Managing Change in Answer Set Programs: The Logical Approach

James Delgrande

Simon Fraser University
Canada

LPNMR 2013

Introduction

ASP

- Declarative paradigm for problem solving
- Research in ASP has broadly focussed on:
  - ASP solvers
  - efficiency
  - language extensions
  - ☞ syntactic aspects
- More recently: Logical view of AS programs
  - based on work involving strong equivalence, SE models, and the connection with the logic of H&T
  - program can be viewed as an abstract entity.
  - ☞ semantic aspects

Introduction

Program Change

Recently too, work has arisen in which a *logic program* is subject to change.

- revision
- update
- merging
- forgetting

☞ This is quite different from the "standard" focus on solvers, efficiency, etc.

Introduction: Program Change

Claim

- The general area of logic program change is interesting and potentially important.
- A logical approach is suitable.
  - In viewing a program as an abstract object, syntax is suppressed, and one can focus on the underlying logic.
  - One can exploit notions having to do with interpretations or with logical consequence.
- Can take advantage of work from classical logic, and more recent work with Horn theories, for addressing change in ASP.

☞ In this talk I'll expand on, and try to justify, these claims.

Outline

√ 1. Introduction
2. Background
   - ASP
   - Belief change in classical logic
   - LP Update
3. Belief change in ASP: The logical approach
   - Case studies
     - A class of merging functions in ASP
     - A specific (distance-based) revision operator in ASP
     - Forgetting in DLPs.
   - Belief change in ASP and beyond
4. Conclusion

Background: Answer Set Programming

Definition ([LW92])

- A *generalised logic program* (*GLP*) is a finite set of rules of the form

$$a_1; \ldots; a_m;\ \sim b_{m+1}; \ldots; \sim b_n \ \leftarrow \\ c_{n+1}, \ldots, c_o,\ \sim d_{o+1}, \ldots, \sim d_p$$

  where a_i, b_j, c_k, d_l are *atoms*.
- A rule r is called *disjunctive* if $m = n$.
- Other classes: *normal* (m=n=1), *positive*, *Horn*

ASP

- Can write a rule r as

$$H(r)^+; {\sim}H(r)^- \leftarrow B(r)^+, {\sim}B(r)^-.$$

- For set of atoms Y and program P the *reduct* P^Y, is defined as

$$\{H^+(r) \leftarrow B^+(r) \mid r \in P, H^-(r) \subseteq Y, B^-(r) \cap Y = \emptyset\}.$$

- A set X of atoms is an *answer set* of a program P if it is a subset-minimal model of P^X.

ASP and Nonmonotonicity

Example

- $\{p \leftarrow \sim q\}$ has answer set $\{p\}$.
- $\{p \leftarrow \sim q\} \cup \{q\}$ has answer set $\{q\}$.

☞ I.e. ASP is nonmontonic wrt answer sets.

Strong Equivalence and SE Models [LPV01],[Tur03]

Definition

Programs P and Q are *strongly equivalent* iff for every program R, $AS(P \cup R) = AS(Q \cup R)$.

Definition

- An *SE interpretation* is a pair (X, Y) of interpretations such that $X \subseteq Y \subseteq \mathcal{A}$.
- An SE interpretation (X, Y) is an *SE model* of a program P if $Y \models P \quad \text{and} \quad X \models P^Y$.

Obtain:

- P and Q are *strongly equivalent* iff $SE(P) = SE(Q)$.
- Y is an AS of program P iff: $(X, Y) \in SE(P) \implies X = Y$.

SE Models and Programs

Definition

- A set S of SE interpretations is *well-defined* if, for each $(X, Y) \in S$, also $(Y, Y) \in S$.

Obtain

- For each well-defined set S of SE interpretations, there exists a GLP P such that $SE(P) = S$, and vice versa.
- Programs meeting these conditions can be constructed.
- Other classes of programs (disjunctive, etc.) can be similarly characterised.

Key Point

SE models provide a monotonic foundation for ASP.

