Conflict-driven ASP Solving with External Sources and Program Splits

Christoph Redl

redl@kr.tuwien.ac.at

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

August 25, 2017

Outline

HEX-Programs

HEX-programs extend ordinary ASP programs by external sources

Definition (HEX-programs)

A HEX-program consists of rules of form

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

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

Definition (External Atoms)

An external atom is of the form

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

p ...external predicate name
q_i ...predicate names or constants
t_j ...terms

Problem Statement

Evaluation

- Two steps: grounding and solving.

Problem Statement

Evaluation

- Two steps: grounding and solving.
- Due to value invention, grounding nonmonotonic external atoms is expensive.

Problem Statement

Evaluation

- 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.

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.

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

$$
\begin{aligned}
P = \{ r_1 &: in(X) \lor out(X) \leftarrow person(X).\\
r_2 &: \leftarrow in(X), in(Y), conflict(X,Y).\\
r_3 &: comp(X) \leftarrow \&competences[in](X).\\
r_4 &: \leftarrow \text{not } comp(technical), \text{not } comp(financial). \}
\end{aligned}
$$

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

$$
\begin{aligned}
P = \{ r_1 &: in(X) \lor out(X) \leftarrow person(X).\\
r_2 &: \leftarrow in(X), in(Y), conflict(X,Y).\\
r_3 &: comp(X) \leftarrow \&competences[in](X).\\
r_4 &: \leftarrow \text{not } comp(technical), \text{not } comp(financial). \}
\end{aligned}
$$

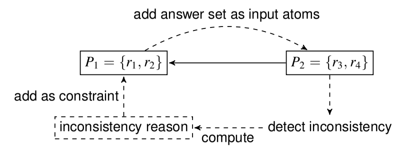
$$\boxed{P_1 = \{r_1, r_2\}} \longleftarrow \boxed{P_2 = \{r_3, r_4\}}$$

Main Idea

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

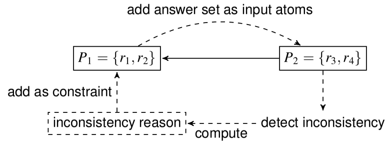
Main Idea

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



Main Idea

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

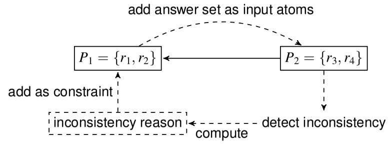


Contribution

- An algorithm for computing inconsistency reasons (IRs) for a program.

Main Idea

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

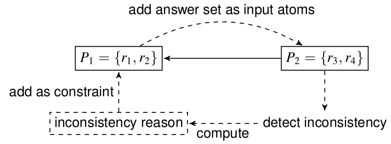


Contribution

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

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.

Outline

Formalizing Inconsistency Reasons

Definition (Inconsistency Reason (IR) [R., 2017])

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

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

Formalizing Inconsistency Reasons

Definition (Inconsistency Reason (IR) [R., 2017])

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

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

Example

Consider $P = \{\leftarrow a, \mathrm{not}\, c;\ d \leftarrow b.\}$ and $D = \{a, b, c\}$.

An IR is $R = (\{a\}, \{c\})$ because $P \cup \mathit{facts}(I)$ is inconsistent for all $I \subseteq D$ whenever $a \in I$ and $c \notin I$.

Computing Inconsistency Reasons – Ground Case

Consider the implication graph:
its nodes are literals in the current assignment, edges represent implications;
dedicated nodes $\perp$ represent conflicts.

Example

Let $\Delta = \{\delta_1\colon \{\mathbf{T}a, \mathbf{T}b\}, \delta_2\colon \{\mathbf{T}a, \mathbf{F}b, \mathbf{F}c\}, \delta_3\colon \{\mathbf{T}c, \mathbf{T}d, \mathbf{F}e\}, \delta_4\colon \{\mathbf{T}d, \mathbf{T}e\}\}$.

$$\begin{tikzcd}
\mathbf{T}a@1/\perp \arrow[rr] \arrow[dr] & & \mathbf{T}c@1/\delta_2 \arrow[r] & \mathbf{T}e@2/\delta_3 \arrow[r] & \perp@2/\delta_4 \\
& \mathbf{F}b@1/\delta_1 \arrow[ur] & & \mathbf{T}d@2/\perp \arrow[u] \arrow[ur] &
\end{tikzcd}$$

Computing Inconsistency Reasons – Ground Case

Consider the implication graph:
its nodes are literals in the current assignment, edges represent implications;
dedicated nodes $\perp$ represent conflicts.

