \arrow[loop above, "\rightarrow_p^e"] & & p \arrow[ll, "\rightarrow_p^e"'] \arrow[rr, leftrightarrow, "\rightarrow_p"] & & q
\end{tikzcd}$$

Cuts

Definition (Cut)

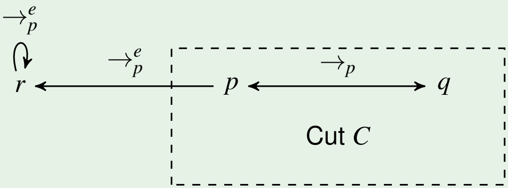
Let U be an UFS of Π wrt. $\mathbf{A}$. A set of atoms $C \subseteq U$ is a cut, if

(i) for all $a \in C, b \in U$: $b \not\rightarrow_p^e a$; and

(ii) for all $a \in C, b \in U \setminus C$: $b \not\rightarrow_p a$ and $a \not\rightarrow_p b$.

Example (ctd.)

$\Pi = \{r \leftarrow \&id[r](); \quad p \leftarrow \&id[r](); \quad p \leftarrow q; \quad q \leftarrow p\}$



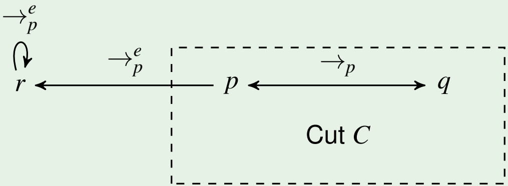
Cuts

Lemma (Unfounded Set Reduction Lemma)

Let U be an UFS of Π wrt. $\mathbf{A}$ and let C be a cut. Then $Y = U \setminus C$ is an unfounded set of Π wrt. $\mathbf{A}$.

Example (ctd.)

$$\Pi = \{r \leftarrow \&id[r](); \quad p \leftarrow \&id[r](); \quad p \leftarrow q; \quad q \leftarrow p\}$$



UFS $U = \{p, q, r\}$ wrt. $\mathbf{A} = \{\mathbf{T}p, \mathbf{T}q, \mathbf{T}r\}$
$\Rightarrow$ UFS $U' = \{p, q, r\} \setminus \{p, q\} = \{r\}$ wrt. $\mathbf{A}$

External-Atom Input Unfoundedness

Lemma (External-Atom Input Unfoundedness)

Let U be an unfounded set of Π wrt. $\mathbf{A}$. If there are no $x, y \in U$ s.t. $x \rightarrow_p^e y$, then U is an unfounded set of $\hat{\Pi}$ wrt. $\hat{\mathbf{A}}$.

Example

$\Pi = \{r \leftarrow \&id[r](); \quad p \leftarrow \&id[r](); \quad p \leftarrow q; \quad q \leftarrow p\}$

UFS $U_1 = \{p, q\}$ wrt. $\mathbf{A}' = \{\mathbf{T}p, \mathbf{T}q, \mathbf{F}r\}$ is already detected when $\hat{\Pi} = \left\{e_{\&id[r]}() \vee \neg e_{\&id[r]}() \leftarrow; \quad r \leftarrow e_{\&id[r]}(); \quad p \leftarrow e_{\&id[r]}(); \quad p \leftarrow q; \quad q \leftarrow p\right\}$ is evaluated

UFS $U_2 = \{p, q, r\}$ wrt. $\mathbf{A}'' = \{\mathbf{T}p, \mathbf{T}q, \mathbf{T}r\}$ is not detected during model generation phase of the ordinary part as $p, r \in U_2$ and $p \rightarrow_p^e r$

E-Cycles

Definition (Cycle and E-Cycle)

A cycle under a binary relation $\circ$ is a sequence of elements $C = c_0, \dots, c_{n+1}$ ($n \geq 0$) s.t. $(c_i, c_{i+1}) \in \circ$ for all $i \in \{0, \dots, n\}$ and $c_0 = c_{n+1}$.

Let $\rightarrow_p^d = \rightarrow_p \cup \leftarrow_p \cup \rightarrow_p^e$ ($\leftarrow_p$ is the inverse of $\rightarrow_p$).
A cycle $c_0, \dots, c_{n+1}$ in $\rightarrow_p^d$ is called an e-cycle, if it contains e-edges.

Proposition (Relevance of e-cycles)

Suppose U is an unfounded set of Π wrt. $\mathbf{A}$ which contains no e-cycle under $\rightarrow_p^d$. Then there exists an unfounded set of $\hat{\Pi}$ wrt. $\hat{\mathbf{A}}$.

Corollary

If there is no e-cycle under $\rightarrow_p^d$ and $\hat{\Pi}$ has no unfounded set wrt. $\hat{\mathbf{A}}$, then $\mathbf{A}$ is unfounded-free for Π.

E-Cycles

Example (Programs without E-Cycles)

$\Pi_1 = \{out(X) \leftarrow \&diff[set_1, set_2](X)\} \cup F$ ($F \ldots$ set of facts)
$\Pi_2 = \{str(Z) \leftarrow dom(Z), str(X), str(Y), \text{not } \&concat[X,Y](Z)\}$

E-Cycles

Example (Programs without E-Cycles)

$\Pi_1 = \{out(X) \leftarrow \&diff[set_1, set_2](X)\} \cup F \quad (F \ldots \text{set of facts})$
$\Pi_2 = \{str(Z) \leftarrow dom(Z), str(X), str(Y), \text{not } \&concat[X, Y](Z)\}$

Proposition (Unfoundedness of Cyclic Input Atoms)

If U is an unfounded set of Π wrt. $\mathbf{A}$ and U contains no cyclic input atoms, then $\hat{\Pi}$ has an unfounded set wrt. $\hat{\mathbf{A}}$.

Program Decomposition

Let $\mathcal{C}$ be a partitioning of the ordinary atoms $A(\Pi)$ of Π into $\subseteq$-maximal strongly connected components under $\rightarrow_p \cup \rightarrow_p^e$.

Definition (Associated Programs)

For each $C \in \mathcal{C}$, the program associated with C is defined as

$$\Pi_C = \{r \in \Pi \mid H(r) \cap C \neq \emptyset\} \quad .$$

Proposition

Let U be a nonempty unfounded set of Π wrt. $\mathbf{A}$. Then for some Π_C with $C \in \mathcal{C}$ we have that $U \cap C$ is an unfounded set of Π_C wrt. $\mathbf{A}$.

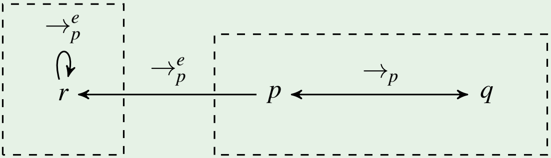
Proposition

Let U be a nonempty unfounded set of Π_C wrt. $\mathbf{A}$ such that $U \subseteq C$. Then U is an unfounded set of Π wrt. $\mathbf{A}$.