The Logical View

Via strong equivalence, and its characterisation in terms of SE models, we can view a program at the *knowledge level*

- Can consider a program as an abstract object
- Independent of syntax (i.e. *how* information is expressed)
- Provides clarity to fundamental or representational issues

The Logical View

Via strong equivalence, and its characterisation in terms of SE models, we can view a program at the *knowledge level*

- Can consider a program as an abstract object
- Independent of syntax (i.e. *how* information is expressed)
- Provides clarity to fundamental or representational issues

Arguably, for program change, we are foremost interested in the *logical content* of the program.

- This suggests that the logical approach is most appropriate for investigating properties of various types of change.
- Once an approach to program change is (somewhat) understood, computational issues can be better addressed.

Background: Belief Change

Belief change

Belief change studies how an agent may change its beliefs in the face of new information.

- It got going in the early 1980's with the work of Alchourrón, Gärdenfors, and Makinson, the so-called *AGM approach*.
- The AGM approach assumes that the underlying logic subsumes classical PC.

Classes of Belief Change Functions

Domain-dependent approaches

- Revision: An agent consistently accommodates new information
- Merging: The knowledge of 2+ agents is consistently amalgamated
- Contraction: An agent's ignorance increases
- Update: An agent's knowledge changes due to a change in the world.

Domain-independent approaches

- Expansion: New information is added to the agent's beliefs.
- Forgetting: Essentially, the agent's vocabulary decreases.

Defining Belief Change Functions

Domain-dependent approaches

☞ Logical considerations alone are not sufficient to determine domain-dependent functions.

This suggests the approaches:

1. Characterise a class of domain-dependent functions, *or*
2. Define a specific instance of a class conforming to the formal characterisation.

Domain-independent approaches

☞ Logical considerations are sufficient to determine domain-independent functions.

This suggests:

3. Provide an appropriate definition

Aside: Logic Program Update

☞ In the last 15-20 years or so, change in ASP has most often been addressed as the problem of *Logic Program Update*

Standard Approach to LP Update

Input:	A sequence: $(P_1, P_2, \ldots, P_n)$ of programs.
Intuition:	For $i > j$, P_i has higher priority than P_j.
Result:	A *set of answer sets* that respects the ordering.

Suggest

- This is not really update in the AGM sense, since one doesn't obtain an updated program.
- Rather, best regarded as dealing with priorities among rules

Belief Change in ASP

The Logical Approach to Change in ASP

Once we adopt the logical view of ASP:

- We can use intuitions from classical belief change to guide change in ASP.
- Many techniques from belief change can be adapted to ASP.
- Not necessarily straightforward, but once done (where possible), can pretty much get results from classical logic.

Recall: Defining Belief Change Functions

☞ Recall we identified the following broad approaches to defining belief change functions:

Domain-dependent approaches

1. Characterise a class of domain-dependent functions, *or*
2. Define a specific instance of a class conforming to the formal characterisation.

Domain-independent approaches

3. Provide an appropriate definition

Belief Change in ASP

Case Studies

I'll illustrate the application of "classical" belief change to ASP by three examples:

1. A characterisation of a class of merging functions in ASP
2. An instance of (distance-based) revision in ASP
3. A definition of forgetting in ASP

Caveat

This will be rather informal (though the underlying approaches have been formally specified).

Example 1: Merging in ASP

IC-Merging in ASP

- In AGM, characterisations are given by 2 means:

  Constructions: A general technique is given whereby belief change functions are characterised.

  Postulates: Criteria that bound any "rational" function.

  Then: Show construction $\approx$ postulates

I'll sketch how this can be done for the approach of [KP02]

- See the talks on Thursday for details on how this is done for revision.
- Also see [SL10] for update in GLPs.

Merging in ASP: The problem

Given

A *belief profile* of programs

$$\Psi = \langle P_1, \ldots, P_n \rangle$$

and a program U specifying *global constraints*.

Goal

Merge the programs in Ψ into a program, P', such that

$$SE(P') \subseteq SE(U).$$

Denote by: $P' = \Delta_U(\Psi)$.

Merging: Construction

Corresponding to a belief profile Ψ is a total preorder $\preceq_\Psi$ over interpretations.

