On the Existence of Answer Sets in Normal Extended Logic Programs

Martin Caminada
Utrecht University
The Netherlands

Chiaki Sakama
Wakayama University
Japan

NRAC 2007, January

Answer Set Programming: ASP

In ASP a problem is represented by an extended logic program (ELP) whose declarative meaning is given by the **answer set semantics.**

Program: $r \leftarrow p,$

$\neg r \leftarrow q,$

$p \leftarrow \textbf{not}\ q,$

$q \leftarrow \textbf{not}\ p,$

Answer Sets: { p , r }, { q , r }

The problem of ASP

- Answer set semantics is FRAGILE**!** Relatively **small pieces** of information may cause a **total absence** of answer sets.
- A notorious example is the existence of a "**negative loop**" such as p← **not** p*.* A program including such **anomalous information** is **unusual** anyway.
- **More serious problem** is: programs that are seemingly natural and do not include any such anomalous information often **fail to have** answer sets.

Example: Married John

Consider the following information:

(1) John wears something that looks like a wedding ring.

(2) John parties with his friends until late.

(3) Someone wearing a wedding ring is usually married.

(4) A party-animal is usually a bachelor.

(5) A married person, by definition, has a spouse.

(6) A bachelor, by definition, does not have a spouse.

It is naturally represented by the program.

r ←, p ←,

m ← r, **not** ¬ m, b ← p, **not** ¬ b,

hs ← m, ¬ hs ← b.

(r:ring, p:party, m:married, b:bachelor, hs: has spouse)

! This program has no answer set, even though it contains no negative loops.

Justification: S={ r, p, m, hs } or { r, p, b, ¬hs } produces the reduct

r ←, p ←, m ← r, b ← p, hs ← m, ¬ hs ← b,

which has the minimal closed set Lit. And S=Lit produces the reduct

r ←, p ←, hs ← m, ¬ hs ← b,

which has the minimal closed set { r, p }.

The problem does not arise if we encode the same knowledge in Reiter's default logic:

$$r, \quad p, \quad m \supset hs, \quad b \supset \neg hs,$$

$$r : m / m, \qquad p : b / b.$$

The above default theory has two extensions:

$\{ r, p, b, \neg hs, \neg m \}$ and $\{ r, p, m, hs, \neg b \}$.

* This is a normal default theory that always has an extension.

Why such difference happens?

- In extended LPs, every rule is interpreted as a one way "**inference rule**".
- A rule does not have a **contrapositive meaning** even if it is a definite rule without default negation.
- In default theories, definite information is represented by **first-order formulas** having contrapositive meaning.

To bridge the gap between ELPs and default theories, [Gelfond and Lifschitz] characterize the program in terms of non-normal default theories:

r, p, m: / hs, b: / ¬ hs,

r : m / m, p : b / b,

which has no extension.

- The fact that ELPs with **normal default rules** may not have any answer set is **discouraging**. Because normal default theories are "a very large and natural class of default theories" [Reiter].
- A knowledge engineer can encode problems like "Married-John" in a normal default theory in a straightforward manner, while he/she may **fail to encode the same problem in a logic program.**

Questions

- How can one provide a **natural meaning** of **an ELP with normal default rules**?
- Is there any condition to **guarantee the existence of answer sets** of ELPs with **normal default rules**?

Our solution:

- We introduce the class of **normal ELPs** whose default rules are in the form of **normal default rules**.
- We provide **syntactic conditions to guarantee the existence of answer sets in normal ELPs**.

Terms and Notations

- A **strict rule** is of the form:

  $c \leftarrow a_1, \ldots, a_n$ (a_i: literal, c: (nonempty) literal)

- A **defeasible rule** is of the form:

  $c \leftarrow a_1, \ldots, a_n$, **not** $b_1, \ldots,$ **not** b_m (b_i: literal, $m \neq 0$)

  A defeasible rule is **normal** if it is of the form:

  $c \leftarrow a_1, \ldots, a_n$, **not** $\neg c$

- A **rule** is either strict or defeasible.