Program Decomposition

Example

$\Pi = \{r \leftarrow \&id[r](); \quad p \leftarrow \&id[r](); \quad p \leftarrow q; \quad q \leftarrow p\}$



$\mathcal{C} = \{C_1, C_2\}$ with $C_1 = \{p, q\}$ and $C_2 = \{r\}$
$\Pi_{C_1} = \{p \leftarrow \&id[r](); p \leftarrow q; q \leftarrow p\}$
$\Pi_{C_2} = \{r \leftarrow \&id[r]()\}$.

Let $U = \{p, q, r\}$ be an UFS wrt. $\mathbf{A} = \{\mathbf{T}p, \mathbf{T}q, \mathbf{T}r\}$
Then $U \cap \{r\} = \{r\}$ is also an unfounded set of Π_{C_2} wrt. $\mathbf{A}$

Liberal Safety: Concrete TBF

Definition (Syntactic Term Bounding Function)

$t \in b_{syn}(\Pi, r, S, B)$ if

(i) t is a constant in r; or

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

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

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

Liberal Safety: Concrete TBF

Example

Program Π:

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

- $B_1(r_2, \Pi, b_{syn}) = \{ax\}$, $B_1(r_3, \Pi, b_{syn}) = \{axx\}$, $B_1(r_4, \Pi, b_{syn}) = \{x\}$

Liberal Safety: Concrete TBF

Example

Program Π:

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

- $B_1(r_2, \Pi, b_{syn}) = \{ax\}$, $B_1(r_3, \Pi, b_{syn}) = \{axx\}$, $B_1(r_4, \Pi, b_{syn}) = \{x\}$
- $\Rightarrow S_1(\Pi) = \{dom\restriction 1, \&cat[X,x]_{r_4}\restriction_{\mathrm{I}} 2\}$

Liberal Safety: Concrete TBF

Example

Program Π:

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

- $B_1(r_2, \Pi, b_{syn}) = \{ax\}$, $B_1(r_3, \Pi, b_{syn}) = \{axx\}$, $B_1(r_4, \Pi, b_{syn}) = \{x\}$
- $\Rightarrow S_1(\Pi) = \{dom\restriction 1, \&cat[X,x]_{r_4}\restriction_I 2\}$
- $B_2(r_4, \Pi, b_{syn}) = \{Y\}$, $B_2(r_1, \Pi, b_{syn}) = \{a\}$

Liberal Safety: Concrete TBF

Example

Program Π:

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

- $B_1(r_2, \Pi, b_{syn}) = \{ax\}$, $B_1(r_3, \Pi, b_{syn}) = \{axx\}$, $B_1(r_4, \Pi, b_{syn}) = \{x\}$
- $\Rightarrow S_1(\Pi) = \{dom\restriction 1, \&cat[X,x]_{r_4}\restriction_{\mathrm{I}} 2\}$
- $B_2(r_4, \Pi, b_{syn}) = \{Y\}$, $B_2(r_1, \Pi, b_{syn}) = \{a\}$
- $\Rightarrow S_2(\Pi) \supseteq \{s\restriction 1, \&cat[X,x]_{r_4}\restriction_{\mathrm{O}} 1\}$

Liberal Safety: Concrete TBF

Example

Program Π:

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

- $B_1(r_2, \Pi, b_{syn}) = \{ax\}$, $B_1(r_3, \Pi, b_{syn}) = \{axx\}$, $B_1(r_4, \Pi, b_{syn}) = \{x\}$
- $\Rightarrow S_1(\Pi) = \{dom{\restriction}1, \&cat[X,x]_{r_4}{\restriction_{\mathsf{I}}}2\}$
- $B_2(r_4, \Pi, b_{syn}) = \{Y\}$, $B_2(r_1, \Pi, b_{syn}) = \{a\}$
- $\Rightarrow S_2(\Pi) \supseteq \{s{\restriction}1, \&cat[X,x]_{r_4}{\restriction_{\mathsf{O}}}1\}$
- $X \in B_3(r_4, \Pi, b_{syn})$

Liberal Safety: Concrete TBF

Example

Program Π:

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

- $B_1(r_2, \Pi, b_{syn}) = \{ax\}$, $B_1(r_3, \Pi, b_{syn}) = \{axx\}$, $B_1(r_4, \Pi, b_{syn}) = \{x\}$
- $\Rightarrow S_1(\Pi) = \{dom \upharpoonright 1, \&cat[X,x]_{r_4} \upharpoonright_{\mathrm{I}} 2\}$
- $B_2(r_4, \Pi, b_{syn}) = \{Y\}$, $B_2(r_1, \Pi, b_{syn}) = \{a\}$
- $\Rightarrow S_2(\Pi) \supseteq \{s \upharpoonright 1, \&cat[X,x]_{r_4} \upharpoonright_{\mathrm{O}} 1\}$
- $X \in B_3(r_4, \Pi, b_{syn})$
- $\Rightarrow \&cat[X,x]_{r_4} \upharpoonright_{\mathrm{I}} 1 \in S_3(\Pi)$

Liberal Safety: Concrete TBF

Example

Program Π:

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

- $B_1(r_2, \Pi, b_{syn}) = \{ax\}$, $B_1(r_3, \Pi, b_{syn}) = \{axx\}$, $B_1(r_4, \Pi, b_{syn}) = \{x\}$
- $\Rightarrow S_1(\Pi) = \{dom\restriction 1, \&cat[X,x]_{r_4}\restriction_{\mathrm{I}} 2\}$
- $B_2(r_4, \Pi, b_{syn}) = \{Y\}$, $B_2(r_1, \Pi, b_{syn}) = \{a\}$
- $\Rightarrow S_2(\Pi) \supseteq \{s\restriction 1, \&cat[X,x]_{r_4}\restriction_{\mathrm{o}} 1\}$
- $X \in B_3(r_4, \Pi, b_{syn})$
- $\Rightarrow \&cat[X,x]_{r_4}\restriction_{\mathrm{I}} 1 \in S_3(\Pi)$

We also provide a TBF which exploits semantic properties of external sources

Grounding Algorithm

Definition (Input Auxiliary Rule)

Let Π be a HEX-program, and let $a = \&g[\vec{Y}](\vec{X})$ be some external atom with input list $\vec{Y}$ occurring in a rule $r \in \Pi$. Then, for each such atom, a rule r^a_{inp} is composed:

- The head is $H\left(r^a_{inp}\right) = \{g_{inp}(\vec{Y})\}$, where g_{inp} is a fresh predicate.
- The body $B\left(r^a_{inp}\right)$ contains each $b \in B^+(r) \setminus \{a\}$ such that a joins b in $\vec{Y}$, and b is de-safety-relevant if it is an external atom.

