Using answer set programming to answer complex queries

Chitta Baral
Dept. of Computer Sc. & Eng.
Arizona State University
Tempe, AZ 85287
chitta@asu.edu

Michael Gelfond
Dept. of Computer Sc.
Texas Tech University
Lubbock, TX 79409
mgelfond@cs.ttu.edu

Richard Scherl
Computer Science Dept.
Monmouth University
West Long Branch, NJ 07764
rscherl@monmouth.edu

Abstract

In this paper we discuss the applicability of the knowledge representation and reasoning language AnsProlog for the design and implementation of query answering systems. We consider a motivating example, and illustrate how AnsProlog can be used to represent defaults, causal relations, and other types of commonsense knowledge needed to properly answer non-trivial questions about the example's domain.

1 Introduction and Motivation

Let us envision a query answering system (QAS) consisting of a *search engine* which searches diverse sources for information relevant to the given query, Q; a *natural language processing module* (NLPM), which translates this information (including the query) into a theory, F, of some knowledge representation language $\mathcal{L}$; a *general knowledge base*, KB, containing common-senses and expert knowledge about various domains; and an *inference engine* which takes F and KB as an input and returns an answer to Q. Even though the choice of the KR language $\mathcal{L}$ is irrelevant for the first component of the system it plays an increasingly important role in the design of its other components. In this paper we hypothesize that AnsProlog - a language of logic programs under the answer set semantics (Gelfond and Lifschitz, 1988; Gelfond and Lifschitz, 1991) - is a good candidate for the KR language of QAS. This is especially true if answering a query Q requires sophisticated kinds of reasoning including default, causal, and counterfactual reasoning, reasoning about narratives, etc.

The list of attractive properties of AnsProlog include its simplicity and expressive power, ability to reason with incomplete information, existence of a well developed mathematical theory and programming methodology (Baral, 2003), and the availability of rather efficient reasoning systems such as SMODELS(Niemela and Simons, 1997) and others as well(Eiter et al., 1997; Yu and Maratea, 2004). AnsProlog allows its users to encode defaults, causal relations, inheritance hierarchies, and other types of knowledge not readily available in other KR languages. In addition it supports construction of elaboration tolerant knowledge bases, i.e., ability to accommodate new knowledge without doing large scale surgery. The main drawback of the language is the inability of its current inference engines to effectively deal with numbers and numerical computations[1].

In this paper we illustrate the use of AnsProlog for query answering via a simple example. Of course substantially more work is needed to to prove (or disprove) our main hypothesis.

Consider an analyst who would like to use his QAS to answer simple queries Q_1 and Q_2 about two people, John and Bob:

- Q_1 – Was John in the Middle East in mid-December?
- Q_2 – If so, did he meet Bob in the Middle East in mid-December?

Let us assume that the search engine of QAS extracted the following simple text relevant to Q_1 and Q_2:

[1]The current answer set solvers start their computation with grounding the program, i.e. replacing its variables by possible ground instantiations. The grounding algorithms are smart and capable of eliminating many useless rules; answer sets can be effectively computed even if the resulting program consists of hundreds of thousands of rules. However, if several integer variables are used by the program rules, the size of the grounded program becomes unmanageable. We hope however that this problem will be remedied by the development of new reasoning algorithms and systems.

John spent Dec 10 in Paris and took a plane to Baghdad the next morning. He was planning to meet Bob who was waiting for him there.

We will also assume that the NLP module of the QAS realizes that to answer our queries it needs general knowledge of geography, calendar, and human activities including travel, meetings, and plans. In the next section we outline how such knowledge can be represented in AnsProlog.