- $\preceq_\Psi$ expresses the consensus of the plausibility of interpretations according to Ψ.

Merging: Construction

Corresponding to a belief profile Ψ is a total preorder $\preceq_\Psi$ over interpretations.

- $\preceq_\Psi$ expresses the consensus of the plausibility of interpretations according to Ψ.

Intuitions regarding $\preceq_\Psi$

1. (X, Y) is not more preferred than (Y, Y).
2. Any model of all the programs in Ψ is minimal in the ordering
3. No program in Ψ is "favoured" over another in $\preceq_\Psi$.
4. If w_1 is ranked over w_2 by both Ψ_1 and Ψ_2, then w_1 is ranked over w_2 in $\Psi_1 \sqcup \Psi_2$.

Formalising these intuitions yields a *syncretic assignment*.

Merging: Postulates

Don't Read!

Merging: Postulates

Don't Read! Merging: Postulates

Definition

Δ is a *GLP IC merging operator*, if it satisfies the postulates.

(*IC*0) $\Delta_U(\Psi) \models_s U$.

(*IC*1) If $U \not\models_s \bot$ then $\Delta_U(\Psi) \not\models_s \bot$.

(*IC*2) If $\sqcap\Psi \sqcap U \not\models_s \bot$ then $\Delta_U(\Psi) \equiv_s \sqcap\Psi \sqcap U$.

(*IC*3) If $\Psi_1 \equiv_s \Psi_2$ and $U_1 \equiv_s U_2$ then $\Delta_{U_1}(\Psi_1) \equiv_s \Delta_{U_2}(\Psi_2)$.

(*IC*4) If $P_1 \models_s U$ and $P_2 \models_s U$ then $\Delta_U(P_1 \sqcup P_2) \sqcap P_1 \not\models_s \bot$ implies $\Delta_U(P_1 \sqcup P_2) \sqcap P_2 \not\models_s \bot$.

(*IC*5) $\Delta_U(\Psi_1) \sqcap \Delta_U(\Psi_2) \models_s \Delta_U(\Psi_1 \sqcup \Psi_2)$.

(*IC*6) If $\Delta_U(\Psi_1) \sqcap \Delta_U(\Psi_2) \not\models_s \bot$ then $\Delta_U(\Psi_1 \sqcup \Psi_2) \models_s \Delta_U(\Psi_1) \sqcap \Delta_U(\Psi_2)$.

(*IC*7) $\Delta_{U_1}(\Psi) \sqcap U_2 \models_s \Delta_{U_1 \sqcap U_2}(\Psi)$.

(*IC*8) If $\Delta_{U_1}(\Psi) \sqcap U_2 \not\models_s \bot$ then $\Delta_{U_1 \sqcap U_2}(\Psi) \models_s \Delta_{U_1}(\Psi) \sqcap U_2$.

Merging: A Representation Result

Outcome

- These two approaches coincide for GLPs.
  - I.e. any function that satisfies the postulates is captured by a syncretic assignment, and vice versa.
- This extends to other classes of programs, though less straightforwardly.
- Other types of merging operators can be similarly captured.

Moral

With a bit of work, characterisations from classical belief change can be ported over to ASP.

Example 2: Distance-Based Revision [DSTW13]

Goal

- Define a specific revision operator $P * Q$ between programs.
- Examine properties of the approach

Intuition

Characterise $P * Q$ by those SE models of Q that are *closest* to the SE models of P.

Example 2: Distance-Based Revision [DSTW13]

Goal

- Define a specific revision operator $P * Q$ between programs.
- Examine properties of the approach

Intuition

Characterise $P * Q$ by those SE models of Q that are *closest* to the SE models of P.

Q: What do we mean by *closeness* between SE models?

Example 2: Distance-Based Revision [DSTW13]

Goal

- Define a specific revision operator $P * Q$ between programs.
- Examine properties of the approach

Intuition

Characterise $P * Q$ by those SE models of Q that are *closest* to the SE models of P.