Definition (External Atom Guessing Rule)

For HEX-program Π and $a = \&g[\vec{Y}](\vec{X})$, construct r^a_{guess}:

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

Grounding Algorithm

Example

Program Π:

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

Grounding Algorithm

Example

Program Π:

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

Π_p at the beginning of the first iteration:

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

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

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

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

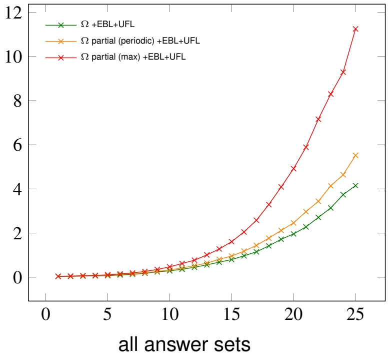
Grounding Algorithm

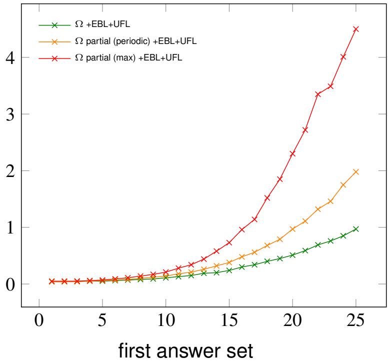
Theorem (Correctness of Algorithm GroundHEX)

If Π *is a liberally de-safe* HEX*-program, then GroundHEX*$(\Pi) \equiv^{pos} \Pi$.

Evaluation of the Learning-Based Algorithm wrt. Partial Assignments

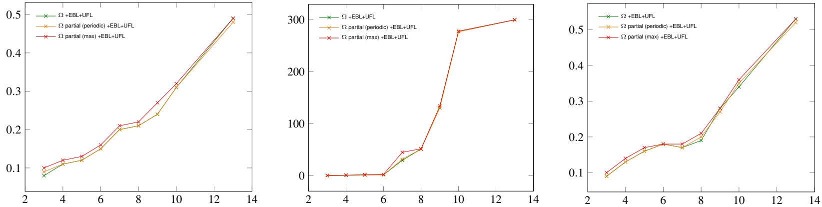
Set Partitioning





Evaluation of the Learning-Based Algorithm wrt. Partial Assignments

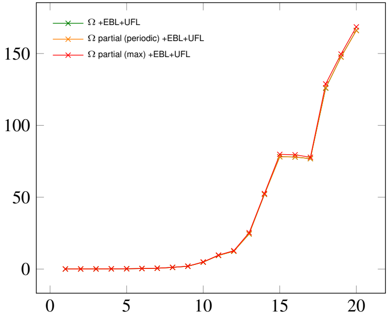
Multi-Context Systems



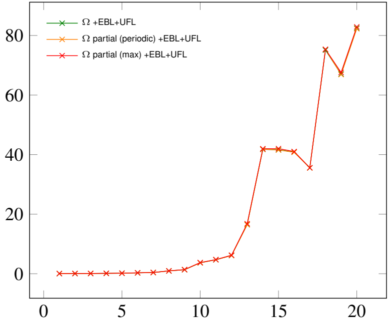
consistent (no answer sets) | inconsistent, all answer sets | inconsistent, first answer set

Evaluation of the Learning-Based Algorithm wrt. Partial Assignments

Abstract Argumentation

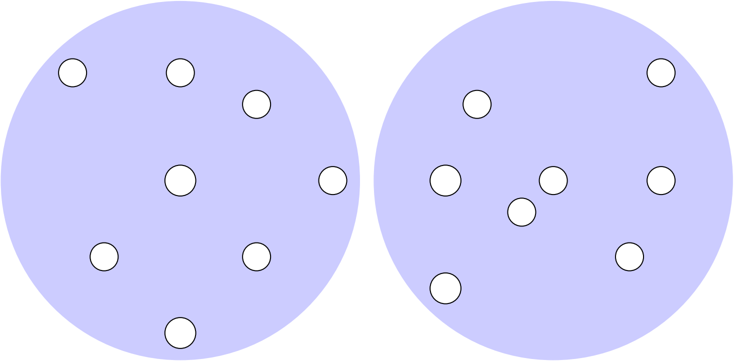


all answer sets



first answer set

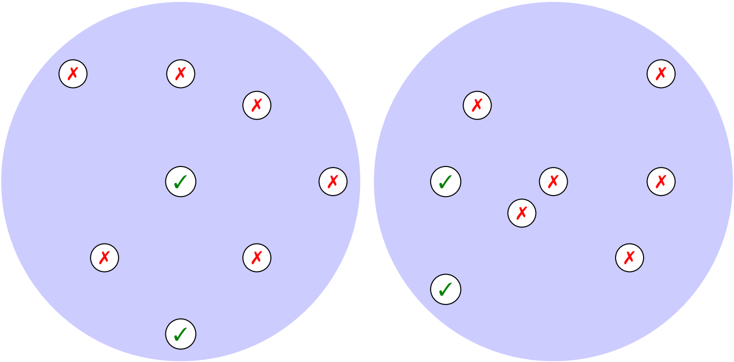
Unfounded Sets – Interesting Observations



Explicit Check
Search for Smaller Models
of the Reduct

UFS-based Check
Search for Unfounded Sets

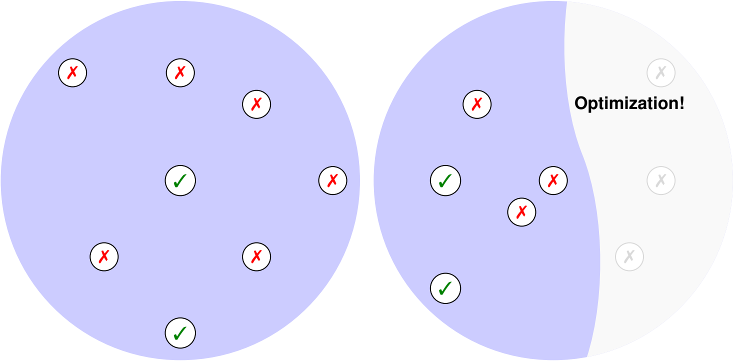
Unfounded Sets – Interesting Observations



