Grounding HEX-Programs with Expanding Domains

Thomas Eiter, Michael Fink, Thomas Krennwallner, Christoph Redl

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

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

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

\[ a_1 \lor \cdots \lor a_n \leftarrow b_1, \ldots, b_m, \neg b_{m+1}, \ldots, \neg b_n, \]

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 \( A \)

iff \( f_{&p}(A, q_1, \ldots, q_k, t_1, \ldots, t_l) = 1 \).
Liberal Safety: Basic Concepts

Monotone Grounding Operator

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

where \( A(\Pi') = \{ Ta, Fa \mid a \in A(\Pi') \} \setminus \{ Fa \mid a \leftarrow . \in \Pi \} \)
and \( r\theta \) is the instance of \( r \) under variable substitution \( \theta : V \rightarrow C \).

Example

Program \( \Pi \):

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

Least fixpoint \( G_\Pi^\infty (\emptyset) \) of \( G_\Pi \):

\[ r_1' : s(a). \quad r_2' : \text{dom}(ax). \quad r_3' : \text{dom}(axx). \]
Liberal Safety: Basic Concepts

Monotone Grounding Operator

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

where \( A(\Pi') = \{ Ta, Fa \mid a \in A(\Pi') \} \setminus \{ Fa \mid a \leftarrow . \in \Pi \} \)

and \( r\theta \) is the instance of \( r \) under variable substitution \( \theta : V \rightarrow C \).

Example

Program \( \Pi \):

\begin{align*}
r_1 &: s(a). \\
r_2 &: \text{dom}(ax). \\
r_3 &: \text{dom}(axx). \\
r_4 &: s(Y) \leftarrow s(X), \&\text{cat}[X, x](Y), \text{dom}(Y).
\end{align*}

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

\begin{align*}
r'_1 &: s(a). \\
r'_2 &: \text{dom}(ax). \\
r'_3 &: \text{dom}(axx). \\
r'_4 &: s(ax) \leftarrow s(a), \&\text{cat}[a, x](ax), \text{dom}(ax).
\end{align*}
**Liberal Safety: Basic Concepts**

### Monotone Grounding Operator

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

where \( A(\Pi') = \{ \text{Ta, Fa} \mid a \in A(\Pi') \} \setminus \{ \text{Fa} \mid a \leftarrow . \in \Pi \} \)

and \( r\theta \) is the instance of \( r \) under variable substitution \( \theta : V \rightarrow C \).

### Example

**Program \( \Pi \):**

\[
\begin{align*}
& r_1 : s(a) . \\
& r_2 : \text{dom}(ax) . \\
& r_3 : \text{dom}(axx) . \\
& r_4 : s(Y) \leftarrow s(X), \& \text{cat}[X, x](Y), \text{dom}(Y) .
\end{align*}
\]

**Least fixpoint** \( G_\Pi^{\infty}(\emptyset) \) of \( G_\Pi \):

\[
\begin{align*}
& r'_1 : s(a) . \\
& r'_2 : \text{dom}(ax) . \\
& r'_3 : \text{dom}(axx) . \\
& r'_4 : s(ax) \leftarrow s(a), \& \text{cat}[a, x](ax), \text{dom}(ax) . \\
& r'_5 : s(axx) \leftarrow s(ax), \& \text{cat}[ax, x](axx), \text{dom}(axx) .
\end{align*}
\]

**Intuition:** We call a program safe if this operator produces a finite grounding.
## Liberal Safety

### Two concepts

- **Term** is **bounded** if $G_{\Pi}(\Pi')$ contains only finitely many substitutions for it.
- **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:
   - Identify additional bounded terms $\Rightarrow B_{n+1}$ (assuming that $B_n$ are bounded and $S_n$ are de-safe).
   - 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 binding functions (TBFs)

Concrete safety criteria can be plugged in by specific TBF $b(\Pi, r, S, B)$

⇒ TBFs are a flexible means that however must fulfill certain conditions.
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

- \( \Pi \) ... Program
- \( r \) ... rule in \( \Pi \)
- \( S \) ... 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 \( \text{terms}(r) \cap B \) 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.

Eiter et al. (TU Vienna)
**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

- $\Pi$ . . . Program
- $r$ . . . rule in $\Pi$
- $S$ . . . 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 $G_\Pi^\infty(\emptyset)$ if

(i) the attributes $S$ have a finite range in $G_\Pi^\infty(\emptyset)$ and

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

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 $\Pi$, if $\Pi|_R$ is liberally de-safe and $\text{var}(r) = \text{var}(r|_R)$, for all $r \in \Pi$.