Example

Let $\Delta = \{\delta_1\colon \{\mathbf{T}a, \mathbf{T}b\}, \delta_2\colon \{\mathbf{T}a, \mathbf{F}b, \mathbf{F}c\}, \delta_3\colon \{\mathbf{T}c, \mathbf{T}d, \mathbf{F}e\}, \delta_4\colon \{\mathbf{T}d, \mathbf{T}e\}\}$.

$$\begin{tikzcd}
\mathbf{T}a@1/\perp \arrow[rr] \arrow[dr] & & \mathbf{T}c@1/\delta_2 \arrow[r] & \mathbf{T}e@2/\delta_3 \arrow[r] & \perp@2/\delta_4 \\
& \mathbf{F}b@1/\delta_1 \arrow[ur] & & \mathbf{T}d@2/\perp \arrow[u] \arrow[ur] &
\end{tikzcd}$$

Basic Approach

- To find a reason for a literal l being true or a conflict $\perp$ in terms of a set D, find all (transitive) predecessors l that are in D.

Computing Inconsistency Reasons – Ground Case

Consider the implication graph:
its nodes are literals in the current assignment, edges represent implications;
dedicated nodes $\perp$ represent conflicts.

Example

Let $\Delta = \{\delta_1\colon \{\mathbf{T}a, \mathbf{T}b\}, \delta_2\colon \{\mathbf{T}a, \mathbf{F}b, \mathbf{F}c\}, \delta_3\colon \{\mathbf{T}c, \mathbf{T}d, \mathbf{F}e\}, \delta_4\colon \{\mathbf{T}d, \mathbf{T}e\}\}$.

$$\begin{tikzcd}
\mathbf{T}a@1/\perp \arrow[rr] \arrow[dr] & & \mathbf{T}c@1/\delta_2 \arrow[r] & \mathbf{T}e@2/\delta_3 \arrow[r] & \perp@2/\delta_4 \\
& \mathbf{F}b@1/\delta_1 \arrow[ur] & & \mathbf{T}d@2/\perp \arrow[u] \arrow[ur] &
\end{tikzcd}$$

Basic Approach

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

Outline

Novel Technique: Trans-Unit (TU-)Propagation

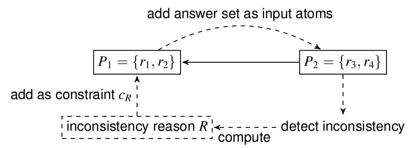
Main Idea:

Associate an IR $R = (R^+, R^-)$ of a later program component with a constraint $c_R = \leftarrow R^+, \{\mathrm{not}\, a \mid a \in R^-\}$ which we propagate to predecessors.

Novel Technique: Trans-Unit (TU-)Propagation

Main Idea:

Associate an IR $R = (R^+, R^-)$ of a later program component with a constraint $c_R = \leftarrow R^+, \{\mathrm{not}\, a \mid a \in R^-\}$ which we propagate to predecessors.



Outline

1 Motivation

2 Inconsistency Analysis

3 Trans-Unit Propagation

4 Implementation and Experiments

5 Conclusion

Implementation

We integrated the new evaluation technique into the DLVHEX-system.

DLVHEX

DLVHEX

http://www.kr.tuwien.ac.at/research/systems/dlvhex

- Based on GRINGO and CLASP from the Potassco suite.
- Supported platforms: Linux-based, OS X, Windows.
- External sources are implemented as plugins using a plugin API (available for C++ or Python).

Experiments

Abstract Configuration Problem

- Domain D, set of properties P.
- Association m of each selection $S \subseteq D$ with a set of properties $m(S) \subseteq P$.
- Set of constraints C of form $C_i = (C_i^+, C_i^-)$ with $C_i^+ \subseteq 2^P$ and $C_i^- \subseteq 2^P$.
- Goal: $S \subseteq D$ s.t. for all $(C_i^+, C_i^-) \in C$ we have $C_i^+ \not\subseteq m(S)$ or $m(S) \cap C_i^- \neq \emptyset$.

size	monolithic			splitting			tu-propagation		
	total	ground	solve	total	ground	solve	total	ground	solve
8	0.20 (0)	0.05	0.04	0.49 (0)	0.15	0.20	0.19 (0)	<0.005	0.02
10	0.45 (0)	0.22	0.10	2.17 (0)	0.81	1.11	0.33 (0)	<0.005	0.05
12	1.51 (0)	0.90	0.44	9.50 (0)	3.59	5.17	0.86 (0)	0.01	0.10
14	4.84 (0)	3.68	0.82	44.23 (0)	16.58	24.68	1.48 (0)	0.02	0.20
16	19.52 (0)	16.55	2.08	217.46 (0)	86.17	119.93	3.80 (0)	0.07	0.57
18	143.09 (3)	68.89	10.89	300.00 (10)	122.23	165.44	44.76 (1)	0.43	6.29
20	300.00 (10)	300.00	n/a	300.00 (10)	126.40	162.73	37.44 (0)	0.36	3.44
22	300.00 (10)	300.00	n/a	300.00 (10)	122.41	165.68	224.83 (6)	1.03	11.74