Explicit Check
Search for Smaller Models of the Reduct

UFS-based Check
Search for Unfounded Sets

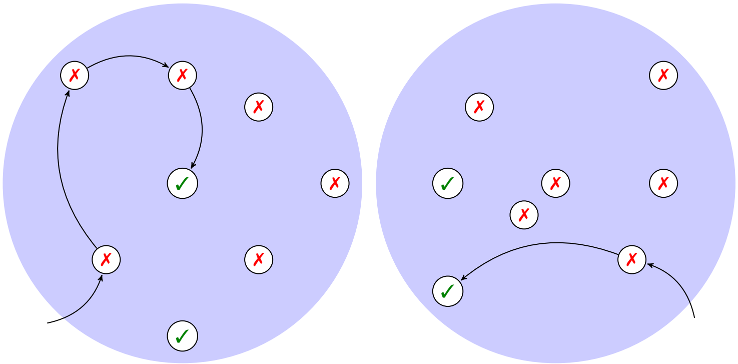
Unfounded Sets – Interesting Observations



Explicit Check
Search for Smaller Models of the Reduct

UFS-based Check
Search for Unfounded Sets

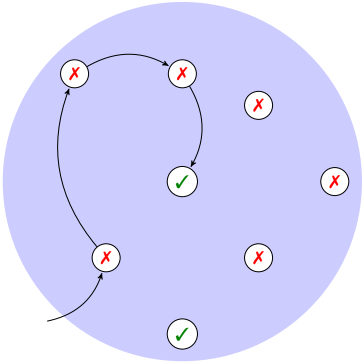
Unfounded Sets – Interesting Observations



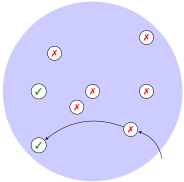
Explicit Check
Search for Smaller Models of the Reduct

UFS-based Check
Search for Unfounded Sets

Unfounded Sets – Interesting Observations

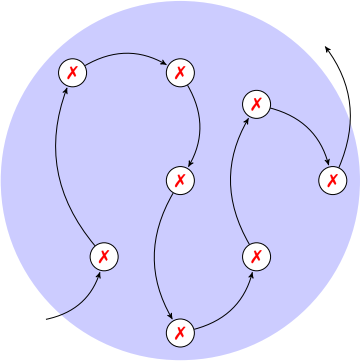


Explicit Check
Search for Smaller Models
of the Reduct

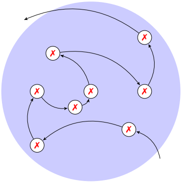


UFS-based Check
Search for Unfounded Sets

Unfounded Sets – Interesting Observations

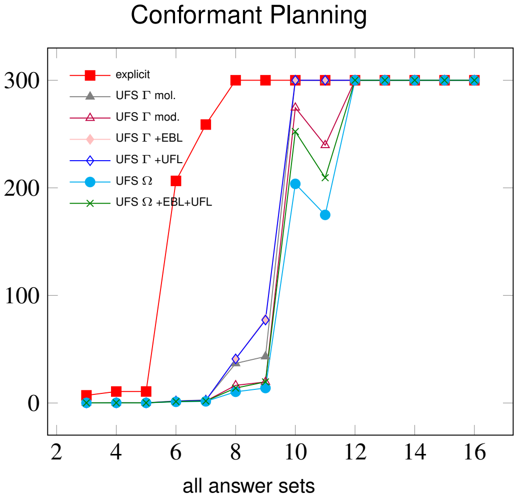


Explicit Check
Search for Smaller Models
of the Reduct

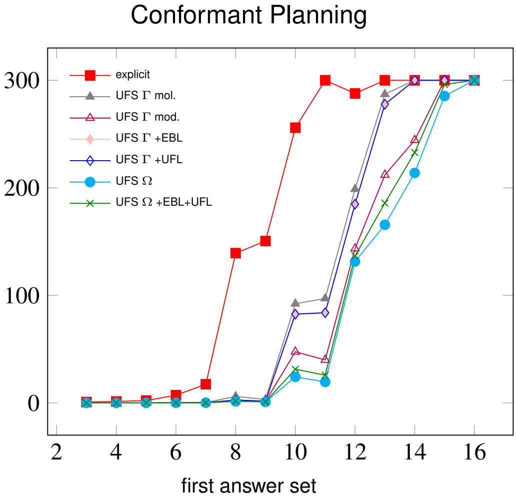


UFS-based Check
Search for Unfounded Sets

Evaluation of the Learning-Based Algorithm

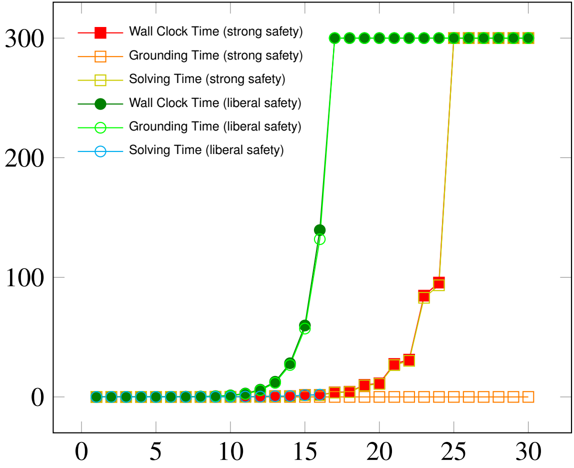


Evaluation of the Learning-Based Algorithm



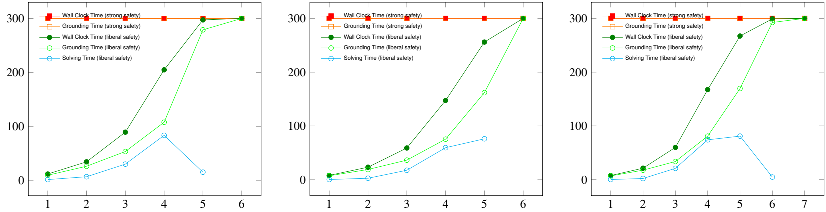
Evaluation of the Grounding Algorithm

Default Reasoning over DL-KBs with Nonmonotnic External Atom (Bird-Penguin)



