Vicious Circle Principle and Formation of Sets in ASP Based Languages

Michael Gelfond and Yuanlin Zhang

Oct, 2016

The Background: G. Cantor

The story starts with introduction of sets by G. Cantor:

"*A set is a Multiplicity (Many) that allows itself to be thought of as a Unity (One).*"

The efforts to better understand when a Multiplicity gives itself such a permission are still ongoing.

The Background: H. Poincare and B. Russel

One line of research aimed at answering this question is based on Vicious Circle Principle:

"No object or property may be introduced by a definition that depends on that object or property itself".

This, of course, reduces the problem to the definition of *dependency*.

The first attempt to formalize this notion is due to B. Russel.

One such recent definition, due to S. Feferman, was formalized in system W (named after H. Weyl).

It is known that some mathematical results cannot be carried out in this system.

Feferman's hypothesis: all of *scientifically applicable analysis* can be developed in the system W. (So far, all evidence is in its favor, 2004).

Sets in ASP

In ASP, sets appeared mainly as parameters of *aggregates* - functions on finite sets.
Non-recursive use of aggregates in ASP rules, e.g.

$$need\_TA(C) \leftarrow card\{X : enrolled(X, C)\} > 20$$

seem to have clear meaning.
But, despite the absence of infinity, the problem of self-reference reappears in the context of logic programs with recursion through aggregates.

There are substantial differences of opinion which have been a subject of research for a long time.

Clash of Intuitions

- **According to $\mathcal{F}log$ (Faber, Leone, Pfeifer, 2004), program**

$$p(1) \leftarrow p(0)$$
$$p(0) \leftarrow p(1)$$
$$p(1) \leftarrow card\{X : p(X)\} \neq 1$$

has answer set $\{p(0), p(1)\}$**.**

- **According to $\mathcal{S}log$ (Son and Pontelli, 2007) the program is inconsistent.**

The differences between semantics are normally analyzed in terms of means employed to establish correctness of *properties of aggregates*.

A new language, $\mathcal{A}log$ (Gelfond, Zhang, 2014), shifts attention from properties of aggregates to *existence of their parameters*.

A set expression $\{X : p(X)\}$ of $\mathcal{A}log$ denotes *the set of all objects believed by the rational agent associated with the program to satisfy property* p.

To avoid self-supportedness of beliefs, $\mathcal{A}log$ introduces a new form of VCP incorporated in its semantics.

VCP in $\mathcal{A}log$:

The reasoner's belief in $\mathrm{p(t)}$ *can not depend on existence of a set denoted by set expression* $\{\mathrm{X} : \mathrm{p(X)}\}$,

or, equivalently

$\{\mathrm{X} : \mathrm{p(X)}\}$ *denotes a set* S *only if for every* t *rational belief in* $\mathrm{p(t)}$ *can be established without a reference to* S *.*

Recursion through aggregates without VCP

The following is a simple example of a rule which allows recursion through aggregates but avoids vicious circles:

$$
\begin{aligned}
val(W,0) \quad \leftarrow \quad & gate(G, and), \\
& output(W, G), \\
& card\{W : val(W,0), input(W,G)\} > 0.
\end{aligned}
$$

Here $val(W,S)$ holds iff the digital signal on a wire W has value S.
The rule avoids vicious circle since one needs to only construct a particular subset of input wires of G. Since, due to absence of feedback in our circuit, W can not belong to the latter set our definition is reasonable.

$\mathcal{A}log$ and Other Languages

All the known semantics including that of $\mathcal{A}log$ coincide on programs stratified with respect to aggregates.

Moreover, an answer set of $\mathcal{A}log$ is also an answer set of $\mathcal{F}log$, $\mathcal{S}log$ and other languages.

In general, however, $\mathcal{A}log$ semantics is more restrictive and views more programs as inconsistent. This includes known examples such as Companies Control, etc.