- A **program** P is a finite set strict(P) $\cup$ defeasible(P) where strict(P) (resp. defeasible(P)) is the set of all strict (resp. defeasible) rules in P.

- We handle **finite propositional programs**.

Let s_1 ,s_2 , s_3 be strict rules.

- s_2 is a **transpositive** version of s_1 iff

  $s_1 = c \leftarrow a_1, \ldots, a_n$ and

  $s_2 = \neg a_i \leftarrow a_1, \ldots, a_{i-1}, \neg c, a_{i+1}, a_n$ for some $1 \leqq i \leqq n$

- s_3 is a **transitive** version of s_1 and s_2 iff

  $s_1 = c \leftarrow a_1, \ldots, a_n$

  $s_2 = a_i \leftarrow b_1, \ldots, b_m$ for some $1 \leqq i \leqq n$, and

  $s_3 = c \leftarrow a_1, \ldots, a_{i-1}, b_1, \ldots, b_m, a_{i+1}, \ldots, a_n$

- s_2 is an **antecedent cleaned** version of s_1 iff

  $s_1 = \neg a_i \leftarrow a_1, \ldots, a_i, \ldots, a_n$

  $s_2 = \neg a_i \leftarrow a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n$

Let S be a set of strict rules. Then,

- S is **closed under transposition** iff for each rule $s_1 \in S$, if s_2 is a transpositive version of s_1 then $s_2 \in S$.
- S is **closed under transitivity** iff for each rule $s_1, s_2 \in S$, if s_3 is a transitive version of s_1 and s_2 then $s_3 \in S$.
- S is **closed under antecedent cleaning** iff for each rule $s_1 \in S$, if s_2 is an antecedent cleaned version of s_1 then $s_2 \in S$.

Normal Extended Logic Programs

- A program P is called **normal** iff

  1. strict(P) is **closed under transposition, transitivity, and antecedent cleaning**, and

  2. defeasible(P) consists of **normal rules** only.

- **Proposition**

Given an ELP P such that defeasible(P) consists of normal rules only, there is a unique normal ELP which is obtained by closing strict(P) under three operations.

⌘ **Theorem**

Every normal ELP has at least one answer set.

⌘ Example (Married-John)

r ←, p ←,

m ← r, **not** ¬ m, b ← p, **not** ¬ b,

hs ← m, ¬ hs ← b,

¬ m ← ¬ hs, ¬ b ← hs,

¬ m ← b, ¬ b ← m.

The above program has two answer sets:

{ r, p, m, ¬b, hs } and { r, p, b, ¬m, ¬hs }.

Connection to Default Logic

- Given a normal ELP P, its **associated default theory** $\Delta = (W,D)$ is defined as:

$$W = \{ c \vee \neg a_1 \vee \cdots \vee \neg a_n \mid c \leftarrow a_1, \ldots, a_n \in \text{strict}(P) \}.$$

$$D = \{ a_1 \wedge \cdots \wedge a_n : c \,/\, c \mid c \leftarrow a_1, \ldots, a_n, \textbf{not } \neg c \in \text{defeasible}(P) \}.$$

- **Theorem** (not in the paper)

There is a 1-1 correspondence between answer sets of P and extensions of Δ.

Application (1):
Constraint Satisfacion Problems

- Put N pigeons into M holes so that there is at most one pigeon in a hole. The program is coded as [Niemela]:

pos(P,H) ← pigeon(P), hole(H), **not** ¬ pos(P,H),

¬pos(P,H) ← pigeon(P), hole(H), **not** pos(P,H),

← pigeon(P), hole(H), hole(K), pos(P,H), pos(P,K), H≠K,

← pigeon(P), **not** hashole(P),

hashole(P) ← pigeon(P), hole(H), pos(P,H),

← pigeon(P), pigeon(Q), hole(H), pos(P,H), pos(Q,H), P≠Q.

where pos(P,H) represents a legal position of a pigeon P in a hole H.