Evaluation of the Grounding Algorithm

Pair Route Planning



full map — subway and tram — subway only

Evaluation of the Learning-Based Algorithm

domain	All Answer Sets						First Answer Set					
	explicit		UFS Γ	UFS Γ		Ω	explicit		UFS Γ	UFS Γ		UFS Ω
		+EBL	mol.	mod.	+EBL	+EBL		+EBL	mol.	mod.	+EBL	+EBL
1 (1)	0.05 (0)	0.05 (0)	0.04 (0)	0.04 (0)	0.04 (0)	0.04 (0)	0.04 (0)	0.04 (0)	0.04 (0)	0.04 (0)	0.04 (0)	0.04 (0)
2 (1)	0.28 (0)	0.20 (0)	0.04 (0)	0.04 (0)	0.04 (0)	0.04 (0)	0.09 (0)	0.10 (0)	0.04 (0)	0.04 (0)	0.04 (0)	0.04 (0)
3 (1)	4.65 (0)	2.82 (0)	0.06 (0)	0.05 (0)	0.05 (0)	0.05 (0)	0.70 (0)	0.70 (0)	0.04 (0)	0.04 (0)	0.04 (0)	0.04 (0)
4 (1)	69.66 (0)	36.64 (0)	0.14 (0)	0.14 (0)	0.06 (0)	0.06 (0)	6.34 (0)	6.35 (0)	0.04 (0)	0.04 (0)	0.05 (0)	0.05 (0)
5 (1)	300.00 (1)	300.00 (1)	0.33 (0)	0.32 (0)	0.09 (0)	0.07 (0)	54.02 (0)	53.80 (0)	0.05 (0)	0.05 (0)	0.05 (0)	0.05 (0)
6 (1)	300.00 (1)	300.00 (1)	0.77 (0)	0.81 (0)	0.12 (0)	0.10 (0)	300.00 (1)	300.00 (1)	0.04 (0)	0.05 (0)	0.06 (0)	0.06 (0)
7 (1)	300.00 (1)	300.00 (1)	1.73 (0)	1.78 (0)	0.20 (0)	0.13 (0)	300.00 (1)	300.00 (1)	0.06 (0)	0.06 (0)	0.06 (0)	0.07 (0)
8 (1)	300.00 (1)	300.00 (1)	4.35 (0)	4.17 (0)	0.31 (0)	0.16 (0)	300.00 (1)	300.00 (1)	0.07 (0)	0.06 (0)	0.07 (0)	0.07 (0)
9 (1)	300.00 (1)	300.00 (1)	10.42 (0)	10.21 (0)	0.47 (0)	0.23 (0)	300.00 (1)	300.00 (1)	0.08 (0)	0.07 (0)	0.08 (0)	0.09 (0)
10 (1)	300.00 (1)	300.00 (1)	26.31 (0)	25.13 (0)	0.53 (0)	0.29 (0)	300.00 (1)	300.00 (1)	0.09 (0)	0.09 (0)	0.11 (0)	0.12 (0)
15 (1)	300.00 (1)	300.00 (1)	300.00 (1)	300.00 (1)	2.83 (0)	0.79 (0)	300.00 (1)	300.00 (1)	0.19 (0)	0.15 (0)	0.27 (0)	0.26 (0)
20 (1)	300.00 (1)	300.00 (1)	300.00 (1)	300.00 (1)	12.98 (0)	1.95 (0)	300.00 (1)	300.00 (1)	0.38 (0)	0.29 (0)	0.57 (0)	0.57 (0)
25 (1)	300.00 (1)	300.00 (1)	300.00 (1)	300.00 (1)	45.18 (0)	4.11 (0)	300.00 (1)	300.00 (1)	0.70 (0)	0.47 (0)	1.09 (0)	1.08 (0)

Table: Set Partitioning

Evaluation of the Learning-Based Algorithm

#ctx	All Answer Sets explicit	+EBL	UFS Γ mol.	UFS Γ mod.	+EBL	+UFL	Ω +EBL+UFL
3 (9)	3.29 (0)	2.70 (0)	2.44 (0)	2.34 (0)	1.09 (0)	0.14 (0)	0.14 (0)
4 (14)	41.57 (1)	17.94 (0)	37.04 (1)	37.03 (1)	6.05 (0)	2.71 (0)	0.61 (0)
5 (11)	154.55 (5)	148.11 (5)	154.17 (5)	153.94 (5)	108.87 (2)	3.65 (0)	1.28 (0)
6 (18)	130.90 (7)	102.57 (6)	128.26 (7)	128.12 (7)	87.75 (4)	10.61 (0)	1.55 (0)
7 (13)	166.14 (5)	118.04 (5)	157.67 (5)	157.06 (5)	107.50 (4)	84.08 (2)	29.47 (0)
8 (6)	261.96 (5)	143.75 (2)	262.95 (5)	263.00 (5)	118.36 (2)	55.86 (1)	51.13 (1)
9 (14)	286.74 (13)	206.10 (9)	287.10 (12)	287.32 (12)	189.48 (8)	124.34 (5)	130.56 (6)
10 (12)	300.00 (12)	300.00 (12)	300.00 (12)	300.00 (12)	290.18 (11)	290.69 (11)	277.05 (11)

#ctx	First Answer Set explicit	+EBL	UFS Γ mol.	UFS Γ mod.	+EBL	+UFL	Ω +EBL+UFL
3 (9)	0.09 (0)	0.09 (0)	0.08 (0)	0.08 (0)	0.08 (0)	0.08 (0)	0.09 (0)
4 (14)	0.13 (0)	0.14 (0)	0.11 (0)	0.12 (0)	0.12 (0)	0.11 (0)	0.13 (0)
5 (11)	0.16 (0)	0.17 (0)	0.14 (0)	0.14 (0)	0.14 (0)	0.14 (0)	0.16 (0)
6 (18)	0.18 (0)	0.19 (0)	0.16 (0)	0.16 (0)	0.15 (0)	0.15 (0)	0.18 (0)
7 (13)	0.19 (0)	0.17 (0)	0.17 (0)	0.17 (0)	0.15 (0)	0.15 (0)	0.17 (0)
8 (6)	0.23 (0)	0.20 (0)	0.21 (0)	0.20 (0)	0.17 (0)	0.17 (0)	0.19 (0)
9 (14)	0.32 (0)	0.27 (0)	0.28 (0)	0.28 (0)	0.22 (0)	0.23 (0)	0.28 (0)
10 (12)	0.44 (0)	0.33 (0)	0.39 (0)	0.39 (0)	0.29 (0)	0.29 (0)	0.34 (0)

Table: Inconsistent Multi-Context Systems

Evaluation of the Learning-Based Algorithm

#ctx	explicit	+EBL	UFS Γ mol.	UFS Γ mod.	+EBL	+UFL	UFS Ω +EBL+UFL
3 (6)	4.78 (0)	3.97 (0)	2.96 (0)	2.97 (0)	1.65 (0)	0.08 (0)	0.08 (0)
4 (10)	51.90 (1)	45.91 (1)	48.71 (1)	48.59 (1)	23.48 (0)	0.10 (0)	0.11 (0)
5 (8)	149.53 (3)	137.95 (3)	150.80 (3)	150.64 (3)	94.45 (1)	0.10 (0)	0.12 (0)
6 (6)	159.41 (3)	154.69 (3)	157.62 (3)	157.72 (3)	151.89 (3)	0.12 (0)	0.15 (0)
7 (12)	231.23 (9)	227.45 (9)	234.74 (9)	234.63 (9)	216.75 (8)	0.17 (0)	0.20 (0)
8 (5)	244.39 (4)	204.92 (3)	246.42 (4)	246.34 (4)	190.60 (3)	0.17 (0)	0.21 (0)
9 (8)	300.00 (8)	278.44 (7)	300.00 (8)	300.00 (8)	264.65 (6)	0.22 (0)	0.24 (0)
10 (11)	300.00 (11)	268.78 (9)	300.00 (11)	300.00 (11)	247.16 (8)	0.25 (0)	0.31 (0)