Q: What do we mean by *closeness* between SE models?
A [Sat88]: w is closer to w_1 than w_2 if: $\mathit{Diff}(w, w_1) \subset \mathit{Diff}(w, w_2)$

Example 2: Distance-Based Revision [DSTW13]

Goal

- Define a specific revision operator $P * Q$ between programs.
- Examine properties of the approach

Intuition

Characterise $P * Q$ by those SE models of Q that are *closest* to the SE models of P.

Q: What do we mean by *closeness* between SE models?
A [Sat88]: w is closer to w_1 than w_2 if: $\mathit{Diff}(w, w_1) \subset \mathit{Diff}(w, w_2)$

☞ So extend this notion to SE interpretations

Example 2: Distance-Based Revision

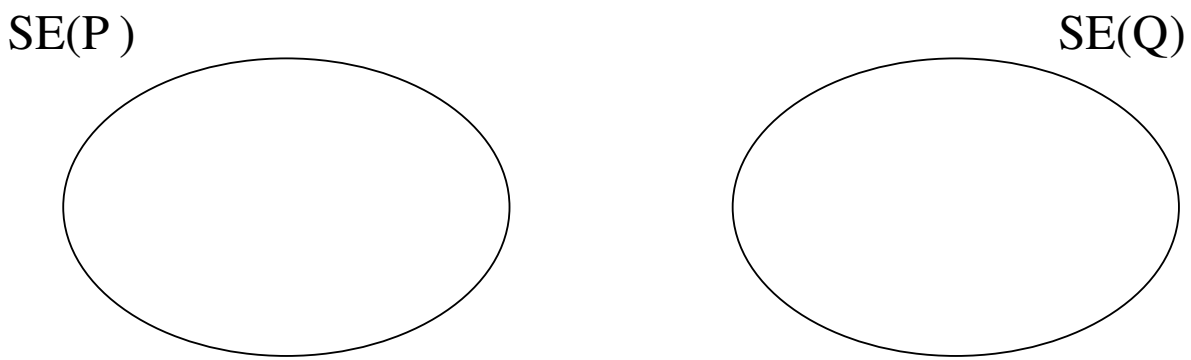
Intuition

Characterise $P * Q$ by those SE models of Q that are *closest* to the SE models of P.

Example 2: Distance-Based Revision

Intuition

Characterise $P * Q$ by those SE models of Q that are *closest* to the SE models of P.

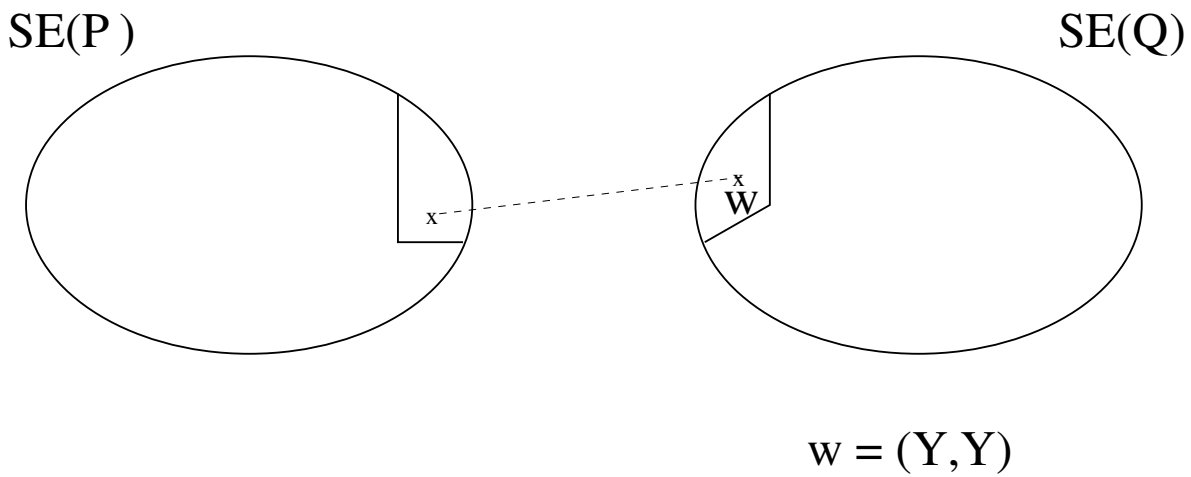


Example 2: Distance-Based Revision

Intuition

Characterise $P * Q$ by those SE models of Q that are *closest* to the SE models of P.