2 Representing general knowledge

2.1 The "geography" module M_1

The geography module M_1 will contain a list

```
is(baghdad,city).
is(iraq,country).
...
```

of places and the definition of relation

$in(P_1, P_2)$ - "P_1 is located in P_2"

The definition is given by a collection of facts:

```
in(baghdad, iraq).
in(iraq,middle_east).
in(paris,france).
in(france,western_europe).
in(western_europe,europe).
...
```

and the rule

```
in(P1,P3) :- in(P1,P2),
             in(P2,P3).
```

For simplicity we assume that our information about relation in is complete, i.e., if $in(p_1, p_2)$ is not known to be true then it is false. This statement can be expressed by the rule

```
-in(P1,P2) :- not in(P1,P2)
```

often referred to as the CWA- Closed World Assumption (Reiter, 1978) (for in). Here $-p$ stands for "p is false" while *not* p says that "there is no reason to belief p". Similar assumption can be written for is. The program has unique answer set containing $in(iraq, middle\_east)$, -$in(iraq, europe)$, etc. This answer set, (or its relevant parts) can be computed by answer set solvers. Sometimes this will require small additions to the program. For instance SMODELS, which require typing of variables, will not be able to compile this program. This problem can be remedied by adding a rule

```
position(P) :- is(P,C).
```

defining the type position and a statement

```
#domain position(P;P1;P2;P3)
```

declaring the type of the corresponding variables. Now SMODELS will be able complete the computation.

2.2 The "travelling" module M_2

This module describes the effects of a person travelling from one place to another. We are mainly interested in locations of people and in various travelling events which change these locations. Construction of M_2 is based on the theory of dynamic systems which views the world as a transition diagram whose states are labelled by fluents (propositions whose values depend on time) and arcs are labelled by actions. For instance, states of the diagram, D, can contain locations of different people; a transition $\langle \sigma_0, \{a_1, a_2\}, \sigma_1 \rangle \in D$ iff σ_1 is a possible state of the domain after the concurrent execution of actions a_1 and a_2 in σ_0. There is a well developed methodology of representing dynamic domain in AnsProlog (Baral and Gelfond, 2000; Turner, 1997) which, in its simplified form, will be used in the construction of M_2.

The language of M_2 will contain time-steps from $[0, n]$, fluent $loc(P, X, T)$ - "place P is a location of person X at step T". Various types of travelling events - fly, $drive$, etc., will be recorded by the list:

```
instance_of(fly,travel).
instance_of(drive,travel).
...
```

Description of an event type will contain the event's name and attributes. The following is a generic description of John flying to Baghdad.

```
event(a1).
type(a1,fly).
actor(a1,john).
destination(a1,baghdad).
```

An actual event of this type will be recorded by a statement

$occurs(a1, i)$.

(where i is a time-step in the history of the world) possibly accompanied by the actual time of i. In addition, M_2 will import relation $in(P_1, P_2)$ from the geography module M_1.

The transition diagram, D, of M_2 will be described by following groups of axioms.

•. The first group consists of *state constraints* establishing the relationship between the domain fluents. In our case it is sufficient to have the rules:

```
loc(P2,X,T)  :- loc(P1,X,T),
                in(P2,P1).
disjoint(P1,P2) :- -in(P1,P2),
                   -in(P2,P1),
                   neq(P1,P2).
-loc(P2,X,T) :- loc(P1,X,T),
                disjoint(P1,P2).
```

Here neq stands for the inequality. The first rule allows us to conclude that if at step T of the domain history X is in Iraq then he is also in the Middle East. The second two rules guarantee that X is not in Europe.

•. The second group contains *causal laws* describing direct effects of actions. For our example it suffices to have the rules

```
loc(P,X,T+1) :- occurs(E,T),
                type(E,travel),
                actor(E,X),
                destination(E,P),
                -interference(E,T).
-interference(E,T) :-
                not interfer-
ence(E,T).
```

The first rule says that, in the absence of interference, a traveller will arrive at his destination. The second - the CWA for $interference$ - states that the interference is an unusual event which normally does not happen.

•. The third group consists of executability conditions for actions, which have the form

```
-occurs(E,T) :- cond(T).
```

which says that it is impossible for an event E occur at time step T if at that time step the domain is in a state satisfying condition *cond*.