- The above program is **beyond** the class of normal ELPs even after closing the strict rules under three operations, due to the existence of **integrity constraints**.
- By definition, normal ELPs do **not** contain integrity constraints – rules with empty heads.
- In fact, $\leftarrow F$ is **semantically equivalent** to $c \leftarrow F$, **not** c for any literal c under the answer set semantics, which is **not** a normal default rule.
- In ASP, however, **integrity constraints play an important role**; imposing conditions on the solutions or pruning unwanted answer sets.

Handling Constraints in Normal ELPs

- An **integrity constraint** is of the form:

  $\leftarrow a_1, \ldots, a_n$, **not** $b_1, \ldots,$ **not** b_m

- **Proposition** Let P be a normal ELP and IC a set of integrity constraints. If S is an answer set of $P \cup IC$, then S is an answer set of P.
- $P \cup IC$ eliminates answer sets of P which do not satisfy the constraints.
- **Normal rules + integrity constraints** can naturally encode many CSP problems and combinatorial problems such as the n-queen problem, the k-colorability problem, etc.

Application (2): Multiagent Systems

- Suppose a **multiagent system** which consists of *k* agents.
- An agent *i* has a knowledge base P_i in ASP, where definite knowledge is shared by every agent $\mathrm{strict}(P_1) = \cdots = \mathrm{strict}(P_k)$ (e.g., **a shared ontology**), while individual agents have their own default knowledge as $\mathrm{defeasible}(P_i)$.
- In this situation, **combining knowledge bases** $\bigcup_{i=1}^{k} P_i$ **may produce no answer set**, even if each individual program P_i is consistent and meaningful.
- When every P_i is coded in **normal ELPs**, $\bigcup_{i=1}^{k} P_i$ is also a normal ELP and **always has a consistent answer set** as far as the shared definite knowledge is consistent.

Application (3): Program Development

- In ASP, a piece of information introduced to a program may lead to a **total collapse** of all entailment (no answer set).
- This is a serious problem when dealing with knowledge bases that **evolve with time and change dynamically**.
- In **normal ELPs**, on the other hand, such a “total collapse” problem **never happens**. A knowledge engineer feels free to update a part of a program without spoiling the global meaning as far as defeasible information is encoded in the form of normal rules.

Discussion: ASP vs. Normal ELP

- Normal ELPs generally **increase the entailment power of ASP**.
- Given the information,

  mammal $\leftarrow$ human, $\quad\quad$ $\neg$ mammal $\leftarrow$

  ASP does **not** entail $\neg$human, while it is entailed in the normal ELP after closing the program by 3 operations.
- **Strict rules naturally have their contrapositive meaning in many programs**. ASP does not read a strict rule in a contrapositive way, which may result in missing information **implicitly represented** in a program.

Discussion: ASP vs. Classical Logic

- Concealing contrapositive meaning makes the theory of LP **different** from first-order logic.
- For example, suppose an agent having the information $P=\{ p \leftarrow q \}$. When new information $Q=\{ \neg p \}$ has arrived, what conclusion should be derived by the agent?
- If P is read as a **Horn logic program**, $\neg q$ is derived from $P \cup Q$ because it is logically equivalent to $\{ p \vee \neg q, \neg p \}$. If P is read as an **ELP**, $\neg q$ is **not** derived.
- The same program may have **different** meaning depending on its reading, **even in the absence of defeasible rules**.

Discussion: ASP vs. Default Logic

- In **default logic**, defeasible rules are represented by defaults and strict rules are represented by first-order formulas.
- Comparing ASP and DL, default logic appears relatively simple, intuitive and straightforward to understand the principle of knowledge representation.
- **Normal ELPs** have the same sprit as default logic in this sense.

Discussion: Computational Cost

- Closure computation on strict rules introduces **additional costs**. In particular, transitivity would **exponentially increase** the number of strict rules in a program.
- In practice, the closure computation could be automatically done as a compilation process (**partial evaluation**). Alternatively, in runtime environments, the closure computation could be done only for the part of strict rules that are needed to solve a particular problem.