1. Find those $(Y, Y) \in SE(Q)$ closest to $SE(P)$. Call this set $\mathcal{S}$.

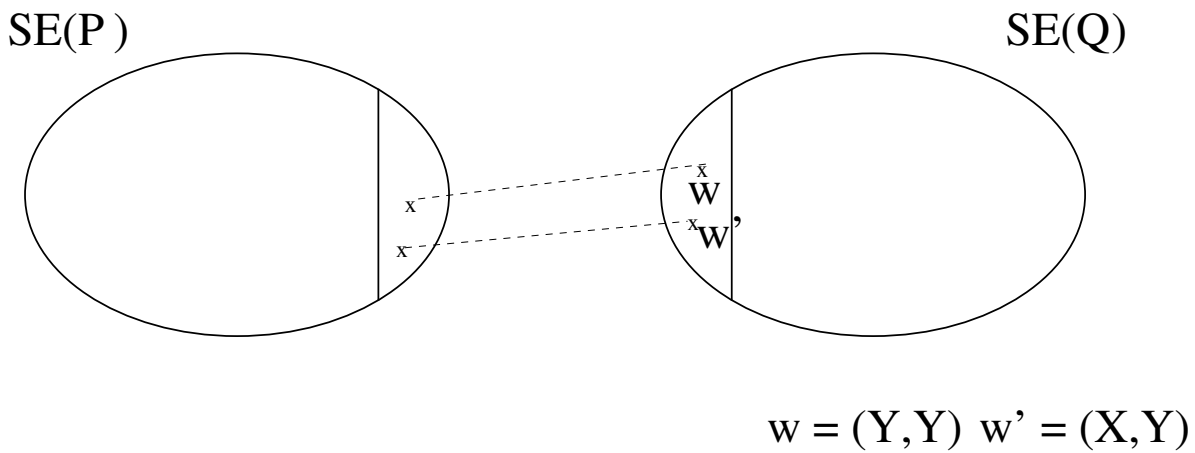


Example 2: Distance-Based Revision

Intuition

Characterise $P * Q$ by those SE models of Q that are *closest* to the SE models of P.

1. Find those $(Y, Y) \in SE(Q)$ closest to $SE(P)$. Call this set $\mathcal{S}$.
2. Find those $(X, Y) \in SE(Q)$ closest to $SE(P)$ s.t. $(Y, Y) \in \mathcal{S}$.