Examples

- **Program** P_1:

$$p(1) \leftarrow p(0). \qquad p(0) \leftarrow p(1).$$

$$p(1) \leftarrow card\{X : p(X)\} \neq 1.$$

is inconsistent in $\mathcal{A}log$ **and** $\mathcal{S}log$.

- **Program** P_2:

$$p(1) \leftarrow card\{X : p(X)\} \geq 0.$$

is consistent in $\mathcal{F}log$ **and** $\mathcal{S}log$ **but inconsistent in** $\mathcal{A}log$.

- **Program** P_3:

$$p(1) \leftarrow card\{X : p(X)\} = Y, \; Y \geq 0.$$

which seems to express the same thought as P_2, **is inconsistent in all three languages.**

An argument in favor of consistency of

$$p(1) \leftarrow card\{X \ : \ p(X)\} \geq 0.$$

says that the premise is true for any set and hence the program is equivalent to

$$p(1).$$

But, this is true only if there is a set denoted by $\{X \ : \ p(X)\}$**, i.e. belief in** $p(1)$ **can be justified by** P_2 **only if one assumes existence of such set and hence existence of an answer set of the program.**

Answer Sets of $\mathcal{A}log$

Definition (Aggregate Reduct)

The *aggregate reduct* of a ground program Π of $\mathcal{A}log$ w.r.t. a set of ground regular literals S is obtained from Π by

1. removing from Π all rules containing aggregate atoms false in S.
2. replacing every remaining aggregate atom $f\{X : p(X)\} \odot n$ by the set $\{p(t) : p(t) \in S\}$

Definition (Answer Set)

A set S of ground regular literals over the signature of a ground program Π of $\mathcal{A}log$ is an *answer set* of Π if it is an answer set of an aggregate reduct of Π with respect to S.

An Example

Consider a program, consisting of a rule

$$p(a) \leftarrow card\{X : p(X)\} = 1.$$

It has two candidate answer sets, $S_1 = \{\ \}$ and $S_2 = \{p(a)\}$. The aggregate reduct of the program with respect to S_1 is the empty program. Hence, S_1 is an answer set of P_1. The program's aggregate reduct with respect to S_2 however is

$$p(a) \leftarrow p(a).$$

The answer set of this reduct is empty and hence S_1 is the only answer of P_1.

Beyond Original $\mathcal{A}log$

In a recent paper original $\mathcal{A}log$ was expanded to

- Allow aggregates on infinite sets.
- Expand the language by several other useful set related constructs.
- Prove some basic properties of the new language.

Set Atoms in the Bodies of Rules

Problem: ***given complete lists of required courses and courses taken by a student define when the student is ready to graduate.***

A natural definition of this relation

$$ready\_to\_graduate(S) \leftarrow \{C : required(C)\} \subseteq \{C : taken(S, C)\}.$$

$$\neg ready\_to\_graduate(S) \leftarrow \; \mathbf{not} \; ready\_to\_graduate(S).$$

contains a rule with set atoms and subset relation in the body.

We were not able to find a better way of defining $ready\_to\_graduate$.

How to mathematically define semantics of such rules?

A Difficulty:

Consider a program Π:

$$p(a) \leftarrow \{X : p(X)\} \subseteq \{X : q(X)\}.$$

$$q(a).$$

Intuitively, it has two possible candidate answer sets: $A_1 = \{q(a)\}$ **and** $A_2 = \{q(a), p(a)\}$.

A_1 **does not satisfy the first rule and, hence, is not an answer set. But what about** A_2**?**

If VCP is accepted then A_2 **is not an answer set.**

If only minimality is required then it is.

Definition (Set Reduct)

The *set reduct* of Π w.r.t. a set of ground regular literals A is obtained from Π by

1. removing rules containing set atoms which are *false* or *undefined* in A.
2. replacing every set name $\{X : p(X)\}$ by $\{p(t) : p(t) \in A\}$ and removing set atoms.

