Justification and Debugging of Answer Set Programs in ASP-PROLOG

Omar Elkhatib, Enrico Pontelli, Tran Cao Son

Knowledge representation, Logic, and Advanced Programming Laboratory
Department of Computer Science
New Mexico State University

Answer Set Programming (ASP)

- ASP: Logic Programming under answer set semantics
  - New Logic Programming Paradigm
  - Semantics of a Program = collection of answer sets (sets of atoms)
  - Rules
    \[ \text{Head} \leftarrow A_1, \ldots, A_n, \text{not } B_1, \ldots, \text{not } B_m \]
    as constraints on admissible answer sets
  - Answer Sets of a Program P correspond to the solution of the problem
  - Good Implementations (e.g., Smodels, DLV)
  - However, No Debugging systems exists.
Debugging of ASP

- Very hard, because of its highly declarative nature.
- Most of the computational details are hidden from the programmer.
- Hard to understand the reasons of the solver’s outcomes.
- Tracing is one way of Debugging ASP:
  - Large search trees
  - Intermixed proofs of different atoms

Justification of ASP

- **Justification** is a new approach:
  - Creates proof graphs for each true atom
  - Creates counter-examples for false atoms
  - Originally developed for well-founded semantics in XSB.
  - In ASP, it provides a proof of why an atom is or is not in an answer set.
  - We develop justification for ASP and integrate it into the ASP-PROLOG System.
**ASP-PROLOG System**

- It provide a tight and semantically well-defined integration of Prolog and Answer Set Programming (ASP).
- The combined system enhances the expressive power of ASP:
  - Dynamic ASP modules (add/remove rules)
  - Reasoning about ASP modules from Prolog
  - Reasoning about collections of answer sets from Prolog
- The system is developed using the module and class capabilities of CIAO Prolog.

**System Download:** [www.cs.nmsu.edu/~okhatib/asp_prolog.html](http://www.cs.nmsu.edu/~okhatib/asp_prolog.html)

Under Linux.

---

**Justification of ASP Programs**

- For ASP P and model M:
  - True literal L means:
    - L in M if L is an atom (L=a).
    - L not in M if L is a negated atom (L=not a).
  - False literal L means:
    - L not in M if L is an atom (L=a).
    - L in M if L is a negated atom (L=not a).
  - A rule r is **active** if all literals in body of r are true wrt M.
  - Locally consistent explanation (LCE):
    - A∈M: ζ(A,M) = set of the bodies of the active rules that have A as head (i.e., reasons for A’s truth)
    - A∈M: ζ(A,M) = a collection of literals such that we have exactly one false literal per rule for all rules which has head A (i.e., reasons for A’s falsity)
### Justification Example

\[
\begin{align*}
a &\ :- \ b. \\
b &\ :- \ a. \\
a &\ :- \ c. \\
c. \\
d &\ :- \ s. \\
d &\ :- \ b. \\
s &\ :- \ a, w. \\
\end{align*}
\]

\[M = \{a,b,c,d\}\]

\[\zeta_{a,M} = \{b\}, \{c\}\].

\[\zeta_{d,M} = \{b\}\].

\[\zeta_{s,M} = \{w\}\].

Positive cycles problem.

### Justification of ASP

Justification of ASP P is a graph \(J=(V,E)\).

- **If A in M:**
  - If A is a fact then \((A, \text{fact})\) in E.
  - If there is rule r : all literals in body of r are not in a positive cycle with A, then \((A, B)\) in E, \(\forall B\) in body of r.
  - No other outgoing edges from A are possible.

- **If A not in M:**
  - If no rule defined for A then \((A, \text{no_support})\) in E.
  - For each rule r with head A, choose one false literal B in body of r, then \((A, B)\) in E.
  - No other outgoing edges from A are possible.
Justification Example

\[
\begin{align*}
a & : \neg b. \\
b & : \neg a. \\
a & : e. \\
e & : a. \\
c & : a. \\
d & : b. \\
M_1 & = \{a, c, e\}. \\
M_2 & = \{b, d\}.
\end{align*}
\]

Positive and negative cycles.

r-Justification

- Break all negative cycles.
- Define: Assumption Set \( \text{AS}(M) = \) set of all atoms that satisfy the following conditions:
  - Atom A is false wrt M.
  - A appears in negation form in P.
  - A appears in a negative cycle in E
- R-justification:
  Add: If A \( \in \text{AS} \) then (A, assume) \( \in E \).
r-Justification Example

a :- not b.
b :- not a.
a :- e.
e :- a.
c :- a.
d :- b.
M1={a,c,e).
M2={b,d}.
\( AS(M1) = \{b\} \).
\( AS(M2) = \{a\} \).

System Implementation

- Justification is integrated into ASP-PROLOG.
- Justification is written in CIAO-PROLOG.
- lparse/smodels is used to find answer set models.

- Predicate added for programmer:
  - Justify_atoms(model_name, atom_list).
  - Output: text format and graph format (uDrawGraph).
  - System shows the rules that cause the justification.

- System can handle all type of lparse/smodels rule: cardinality, weight and choice rules.
ASP-PROLOG System Overview

Prolog modules

ASP modules

New Prolog modules

Interaction Prolog - ASP

Prolog modules

ASP program

Justify_atoms

Justification file

New Prolog modules

atom

rule

models

uDrawGraph

Graph
Example

a :- 2 \{b, c, not d\} 2.
b :- not f.
f :- not b.
c :- g.
g.
d :- not e.
e :- not d.

M1={b,c,g,d,a}.
M2={b,c,g,e}.
M3={f,c,g,d}.
M4={f,c,g,e,a}.

Conclusion & Future Work

- Justification is one type of debugging. It is used in this paper to justify ASP models.

- Partial justification of answer sets is under investigation. Allow users to justify atoms in the middle of computation.

- Work is in progress to present the non-ground rules defining the atom.