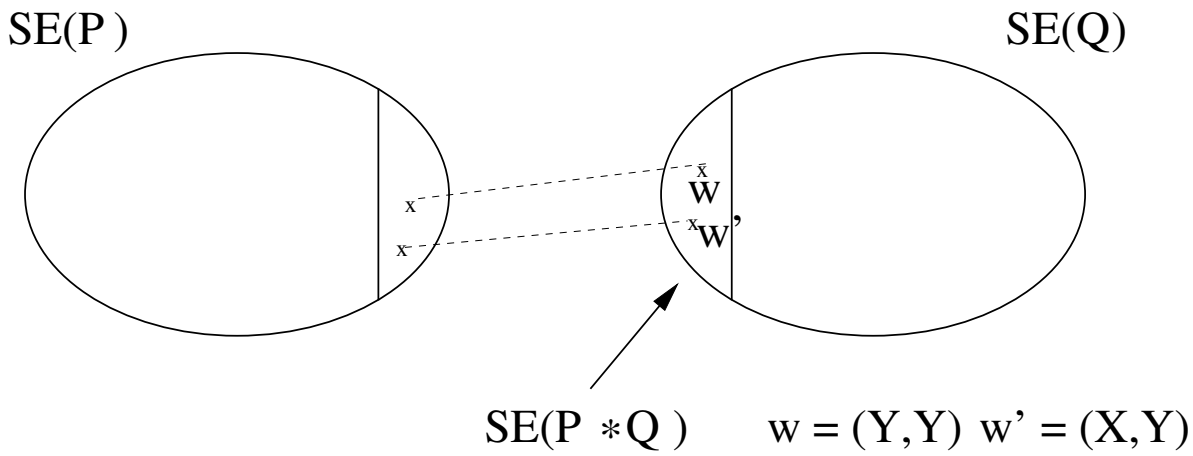
Example 2: Distance-Based Revision

Intuition

Characterise $P * Q$ by those SE models of Q that are *closest* to the SE models of P.

1. Find those $(Y, Y) \in SE(Q)$ closest to $SE(P)$. Call this set $\mathcal{S}$.
2. Find those $(X, Y) \in SE(Q)$ closest to $SE(P)$ s.t. $(Y, Y) \in \mathcal{S}$.

- $P * Q$ is a logic program whose SE models are given in 1 and 2.



Examples

Example

This gives reasonable results:

$$\{p; q \leftarrow\} \quad * \quad \{\bot \leftarrow q\} \quad \equiv_s \quad \left\{ \begin{array}{l} p \leftarrow \\ \bot \leftarrow q \end{array} \right\}$$

$$\left\{ \begin{array}{l} p \leftarrow \sim q \\ q \leftarrow \sim p \end{array} \right\} \quad * \quad \{p \leftarrow q\} \quad \equiv_s \quad \left\{ \begin{array}{l} p \leftarrow q \\ p \leftarrow \sim q \end{array} \right\}$$

$$\left\{ \begin{array}{l} \bot \leftarrow \sim p \\ \bot \leftarrow \sim q \end{array} \right\} \quad * \quad \{\bot \leftarrow p, q\} \quad \equiv_s \quad \left\{ \begin{array}{l} \bot \leftarrow \sim p, \sim q \\ \bot \leftarrow p, q \end{array} \right\}$$

Outcome

Properties

The first 7 of the AGM revision postulates are satisfied as well as properties that have been proposed for LP update.

Conclude

Constructions for specific revision (and by extension, other operators) carry over to logic programs.

Example 3: Forgetting in ASP

Goal

- Define forgetting semantically and syntactically in ASP
- Exhibit properties of the approach

Approach

Examine forgetting in propositional logic, then apply the techniques to logic programs.

Forgetting in PC

Definition

The standard definition for forgetting atom p from a KB K is

$$K[p/0] \vee K[p/1]$$

Suggest

A more appropriate knowledge level definition, for a language over atoms $\mathcal{P}$, is

$$Forget(K, p) \doteq Cn(K) \cap \mathcal{L}_{\mathcal{P} \setminus p}$$

Forgetting in PC

Definition

The standard definition for forgetting atom p from a KB K is

$$K[p/0] \vee K[p/1]$$

Suggest

A more appropriate knowledge level definition, for a language over atoms $\mathcal{P}$, is

$$Forget(K, p) \doteq Cn(K) \cap \mathcal{L}_{\mathcal{P} \setminus p}$$

☞ This is not computationally friendly!

Forgetting in PC

Syntactic characterisation

- Assume that K is in CNF.
- Let:
  - $Res(K, p)$ be the set of resolvents of clauses in K on p.
  - $K \downarrow p$ be the clauses in K not mentioning p.
- Then:

$$Forget(K, p) \equiv Res(K, p) \cup (K \downarrow p)$$

☞ We can use this a guide for defining forgetting in ASP.