The set reduct of

$$q(a). \qquad p(a) \leftarrow \{X : p(X)\} \subseteq \{X : q(X)\}.$$

with respect to $A_2 = \{q(a), p(a)\}$ **is**

$$q(a). \qquad p(a) \leftarrow p(a), q(a).$$

$\mathcal{A}log$ expands original ASP by allowing:

- **rules with infinite heads and bodies,**
- **rules with set atoms.**

No change in semantics is needed to accomodate infinite rules. Set atoms are dealt with as follows:

Definition (Answer Set)

A set A of ground regular literals over the signature of a ground $\mathcal{A}log$ program Π is an *answer set* of Π if A is an answer set of the set reduct of Π with respect to A.

New $\mathcal{A}log$ uses rules with set atoms in the heads to express statements of the form: ***Let*** P ***be a subset (superset) of a set*** Q**.**

A program

$$q(a).$$

$$p \subseteq \{X : q(X)\}.$$

has answer sets $A_1 = \{q(a)\}$ **where the set** p **is empty and** $A_2 = \{q(a), p(a)\}$ **where** $p = \{a\}$**.**

The construct may be used in ways similar to the choice rule of Clingo but has simpler informal and formal semantics.

Discussion

This is a work on language design and, as such, has been guided by some general design principles, e.g.

- The constructs of a language should be close to those used in practice and have a simple syntax and a clear intuitive semantics based on understandable informal principles.
- A language should be *elaboration tolerant* i.e. it should be possible to add new constructs without substantial changes in its syntax and semantics.

We believe that, in this respect, $\mathcal{A}log$ is a success. Since our emphasis is on teaching some loss of expressive power is tolerable. It remains to be seen if more powerful extensions are needed in programming practice.

We made an attempt to expand $\mathcal{A}log$ by weakening VCP incorporated in its semantics, as follows:
Let C be an atom containing a set name $\{X : p(X)\}$. Then *belief in p(t) must be established without reference to the truth of C in A unless this truth can be demonstrated without reference to $p(t)$.*

To our surprise the resulting language turned out to be basically equivalent to $\mathcal{S}log$, and hence replacement of $f\{X : p((X)\} \geq 0$ by $f\{X : p((X)\} = Y, Y \geq 0$ does not preserve the program's semantics. So paradoxes persist.

Remaining Problems

I'll only mention two:

- **Check if a different form of VCP can lead to a richer language without paradoxes.**
- **A good language should have a type system, total and partial functions, and a decent implementation. Ingredients are available but integration and implementation is a non-trivial task.**

THANKS!

The relation can be defined without set atoms, i.e.

$$-ready\_to\_graduate(S) \leftarrow required(C),\ \textbf{not}\ taken(S, C).$$

$$ready\_to\_graduate(S) \leftarrow\ \textbf{not}\ \neg ready\_to\_graduate(S).$$

However, this program is more difficult to update. For instance, addition of rule

$$ready\_to\_graduate(S) :-special\_permission(S).$$

leads to inconsistency.

Additional difficulties appear when the list of classes taken by s is incomplete, or when system is dynamic and the rules can interfere with the inertia axiom.

Positive Ground Program Π

Normal form: All occurrences of $f\{X : p(X)\} = Y, cond(Y)$ **are replaced by** $cond(f\{X : p(X)\})$**.**

An ***instantiation*** **of condition** $C = \{X : p(X)\} \odot k$ **with respect to set** S **of ground literals is the set** $\{t : p(t) \in S\}$**.**

A subset-minimal set of ground atoms satisfying the rules of Π **is called a** ***candidate answer set*** **of** Π**.**

A **is an** ***answer set*** **of** Π **if**

- A **is a candidate answer set of** Π**, and**
- **for every** $p(t) \in A$, $p(t)$ **belongs to every? candidate answer set of the result of replacing** $C = \{X : p(X)\}$ **by their instantiation with respect to** $A \setminus \{p(t)\}$**. CHECK!**