Definition (Input Auxiliary Rule)

For HEX-program $\Pi$ and $\&g[Y](X)$, construct $r^\&g[Y](X)$:

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

### Definition (External Atom Guessing Rule)

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

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

- 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 \( \Pi \)

**Output:** A ground HEX-program \( \Pi_g \) s.t. \( \Pi_g \equiv \Pi \)

Choose a set \( R \) of de-safety-relevant external atoms in \( \Pi \)

\[
\Pi_p := \Pi \cup \{ r_{\text{inp}}^g[Y](X) \mid &g[Y](X) \text{ in } r \in \Pi \} \cup \{ r_{\text{guess}}^g[Y](X) \mid &g[Y](X) \not\in R \}
\]

Replace all external atoms \( &g[Y](X) \) in all rules \( r \) in \( \Pi_p \) by \( e_r, &g[Y](X) \)

repeat

\[
\Pi_{pg} := \text{GroundASP}(\Pi_p) / \text{ partial grounding}
\]

/* evaluate all de-safety-relevant external atoms */

for \( &g[Y](X) \in R \) in a rule \( r \in \Pi \) do

\[
A_{ma} := \{ Tp(c) \mid a(c) \in A(\Pi_{pg}), p \in Y_m \} \cup \{ Fp(c) \mid a(c) \in A(\Pi_{pg}), p \in Y_a \}
\]

/* do this under all relevant assignments */

for \( A_{nm} \subset \{ Tp(c), Fp(c) \mid p(c) \in A(\Pi_{pg}), p \in Y_n \} \text{ s.t. } \neg \exists a : Ta, Fa \in A_{nm} \) do

\[
A := (A_{ma} \cup A_{nm} \cup \{ Ta \mid a \leftarrow \in \Pi_{pg} \}) \setminus \{ Fa \mid a \leftarrow \in \Pi_{pg} \}
\]

for \( y \in \{ c \mid r_{\text{inp}}^g[Y](X) \in A(\Pi_{pg}) \} \) do

Let \( O = \{ x \mid f_{&g}(A \cup A_{nm}, y, x) = 1 \} \)

/* add the respective ground guessing rules */

\[
\Pi_p := \Pi_p \cup \{ e_r, &g[y](x) \vee ne_r, &g[y](x) \leftarrow \mid x \in O \}
\]

until \( \Pi_{pg} \) did not change

Remove input auxiliary rules and external atom guessing rules from \( \Pi_{pg} \)

Replace all \( e_{&g[y]}(x) \) in \( \Pi \) by \( &g[y](x) \)

return \( \Pi_{pg} \)
Grounding Algorithm

Example

Program Π:

\[
\begin{align*}
\text{f} : d(a) \cdot d(b) \cdot d(c). & \quad r_1 : s(Y) \leftarrow \&\text{diff}[d, n](Y), d(Y). \\
& \quad r_2 : n(Y) \leftarrow \&\text{diff}[d, s](Y), d(Y). \\
& \quad r_3 : c(Z) \leftarrow \&\text{count}[s](Z).
\end{align*}
\]
Grounding Algorithm

Example

Program $\Pi$:

\[
\begin{align*}
  f : d(a) \cdot d(b) \cdot d(c). & \quad r_1 : s(Y) \leftarrow \&\text{diff}[d,n](Y), d(Y). \\
  r_2 : n(Y) \leftarrow \&\text{diff}[d,s](Y), d(Y). \\
  r_3 : c(Z) \leftarrow \&\text{count}[s](Z).
\end{align*}
\]

$\Pi_p$ at the beginning of the first iteration:

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

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

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

Adds rules \(\{e_3(Z) \vee ne_3(Z) \leftarrow | \hspace{1em} 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 II:

\[
\begin{align*}
  f : d(a) \cdot d(b) \cdot d(c) & \quad r_1 : s(Y) \leftarrow \text{&diff}\left[d, n\right](Y), d(Y).
  \\
  r_2 : n(Y) \leftarrow \text{&diff}\left[d, s\right](Y), d(Y).
  \\
  r_3 : c(Z) \leftarrow \text{&count}\left[s\right](Z).
\end{align*}
\]

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

Input

\[
\Pi
\]

Output

\[
E = \langle V, E \rangle
\]

Let \( V \) be the set of (subset-maximal) strongly connected components of

\[
\Gamma = \langle \Pi, \to \cup \to' \rangle
\]

Update \( E \) while \( V \) was modified

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 & \( g[y](x) \) 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

\[ E = \langle V, E \rangle \]
Program Decomposition

New Grounding Algorithm GreedyGEG

Now: Program decomposition not necessary
But: Sometimes useful

Input: A liberally de-safe HEX-program $\Pi$
Output: A generalized evaluation graph $E = \langle V, E \rangle$ for $\Pi$

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 $\&_{g[y]}(x)$ 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 $E = \langle V, E \rangle$
## Implementation and Evaluation