Causal laws and state constraints determine changes caused by execution of an action. To complete the definition of the transition diagram of the domain we need to specify what fluents do not change as the results of actions. This is a famous Frame Problem from (McCarthy and Hayes, 1969) where the authors suggested to solve it by formalizing the Inertia Axiom which says that "things tend to stay as they are". This is a typical default which can be easily represented in AnsProlog. In our particular case it will have a form:

```
loc(P,X,T+1)  :- loc(P,X,T),
                 not -loc(P,X,T+1).
-loc(P,X,T+1) :- -loc(P,X,T),
                 not loc(P,X,T+1).
```

The above representation is a slightly simplified version of AnsProlog theory of dynamic domains which gives notation for causal relations of the domain, includes general (fluent independent) formulation of the inertia, explains how the set of causal relations define the corresponding transition diagram, etc. We used this version to simple save space. Given the following history of the domain

```
loc(paris,john,0).
loc(baghdad,bob,0).
occurs(a1,0).
```

information contained in M_1 and M_2 is sufficient to conclude $loc(baghdad, john, 1)$, $loc(baghdad, bob, 1)$, $loc(middle\_east, john, 1)$, -$loc(paris, john, 1)$, etc. To answer the original queries we now need to deal with timing our actions. Let us assume, for instance, that the timing of John's departure from Paris is recorded by statements:

```
time(0,day,11).
time(0,month,12).
time(0,year,03).
```

Here day, $month$, and $year$ are the *basic time measuring units*.

Finally we may need to specify typical durations of actions, e.g.

```
time(T+1,day,D) :- occurs(E,T),
                   type(E,fly),
                   time(T,day,D),
                   not -
time(T+1,day,D).
```

where $1 \leq D \leq 31$.

To reason about the $time$ relation we need to include a new module, M_3, which will allow us to change granularity of our time measure.

2.3 M_3 - measuring time

The module contains types for basic measuring units, e.g.

```
day(1..31).
month(1..12).
part(start).
part(end).
part(middle).
...
```

and rules translating from one granularity measure to another, e.g.

```
time(T,part,middle)  :- time(T,d,D),
                        10 < D < 20.
time(T,season,summer):-
time(T,month,M),
                          5 < M < 9.
...
```

M_3 presented in this paper is deliberately short. It includes very little knowledge beyond that needed to answer our query. Ideally it should be much bigger and include a formalization of the calendar. Among other things the module should allow us to prove statements like $next(date(10, 12, 03), date(11, 12, 03)$ and $next(date(31, 12, 03), date(1, 1, 04)$.

Now let us assume that NLP module of our QAS translated

(a) information about John's flight to Baghdad by a history

```
loc(paris,john,0).
loc(baghdad,bob,0).
occurs(a1,0).
time(0,day,11).
time(0,month,12).
```

(b) the query Q_1 by

```
? loc(middle_east,john,T),
  time(T,month,12),
  time(T,part,middle).
```

Modules M_1, M_2 and M_3 have enough information to correctly answer Q_1.

2.4 Planning the meeting - M_4

To answer the second question we need an additional module about the event meet. The event type for $meet$ will be similar to the previously discussed flying event $a1$. It may look like:

```
event(a2).
type(a2,meet).
actor(a2,john).
actor(a2,bob).
place(a2,baghdad).
```

Notice however that the story contains no information about actual occurrence of this event. All we know is that $a2$ is *planned* to occur at time step one. We encode this by simply stating:

```
planned(a2,1).
```

Note that to give a positive answer to the question Q_2 –"Did John meet Bob in the Middle East in mid-December? " we need to reason about planned events. It seems that our positive answer to this question is obtain by using a default: "*people normally follow their plans*". Again this is a typical default statement which, according to the general knowledge representation methodology of AnsProlog could be expressed by the rule:

```
occurs(E,T) :- planned(E,T),
               not -occurs(E).
```

In a slightly more complex situation we may need to assume that people take their plans seriously – they persist with their plans until the planned event actually happen. This is encoded as follows:

```
planned(E,T+1) :- planned(E,T),
                  -occurs(E,T).
```