Table: Consistent Multi-Context Systems

Evaluation of the Learning-Based Algorithm

#args	All Answer Sets					
	explicit		UFS Γ	UFS Γ		Ω
		+EBL	mol.	mod.	+EBL+UFL	+EBL+UFL
1 (30)	0.06 (0)	0.06 (0)	0.05 (0)	0.05 (0)	0.05 (0)	0.05 (0)
2 (30)	0.08 (0)	0.07 (0)	0.06 (0)	0.06 (0)	0.06 (0)	0.07 (0)
3 (30)	0.11 (0)	0.10 (0)	0.08 (0)	0.08 (0)	0.08 (0)	0.09 (0)
4 (30)	0.19 (0)	0.19 (0)	0.14 (0)	0.12 (0)	0.12 (0)	0.13 (0)
5 (30)	0.32 (0)	0.32 (0)	0.26 (0)	0.18 (0)	0.18 (0)	0.19 (0)
6 (30)	0.71 (0)	0.72 (0)	0.55 (0)	0.33 (0)	0.33 (0)	0.36 (0)
7 (30)	1.58 (0)	1.66 (0)	1.16 (0)	0.52 (0)	0.51 (0)	0.56 (0)
8 (30)	4.75 (0)	5.04 (0)	3.06 (0)	1.09 (0)	1.08 (0)	1.15 (0)
9 (30)	14.02 (0)	14.97 (0)	8.65 (0)	1.86 (0)	1.84 (0)	1.95 (0)
10 (30)	41.10 (0)	44.38 (0)	24.53 (0)	4.73 (0)	4.58 (0)	4.79 (0)
11 (30)	129.35 (1)	139.80 (2)	51.39 (0)	9.34 (0)	9.34 (0)	9.48 (0)
12 (30)	250.16 (12)	258.82 (17)	119.44 (0)	12.49 (0)	12.38 (0)	12.39 (0)
13 (30)	294.91 (27)	296.67 (27)	274.65 (19)	24.26 (0)	24.33 (0)	24.44 (0)
14 (30)	290.01 (29)	290.01 (29)	290.00 (29)	51.38 (3)	51.65 (3)	51.98 (3)
15 (30)	290.01 (29)	290.01 (29)	290.00 (29)	79.93 (3)	78.00 (3)	78.19 (3)
16 (30)	300.00 (30)	300.00 (30)	300.00 (30)	80.10 (4)	77.91 (4)	77.95 (4)
17 (30)	300.00 (30)	300.00 (30)	300.00 (30)	81.90 (5)	77.04 (5)	76.85 (5)
18 (30)	300.00 (30)	300.00 (30)	300.00 (30)	127.43 (8)	126.57 (8)	125.91 (8)
19 (30)	300.00 (30)	300.00 (30)	280.39 (28)	173.16 (13)	148.13 (10)	147.62 (10)
20 (30)	300.00 (30)	300.00 (30)	278.20 (27)	167.72 (12)	167.02 (12)	166.07 (12)

Table: Abstract Argumentation

Evaluation of the Learning-Based Algorithm

#args	First Answer Set					
	explicit	+EBL	UFS Γ mol.	UFS Γ mod.	+EBL+UFL	Ω +EBL+UFL
1 (30)	0.05 (0)	0.05 (0)	0.05 (0)	0.05 (0)	0.05 (0)	0.05 (0)
2 (30)	0.07 (0)	0.07 (0)	0.06 (0)	0.06 (0)	0.06 (0)	0.06 (0)
3 (30)	0.09 (0)	0.09 (0)	0.08 (0)	0.08 (0)	0.07 (0)	0.08 (0)
4 (30)	0.14 (0)	0.14 (0)	0.12 (0)	0.10 (0)	0.10 (0)	0.12 (0)
5 (30)	0.22 (0)	0.22 (0)	0.21 (0)	0.15 (0)	0.15 (0)	0.17 (0)
6 (30)	0.46 (0)	0.47 (0)	0.42 (0)	0.27 (0)	0.27 (0)	0.29 (0)
7 (30)	0.76 (0)	0.79 (0)	0.68 (0)	0.37 (0)	0.37 (0)	0.40 (0)
8 (30)	2.34 (0)	2.44 (0)	1.98 (0)	0.89 (0)	0.90 (0)	0.94 (0)
9 (30)	7.35 (0)	7.82 (0)	5.76 (0)	1.36 (0)	1.28 (0)	1.34 (0)
10 (30)	19.47 (0)	21.05 (0)	15.37 (0)	3.54 (0)	3.53 (0)	3.68 (0)
11 (30)	63.39 (1)	67.39 (1)	26.30 (0)	4.61 (0)	4.66 (0)	4.69 (0)
12 (30)	119.65 (4)	126.18 (4)	60.88 (0)	6.11 (0)	6.11 (0)	6.13 (0)
13 (30)	197.04 (14)	201.27 (15)	149.25 (3)	16.34 (0)	16.49 (0)	16.50 (0)
14 (30)	227.27 (22)	227.72 (22)	218.00 (17)	41.28 (2)	41.68 (2)	41.76 (2)
15 (30)	260.02 (26)	260.02 (26)	260.01 (26)	40.92 (2)	41.38 (2)	41.62 (2)
16 (30)	230.04 (23)	230.04 (23)	230.02 (23)	40.63 (3)	40.69 (3)	40.84 (3)
17 (30)	250.03 (25)	250.03 (25)	250.01 (25)	35.24 (2)	35.60 (2)	35.57 (2)
18 (30)	270.02 (27)	270.02 (27)	270.01 (27)	74.89 (5)	75.47 (5)	75.10 (5)
19 (30)	230.06 (23)	230.06 (23)	211.12 (21)	66.58 (4)	67.03 (4)	67.04 (4)
20 (30)	220.07 (22)	220.07 (22)	200.29 (20)	81.81 (5)	82.33 (5)	82.45 (5)

