Motivation

HEX-programs extend ordinary ASP programs by external sources.

Definition (HEX-programs)
A HEX-program consists of rules of form
\[ \text{atom}_1 \lor \cdots \lor \text{atom}_n \leftarrow \text{classical literals}_1, \ldots, \text{classical literals}_m, \neg \text{atom}_{m+1}, \ldots, \neg \text{atom}_n, \]
with classical literals \text{atom}_i and classical literals or an external atom \text{atom}_i.

Definition (External Atoms)
An external atom \text{external predicate name}_i is of the form
\[ \text{external predicate name}_i(\text{term}_1, \ldots, \text{term}_l), \]
\text{external predicate name}_i \ldots \text{predicate names or constants}
\text{term}_i \ldots \text{terms}

Problem Statement

Two steps: grounding and solving.

Due to value invention, grounding nonmonotonic external atoms is expensive.

Previous remedy: program splitting.
### Problem Statement

**Evaluation**
- Two steps: grounding and solving.
- Due to value invention, grounding nonmonotonic external atoms is expensive.
- Previous remedy: program splitting.
- But: deteriorates the performance of conflict-driven solving techniques.
- Hence: there is either a grounding or a solving bottleneck.

**Example**

\[
P = \{ r_1 : \text{in}(X) \lor \text{out}(X) \leftarrow \text{person}(X) \\ r_2 : \leftarrow \text{in}(X), \text{in}(Y), \text{conflict}(X, Y) \\ r_3 : \text{comp}(X) \leftarrow \text{competence}(X) \\ r_4 : \leftarrow \text{not-comp}(\text{technical}), \text{not-comp}(\text{financial})\}
\]

### Main Idea

Keep program splits, but push inconsistency reasons in terms of input atoms to predecessor components.
**Motivation**

Keep program splits, but push inconsistency reasons in terms of input atoms to predecessor components.

\[ P_1 = \{ r_1, r_2 \} \]
\[ P_2 = \{ r_3, r_4 \} \]

Add answer set as input atoms

Detect inconsistency

Inconsistency reason

Compute

Add as constraint

**Main Idea**

Keep program splits, but push inconsistency reasons in terms of input atoms to predecessor components.

**Contribution**

- An algorithm for computing inconsistency reasons (IRs) for a program.
- A technique for propagating IRs as constraints to predecessor components.
- An implementation and experimental evaluation.

**Inconsistency Analysis**

**Outline**

- Motivation
- Inconsistency Analysis
- Trans-Unit Propagation
- Implementation and Experiments
- Conclusion

**Formalizing Inconsistency Reasons**

**Definition (Inconsistency Reason (IR) [PL, 2017])**

Let \( P \) be a HEX-program and \( D \) be a domain of atoms.

An inconsistency reason (IR) \( R \) of \( P \) wrt \( D \) is a pair \( R = (R^+, R^-) \) of sets of atoms \( R^+ \subseteq D \) and \( R^- \subseteq D \) such that \( P \cup \text{facts}(I) \) is inconsistent for all \( I \subseteq D \) with \( R^+ \subseteq I \) and \( R^- \cap I = \emptyset \).
Inconsistency Analysis

Computing Inconsistency Reasons – Ground Case

Consider the implication graph:
Its nodes are literals in the current assignment, edges represent implications; dedicated nodes ⊥ represent conflicts.

Example
Let \( \Delta = \{ A \} \) or \( \{ \neg A \} \) be a set of literals.

Basic Approach

- To find a reason for a literal being true or a conflict ⊥, in terms of a set \( D \), find all (transitive) predecessors that are in \( D \).
- Non-ground case: additional techniques necessary since information might be lost during grounding (see paper).

Outline

- Motivation
- Inconsistency Analysis
- Trans-Unit Propagation
- Implementation and Experiments
- Conclusion

Novel Technique: Trans-Unit (TU-)Propagation

Main Idea:
Associate an IR \( R = (R^+, R^-) \) of a later program component with a constraint \( c_R \Rightarrow R^- \ (\exists x) \ a \in R^- \) which we propagate to predecessors.

Example
Let \( \Delta = \{ A \} \) or \( \{ \neg A \} \) be a set of literals.

Basic Approach

- To find a reason for a literal being true or a conflict ⊥, in terms of a set \( D \), find all (transitive) predecessors that are in \( D \).
- Non-ground case: additional techniques necessary since information might be lost during grounding (see paper).
We integrated the new evaluation technique into the DLVHEX system.

Outline

- Implementation and Experiments
- Experiments
  - Diagnosis Problem
  - Synthetic Set Guessing

Experiments

Abstract Configuration Problem

- Domain D, set of properties P
- Association \( a \) of each selection \( S \subseteq D \) with a set of properties \( w(S) \subseteq P \)
- Set of constraints \( C \) of form \( C = (C_1, C_2) \) with \( C_1 \subseteq 2^D \) and \( C_2 \subseteq 2^D \)
- Goal: \( S \subseteq D \) s.t. for all \( (C_1', C_2') \in C \) we have \( C_1' \cap w(S) \neq \emptyset \)

Table: Configuration Problem

<table>
<thead>
<tr>
<th>( d )</th>
<th>( n )</th>
<th>( m )</th>
<th>( r )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>2.17</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>2</td>
<td>3.59</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>3</td>
<td>5.17</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>4</td>
<td>7.59</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>5</td>
<td>9.83</td>
</tr>
<tr>
<td>6</td>
<td>30</td>
<td>6</td>
<td>12.08</td>
</tr>
</tbody>
</table>

Table: Synthetic Set Guessing

<table>
<thead>
<tr>
<th>( {X} )</th>
<th>( {X} )</th>
<th>( {X} )</th>
<th>( {X} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>6</td>
<td>12</td>
</tr>
</tbody>
</table>

Table: Diagnosis Problem

<table>
<thead>
<tr>
<th>( {X} )</th>
<th>( {X} )</th>
<th>( {X} )</th>
<th>( {X} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>6</td>
<td>12</td>
</tr>
</tbody>
</table>
Problem Statement

Value invention is difficult to tackle with previous evaluation techniques.

Evaluation as a monolithic program leads to a grounding bottleneck.

Evaluation based on program splitting leads to a solving bottleneck.

Idea

Overall idea:

- Keep the program splits for efficient grounding,
- but propagate nogoods over multiple program components for efficient solving.

Contributions

- Novel algorithm for computing inconsistency reasons (IRs).
- Based on this, we show how to transform IRs to nogoods that can be propagated to predecessor components.
- An experimental analysis shows promise for significant speedups.

Future Work

- Generalization of propagation from IRs to other learned nogoods.
- Grounded abstraction refinement on the program to further improve efficiency.

References