Table: Configuration Problem

Experiments

Diagnosis Problem $\langle \mathcal{O}_d, \mathcal{O}_p, \mathcal{H}, \mathcal{C}, P \rangle$

- Sets $\mathcal{O}_d$ and $\mathcal{O}_p$ are definite resp. potential observations.
- Set of hypotheses $\mathcal{H}$, set of constraints over the hypotheses $\mathcal{C}$.
- Logic program P which defines observations following from hypotheses.
- Goal: sets $S_\mathcal{H} \subseteq \mathcal{H}$ and $S_O \subseteq \mathcal{O}_d$ s.t.
  - (i) all answer sets of $P \cup H$, which contain all of $S_O \cup \mathcal{O}_d$, contain also $S_\mathcal{H}$,
  - (ii) $C \not\subseteq S_\mathcal{H}$ for all $C \in \mathcal{C}$.

size	monolithic			splitting			tu-propagation		
	total	ground	solve	total	ground	solve	total	ground	solve
5	0.46 (0)	0.15	0.25	0.16 (0)	0.05	0.02	0.36 (0)	0.04	0.04
10	10.16 (0)	5.94	5.08	2.86 (0)	2.29	0.67	6.38 (0)	1.01	0.89
15	276.19 (5)	258.17	40.11	121.73 (1)	96.76	22.94	105.43 (3)	15.06	13.15
20	300.00 (10)	300.00	n/a	294.97 (9)	253.97	53.55	169.56 (5)	26.54	18.64
25	300.00 (10)	300.00	n/a	300.00 (10)	270.64	45.78	187.90 (5)	29.84	18.08
30	300.00 (10)	300.00	n/a	300.00 (10)	273.50	42.07	226.51 (6)	34.92	20.24
35	300.00 (10)	300.00	n/a	300.00 (10)	276.70	37.01	299.71 (9)	44.17	23.71

Table: Diagnosis Problem

Experiments

Synthetic Set Guessing

- Program of size n:

$$P = \{dom(1..n).\ in(X) \vee out(X) \leftarrow dom(X).$$
$$someIn \leftarrow in(X).$$
$$r(X) \leftarrow \&diff[dom, out](X).\ \leftarrow r(X), someIn\}$$

size	monolithic			splitting			tu-propagation		
	total	ground	solve	total	ground	solve	total	ground	solve
5	3.13 (0)	3.02	<0.005	0.19 (0)	0.04	0.04	0.14 (0)	<0.005	0.01
6	13.59 (0)	13.48	<0.005	0.30 (0)	0.09	0.09	0.15 (0)	<0.005	0.02
7	63.11 (0)	63.00	<0.005	0.54 (0)	0.20	0.21	0.16 (0)	<0.005	0.02
8	281.10 (0)	280.99	<0.005	1.07 (0)	0.44	0.47	0.17 (0)	<0.005	0.03
...	...	...	...	...	...	...	...	...	...
12	300.00 (1)	300.00	n/a	19.97 (0)	9.25	9.83	0.24 (0)	<0.005	0.06
13	300.00 (1)	300.00	n/a	41.02 (0)	19.12	20.23	0.28 (0)	<0.005	0.07
14	300.00 (1)	300.00	n/a	85.38 (0)	39.86	42.48	0.30 (0)	<0.005	0.08
15	300.00 (1)	300.00	n/a	183.04 (0)	84.74	91.64	0.32 (0)	<0.005	0.09

Table: Synthetic Set Guessing

Outline

1 Motivation

2 Inconsistency Analysis

3 Trans-Unit Propagation

4 Implementation and Experiments

5 Conclusion

Conclusion

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.

Conclusion

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.

Conclusion

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 potential for significant speedups.

Conclusion

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 potential for significant speedups.

Future Work

- Generalization of propagation from IRs to other learned nogoods.
- Exploit structural information of the program for further improvements.

References I

Eiter, T., Fink, M., Ianni, G., Krennwallner, T., Redl, C., and Schüller, P. (2016).
A model building framework for answer set programming with external computations.
Theory and Practice of Logic Programming, 16(4):418–464.

Eiter, T., Fink, M., Krennwallner, T., Redl, C., and Schüller, P. (2014).
Efficient HEX-program evaluation based on unfounded sets.
Journal of Artificial Intelligence Research, 49:269–321.

Redl, C. (2017).
Explaining inconsistency in answer set programs and extensions.
In *Proceedings of the 14th International Conference on Logic Programming and Nonmonotonic Reasoning*.