Table: Abstract Argumentation

Evaluation of the Learning-Based Algorithm

#cnt	explicit	+EBL	UFS Γ mol.	UFS Γ mod.	+EBL	+UFL	Ω +EBL+UFL
1 (1)	0.47 (0)	0.50 (0)	0.48 (0)	0.49 (0)	0.49 (0)	0.48 (0)	0.48 (0)
2 (1)	0.57 (0)	0.49 (0)	0.57 (0)	0.55 (0)	0.48 (0)	0.48 (0)	0.51 (0)
3 (1)	0.70 (0)	0.55 (0)	0.75 (0)	0.74 (0)	0.53 (0)	0.54 (0)	0.50 (0)
4 (1)	1.17 (0)	0.48 (0)	1.17 (0)	1.17 (0)	0.48 (0)	0.47 (0)	0.58 (0)
5 (1)	2.57 (0)	0.61 (0)	2.68 (0)	2.65 (0)	0.63 (0)	0.65 (0)	0.60 (0)
6 (1)	4.81 (0)	0.64 (0)	4.59 (0)	4.84 (0)	0.65 (0)	0.65 (0)	0.63 (0)
7 (1)	9.26 (0)	0.69 (0)	9.32 (0)	9.40 (0)	0.66 (0)	0.71 (0)	0.70 (0)
8 (1)	17.68 (0)	0.71 (0)	18.28 (0)	19.30 (0)	0.70 (0)	0.74 (0)	0.70 (0)
9 (1)	39.01 (0)	0.76 (0)	38.59 (0)	39.48 (0)	0.79 (0)	0.75 (0)	0.77 (0)
10 (1)	75.80 (0)	0.86 (0)	72.34 (0)	72.72 (0)	0.86 (0)	0.84 (0)	0.87 (0)
11 (1)	168.96 (0)	0.88 (0)	169.03 (0)	163.63 (0)	0.85 (0)	0.88 (0)	0.91 (0)
12 (1)	300.00 (1)	1.28 (0)	300.00 (1)	300.00 (1)	1.30 (0)	1.28 (0)	1.31 (0)
13 (1)	300.00 (1)	1.38 (0)	300.00 (1)	300.00 (1)	1.30 (0)	1.37 (0)	1.46 (0)
14 (1)	300.00 (1)	1.74 (0)	300.00 (1)	300.00 (1)	1.68 (0)	1.67 (0)	1.67 (0)
15 (1)	300.00 (1)	1.79 (0)	300.00 (1)	300.00 (1)	1.77 (0)	1.79 (0)	1.77 (0)
16 (1)	300.00 (1)	2.94 (0)	300.00 (1)	300.00 (1)	2.95 (0)	2.94 (0)	2.94 (0)
17 (1)	300.00 (1)	3.15 (0)	300.00 (1)	300.00 (1)	3.17 (0)	3.27 (0)	3.16 (0)
18 (1)	300.00 (1)	6.08 (0)	300.00 (1)	300.00 (1)	6.08 (0)	6.15 (0)	6.13 (0)
19 (1)	300.00 (1)	6.67 (0)	300.00 (1)	300.00 (1)	6.48 (0)	6.63 (0)	6.50 (0)
20 (1)	300.00 (1)	14.08 (0)	300.00 (1)	300.00 (1)	14.23 (0)	14.15 (0)	14.11 (0)

Table: Default Reasoning over DL-KBs (Bird-Penguin)

Evaluation of the Learning-Based Algorithm

Ontology	explicit	+EBL	UFS Γ mol.	UFS Γ mod.	+EBL	+UFL	Ω +EBL+UFL
wine_00 (34)	89.30 (9)	33.11 (3)	89.29 (9)	88.75 (9)	33.19 (3)	33.00 (3)	33.15 (3)
wine_01 (34)	188.79 (18)	105.22 (10)	189.51 (18)	188.18 (18)	104.80 (9)	104.59 (10)	105.47 (10)
wine_02 (34)	217.87 (22)	142.67 (14)	217.32 (22)	217.13 (22)	142.79 (14)	142.75 (14)	142.65 (14)
wine_03 (34)	266.10 (30)	183.98 (18)	266.15 (30)	266.14 (30)	183.69 (18)	184.91 (18)	184.73 (18)
wine_04 (34)	266.52 (30)	202.22 (19)	266.47 (30)	266.48 (30)	201.19 (18)	201.46 (19)	201.32 (19)
wine_05 (34)	266.67 (30)	220.87 (21)	266.83 (30)	266.83 (30)	221.21 (21)	221.07 (21)	220.33 (21)
wine_06 (34)	268.03 (30)	258.18 (26)	268.08 (30)	267.98 (30)	257.76 (26)	257.99 (26)	257.96 (26)
wine_07 (34)	271.86 (30)	269.57 (30)	271.97 (30)	272.14 (30)	269.43 (30)	269.45 (30)	269.20 (30)
wine_08 (34)	278.06 (30)	272.57 (30)	278.16 (30)	277.81 (30)	272.37 (30)	272.57 (30)	272.72 (30)
wine_09 (34)	295.35 (31)	282.05 (30)	295.32 (30)	295.35 (31)	282.27 (30)	282.19 (30)	281.87 (30)
wine_10 (34)	300.00 (34)	299.45 (32)	300.00 (34)	300.00 (34)	299.55 (32)	299.63 (32)	299.58 (32)

Table: Default Reasoning over DL-KBs (Wine)

Evaluation of the Grounding Algorithm

#	With Domain Predicate			Without Domain Predicates		
	wall clock	ground	solve	wall clock	ground	solve
15 (10)	0.59 (0)	0.28 (0)	0.08 (0)	0.49 (0)	0.23 (0)	0.06 (0)
25 (10)	5.78 (0)	4.67 (0)	0.33 (0)	2.94 (0)	1.90 (0)	0.35 (0)
35 (10)	36.99 (0)	33.99 (0)	1.00 (0)	14.02 (0)	11.30 (0)	0.95 (0)
45 (10)	161.91 (0)	155.40 (0)	2.18 (0)	53.09 (0)	47.19 (0)	2.22 (0)
55 (10)	300.00 (10)	300.00 (10)	n/a	171.46 (0)	158.58 (0)	5.74 (0)
65 (10)	300.00 (10)	300.00 (10)	n/a	300.00 (10)	300.00 (10)	n/a