<table>
<thead>
<tr>
<th>#</th>
<th>w. domain predicates</th>
<th>w/o domain predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>wall clock</td>
<td>ground</td>
</tr>
<tr>
<td>15</td>
<td>0.59</td>
<td>0.28</td>
</tr>
<tr>
<td>25</td>
<td>5.78</td>
<td>4.67</td>
</tr>
<tr>
<td>35</td>
<td>36.99</td>
<td>33.99</td>
</tr>
<tr>
<td>45</td>
<td>161.91</td>
<td>155.40</td>
</tr>
<tr>
<td>55</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>65</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

**Table:** Reachability
## Implementation and Evaluation

### Table: Set Partitioning

<table>
<thead>
<tr>
<th>#</th>
<th>w. domain predicates</th>
<th>w/o domain predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>wall clock</td>
<td>ground</td>
</tr>
<tr>
<td>10</td>
<td>0.49</td>
<td>0.01</td>
</tr>
<tr>
<td>20</td>
<td>3.90</td>
<td>0.05</td>
</tr>
<tr>
<td>30</td>
<td>16.12</td>
<td>0.18</td>
</tr>
<tr>
<td>40</td>
<td>48.47</td>
<td>0.48</td>
</tr>
<tr>
<td>50</td>
<td>115.56</td>
<td>1.00</td>
</tr>
<tr>
<td>60</td>
<td>254.66</td>
<td>1.84</td>
</tr>
</tbody>
</table>
# Implementation and Evaluation

<table>
<thead>
<tr>
<th>#</th>
<th>w. domain predicates</th>
<th>w/o domain predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>wall clock</td>
<td>ground</td>
</tr>
<tr>
<td>5</td>
<td>0.06</td>
<td>&lt;0.005</td>
</tr>
<tr>
<td>10</td>
<td>0.14</td>
<td>&lt;0.005</td>
</tr>
<tr>
<td>11</td>
<td>0.27</td>
<td>&lt;0.005</td>
</tr>
<tr>
<td>12</td>
<td>0.32</td>
<td>&lt;0.005</td>
</tr>
<tr>
<td>13</td>
<td>0.69</td>
<td>0.01</td>
</tr>
<tr>
<td>14</td>
<td>0.66</td>
<td>&lt;0.005</td>
</tr>
<tr>
<td>15</td>
<td>1.66</td>
<td>0.01</td>
</tr>
<tr>
<td>16</td>
<td>1.69</td>
<td>0.01</td>
</tr>
<tr>
<td>17</td>
<td>3.83</td>
<td>0.01</td>
</tr>
<tr>
<td>18</td>
<td>4.34</td>
<td>0.01</td>
</tr>
<tr>
<td>19</td>
<td>10.07</td>
<td>0.01</td>
</tr>
<tr>
<td>20</td>
<td>11.36</td>
<td>0.01</td>
</tr>
<tr>
<td>24</td>
<td>95.60</td>
<td>0.01</td>
</tr>
<tr>
<td>25</td>
<td>—</td>
<td>0.01</td>
</tr>
</tbody>
</table>

**Table:** Bird-penguin
## Implementation and Evaluation

<table>
<thead>
<tr>
<th>#</th>
<th>w. domain predicates</th>
<th>w/o domain predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>wall clock</td>
<td>ground</td>
</tr>
<tr>
<td>5</td>
<td>0.22</td>
<td>0.04</td>
</tr>
<tr>
<td>6</td>
<td>1.11</td>
<td>0.33</td>
</tr>
<tr>
<td>7</td>
<td>9.84</td>
<td>4.02</td>
</tr>
<tr>
<td>8</td>
<td>115.69</td>
<td>61.97</td>
</tr>
<tr>
<td>9</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>10</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>15</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>20</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>25</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>30</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>35</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>40</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>45</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>50</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

**Table:** Merge Sort
## Implementation and Evaluation

<table>
<thead>
<tr>
<th>#</th>
<th>monolithic</th>
<th>greedy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>wall clock</td>
<td>ground</td>
</tr>
<tr>
<td>4</td>
<td>0.57</td>
<td>0.11</td>
</tr>
<tr>
<td>5</td>
<td>2.12</td>
<td>0.67</td>
</tr>
<tr>
<td>6</td>
<td>18.93</td>
<td>7.45</td>
</tr>
<tr>
<td>7</td>
<td>237.09</td>
<td>170.12</td>
</tr>
<tr>
<td>8</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>9</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>10</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>11</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>12</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>13</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>14</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>15</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

**Table:** Argumentation
Conclusion

ASP Programs with External Sources

- Ordinary safety criteria **not enough** because of **value invention**
- Traditional **strong safety** is **unnecessarily restrictive**
  ⇒ **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

External Sources of Knowledge and Value Invention in Logic Programming.


Omega-restricted logic programs.

Termination of term rewriting: Interpretation and type elimination.