Forgetting in ASP

Definition

- Let $\mathcal{D}_{\mathcal{A}}$ be the language of DLPs over $\mathcal{A}$.
- Define forgetting as:

$$Forget(P, p) \doteq Cn(P) \cap \mathcal{D}_{\mathcal{A} \setminus p}$$

where $Cn(P) = \{r \mid P \models_s r\}$

Forgetting in ASP

Syntactic characterisation

- [Won08] gives a set of sound and complete inference rules for SE consequence.
- Let:
  - $Res(P, p)$ be the set of "resolvents" of rules[a] in P on p.
  - $P \downarrow p$ be the rules in P not mentioning p.
- Then:

$$Forget(P, p) \equiv_s Res(P, p) \cup (P \downarrow p)$$

[a] [Del13]

Forgetting in ASP

Example

Let: $P = \{p \leftarrow \sim q,\ \ q \leftarrow r,\ \ b \leftarrow p,\ \ p \leftarrow a\ \}$

Then:

$$Forget(P, p) \equiv_s \{q \leftarrow r,\ \ b \leftarrow a\ \}$$
$$Forget(P, q) \equiv_s \{b \leftarrow p,\ \ p \leftarrow a\ \}$$
$$Forget(P, r) \equiv_s \{p \leftarrow \sim q,\ \ b \leftarrow p,\ \ p \leftarrow a\ \}$$

Properties

Easy to show the approach has appropriate properties.

Conclude

Again, can use classical logic to guide research.

Relevance of Belief Change in ASP?

Ask

Ok, so beyond theoretical notions, why is studying program change interesting?

Relevance of Belief Change in ASP?

Ask

Ok, so beyond theoretical notions, why is studying program change interesting?

Answer

Any KB will be subject to various types of change, and it's imperative to have a principled account of forms of change.

- ☞ In the next few slides I'll survey possible applications of change in ASP and related areas.

Examples: Belief Change in ASP

The classical belief change operators are relevant to ASP

Revision: Any reasonably large program will evolve over time.

Merging: May want to combine the information in 2+ programs.

Forgetting: Applicable for, e.g.:

- Ignoring part of a program
- Predicate hiding (or views)

Belief Change in Areas Related to ASP

Suggest

Addressing belief change in ASP may contribute to understanding change in related areas.

Example: Reasoning about action

Action languages have tight connections with ASP

Applications of belief change:

- Changing an action theory (*revision*)
- Progressing a KB (*forgetting*)
  (Also relevant for *incremental ASP*)

Yet More Areas

- Argumentation
  - may want to *revise* or *merge* argumentation structures
- Policies
  - Policy change (*revision*)
- Negotiation using ASP
  - An instance of *merging*
- Causal theories

☞ Basically any area where a rule-based approach + naf is appropriate.

Conclusion

Summary

- Belief change in ASP is interesting and potentially important
- Advocated the logical approach for addressing change
- Argued that one can use results and intuitions from belief change in classical logic to guide work.

Potential Benefits

- The notion of change in ASP (and related areas) can be put of a more solid theoretical foundation.
- Addressing belief change in ASP sheds light on belief change in general.
- Classical belief change has had few applications; ASP and related areas may provide a test bed.

James Delgrande.

Forgetting in logic programs.
in preparation, 2013.

James Delgrande, Torsten Schaub, Hans Tompits, and Stefan Woltran.

A model-theoretic approach to belief change in answer set programming.
ACM Transactions on Computational Logic, 14(2), 2013.

S. Konieczny and R. Pino Pérez.

Merging information under constraints: A logical framework.
Journal of Logic and Computation, 12(5):773–808, 2002.

V. Lifschitz, D. Pearce, and A. Valverde.

Strongly equivalent logic programs.
ACM Transactions on Computational Logic, 2(4):526–541, 2001.

V. Lifschitz and T. Woo.

Answer sets in general nonmonotonic reasoning (preliminary report).
In B. Nebel, C. Rich, and W. Swartout, editors, *Proceedings of the Third International Conference on the Principles of Knowledge Representation and Reasoning*, pages 603–614, Cambridge, MA, 1992.

K. Satoh.

Nonmonotonic reasoning by minimal belief revision.
In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 455–462, Tokyo, 1988.

Hudson Turner.

Strong equivalence made easy: nested expressions and weight constraints.
Theory and Practice of Logic Programming, 3(4):609–622, 2003.

Ka-Shu Wong.

Sound and complete inference rules for se-consequence.
Journal of Artificial Intelligence Research, 31(1):205–216, January 2008.