Table: Reachability

Evaluation of the Grounding Algorithm

#	With Domain Predicate			Without Domain Predicates		
	wall clock	ground	solve	wall clock	ground	solve
5 (10)	0.22 (0)	0.04 (0)	0.10 (0)	0.10 (0)	0.01 (0)	0.04 (0)
6 (10)	1.11 (0)	0.33 (0)	0.54 (0)	0.10 (0)	0.01 (0)	0.04 (0)
7 (10)	9.84 (0)	4.02 (0)	4.42 (0)	0.11 (0)	0.01 (0)	0.05 (0)
8 (10)	115.69 (0)	61.97 (0)	42.30 (0)	0.12 (0)	0.01 (0)	0.05 (0)
9 (10)	300.00 (10)	300.00 (10)	n/a	0.14 (0)	0.01 (0)	0.07 (0)
10 (10)	300.00 (10)	300.00 (10)	n/a	0.15 (0)	0.08 (0)	0.01 (0)
15 (10)	300.00 (10)	300.00 (10)	n/a	0.23 (0)	0.14 (0)	0.01 (0)
20 (10)	300.00 (10)	300.00 (10)	n/a	0.47 (0)	0.35 (0)	0.02 (0)
25 (10)	300.00 (10)	300.00 (10)	n/a	1.90 (0)	1.58 (0)	0.06 (0)
30 (10)	300.00 (10)	300.00 (10)	n/a	4.11 (0)	3.50 (0)	0.12 (0)
35 (10)	300.00 (10)	300.00 (10)	n/a	20.98 (0)	18.45 (0)	0.51 (0)
40 (10)	300.00 (10)	300.00 (10)	n/a	61.94 (0)	54.62 (0)	1.46 (0)
45 (10)	300.00 (10)	300.00 (10)	n/a	144.22 (2)	133.99 (2)	2.26 (0)
50 (10)	300.00 (10)	300.00 (10)	n/a	300.00 (10)	300.00 (0)	n/a

Table: Merge Sort

Evaluation of the Grounding Algorithm

#	monolithic wall clock	monolithic ground	monolithic solve	greedy wall clock	greedy ground	greedy solve
4 (30)	0.57 (0)	0.11 (0)	0.38 (0)	0.25 (0)	0.01 (0)	0.18 (0)
5 (30)	2.12 (0)	0.67 (0)	1.26 (0)	0.44 (0)	0.01 (0)	0.37 (0)
6 (30)	18.93 (0)	7.45 (0)	10.86 (0)	0.88 (0)	0.01 (0)	0.80 (0)
7 (30)	237.09 (9)	170.12 (9)	65.12 (0)	1.65 (0)	0.01 (0)	1.57 (0)
8 (30)	300.00 (30)	300.00 (30)	n/a	3.13 (0)	0.01 (0)	3.05 (0)
9 (30)	300.00 (30)	300.00 (30)	n/a	7.41 (0)	0.02 (0)	7.31 (0)
10 (30)	300.00 (30)	300.00 (30)	n/a	15.92 (0)	0.02 (0)	15.81 (0)
11 (30)	300.00 (30)	300.00 (30)	n/a	31.19 (0)	0.02 (0)	31.05 (0)
12 (30)	300.00 (30)	300.00 (30)	n/a	63.16 (0)	0.02 (0)	62.95 (0)
13 (30)	300.00 (30)	300.00 (30)	n/a	172.75 (1)	0.03 (0)	172.38 (1)
14 (30)	300.00 (30)	300.00 (30)	n/a	256.60 (18)	0.01 (0)	256.44 (18)
15 (30)	300.00 (30)	300.00 (30)	n/a	290.01 (29)	$<$0.005 (0)	290.00 (29)

Table: Argumentation with Subsequent Processing

Evaluation of the Grounding Algorithm

#	With Domain Predicate			Without Domain Predicates		
	wall clock	ground	solve	wall clock	ground	solve
10 (1)	0.49 (0)	0.01 (0)	0.39 (0)	0.52 (0)	0.02 (0)	0.41 (0)
20 (1)	3.90 (0)	0.05 (0)	3.62 (0)	4.67 (0)	0.10 (0)	4.23 (0)
30 (1)	16.12 (0)	0.18 (0)	15.32 (0)	19.59 (0)	0.36 (0)	18.32 (0)
40 (1)	48.47 (0)	0.48 (0)	46.71 (0)	51.55 (0)	0.90 (0)	48.74 (0)
50 (1)	115.56 (0)	1.00 (0)	112.14 (0)	119.40 (0)	1.79 (0)	114.11 (0)
60 (1)	254.66 (0)	1.84 (0)	248.88 (0)	257.78 (0)	3.35 (0)	248.51 (0)

Table: Set Partitioning

Evaluation of the Grounding Algorithm

Full Map

#	With Domain Predicate							Without Domain Predicates						
	wall clock	ground	solve	solution (%)	length	changes	lunch (%)	wall clock	ground	solve	solution (%)	length	changes	lunch (%)
3 (50)	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a	16.44 (0)	9.46 (0)	5.81 (0)	76.00	152.21	3.92	0.00
4 (50)	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a	36.60 (0)	16.69 (0)	16.90 (0)	52.00	213.00	5.31	3.85
5 (50)	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a	102.71 (0)	26.63 (0)	69.26 (0)	52.00	281.27	7.58	11.54
6 (50)	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a	284.69 (38)	236.43 (38)	45.56 (0)	16.00	368.12	9.00	100.00
7 (50)	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a

Subway and Tram

#	With Domain Predicate							Without Domain Predicates						
	wall clock	ground	solve	solution (%)	length	changes	lunch (%)	wall clock	ground	solve	solution (%)	length	changes	lunch (%)
3 (50)	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a	8.91 (0)	4.88 (0)	3.01 (0)	96.00	125.19	3.38	0.00
4 (50)	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a	34.05 (2)	20.11 (2)	11.41 (0)	92.00	192.80	4.65	0.00
5 (50)	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a	62.98 (0)	13.57 (0)	43.58 (0)	90.00	284.89	7.53	46.67
6 (50)	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a	192.58 (11)	81.96 (11)	101.66 (0)	74.00	361.86	8.95	100.00
7 (50)	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a	287.99 (38)	234.55 (38)	49.27 (0)	24.00	410.17	10.50	100.00
8 (50)	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a	299.75 (48)	289.34 (48)	9.48 (0)	4.00	418.00	12.00	100.00
9 (50)	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a

Subway Only

#	With Domain Predicate							Without Domain Predicates						
	wall clock	ground	solve	solution (%)	length	changes	lunch (%)	wall clock	ground	solve	solution (%)	length	changes	lunch (%)
3 (50)	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a	8.08 (0)	4.10 (0)	2.91 (0)	100.00	127.16	2.62	0.00
4 (50)	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a	23.87 (0)	7.31 (0)	13.72 (0)	100.00	206.78	4.16	12.00
5 (50)	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a	57.30 (0)	11.69 (0)	39.37 (0)	100.00	317.08	7.20	90.00
6 (50)	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a	107.05 (0)	17.32 (0)	78.33 (0)	100.00	364.32	8.46	100.00
7 (50)	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a	225.16 (9)	73.97 (9)	135.58 (0)	82.00	405.27	9.61	100.00
8 (50)	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a	299.90 (48)	289.15 (48)	9.83 (0)	4.00	353.00	9.00	100.00
9 (50)	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a	300.00 (50)	300.00 (50)	0.00 (0)	0.00	n/a	n/a	n/a

Table: Route Planning