Unlike traveling, the meeting event does not seem to have any obvious causal effects. It, however, has the following executability condition relevant to our story.

```
-occurs(E,T) :- type(E,meet),
                actor(E,X),
                place(E,P),
                -loc(P,X,T).
```

Now we have enough information to answer our second query, which can be encoded as

```
? occurs(E,T),
  type(E,meet),
  actor(E,john),
  actor(E,bob),
  loc(middle_east,john,T),
  time(T,month,12),
  time(T,part,middle).
```

As expected the answer will be positive. There are several ways to obtain this answer. It can of course be extracted from the unique answer set of our program. With small additions of types and declaration of variables similar to that we used to define $position$ in M_1 this answer set can be found by SMODELS or any other answer set solver. This method however may not scale. The problem is caused the calendar. Its integer variables for months, days, etc, in conjunction with a longer history (and therefore a larger number of time steps) may cause an unmanageable increase in the number of ground rules of the program. It seems however that in many interesting cases (including ours), the computation can be made substantially more efficient by properly combining answer set finding algorithms with the traditional resolution of Prolog. The way of doing this will be illustrated in the full paper. We also plan to expand our modules especially those dealing with time and reasoning about plans.

3 FrameNet and Events

Our vision of the NLPM is that it will translate both our short text and also the queries into AnsProlog sentences. There is a body of literature on translating or parsing English sentences into a semantic representation such as First Order Logic. See (Blackburn and Bos, 2003) for a recent survey of such techniques. The semantic representation makes use of symbols based upon the lexicon of English.

The success of our endeavor requires that there be an axiomatization of the relationship between the symbols representing functions and predicate symbols in our various AnsProlog theories (e.g., M1 – M4) and the symbols (based upon the lexicon of English) used in the semantic representation of the English queries and the narrative texts. The online lexical database, FrameNet(Baker et al., 1998) provides such a connection, especially for events. This is done through the notion of frame semantics that underlies FrameNet.

Frame semantics assumes that lexical items draw their meaning from conceptual structures or *frames* that provide an abstract or scematic description of particular types of events. The frames are structured into an inheritance hierarchy. Each frame includes a number of *frame elements* (FEs) or roles that make up the conceptual structure.

For example, our "travelling" module M_2 closely corresponds to the related FrameNet frames **Travel**, **Move**, and **Ride_Vehicle**. The frames relate the various frame elements of Area (where the travelling takes place), Goal (where the travellers end up), Path (route of the travel), Source (starting point of the trip), and the Traveller (the living being which travels).

Consider the phrase *took a plane* used to express the travelling activity. The verb *take* is associated with the frame **Ride_Vehicle**. This information allows the connection with the axiomatization of flying events in M_2. On the other hand FrameNet does not have entries for the verb *spend* as in *spent Dec 10*. But WordNet(Fellbaum, 1998) has 3 senses for the verb *spend*. Sense 1 is "pass – (pass (time) in a specific way. 'How are you spending your summer vacation?'). " Unfortunately, neither *pass* nor *time* allows us to index a useful frame for just being in a place. The coverage of FrameNet is not sufficient. It will be necessary to augment our use of FrameNet with other online sources such as WordNet and to also increase the number of frames within FrameNet.

There has been some related work on using the frame of FrameNet for reasoning (Chang et al., 2002) and also on the automatic annotation of English texts with regard to the relevant frames (Gildea and Jurafsky, 2000) and frame elements.

4 Syntax and Semantics of AnsProlog

An AnsProlog knowledge base consists of rules of the form:

$$l_0 \leftarrow l_1, \ldots, l_m, \textit{not } l_{m+1}, \ldots, \textit{not } l_n \qquad (4.1)$$

where each of the l_is is a literal, i.e. an atom, a, or its classical negation, -a and *not* is a logical connective called *negation as failure* or *default negation*. While -a states that a is false, an expression *not* l says that there is no reason to believe in l.

The answer set semantics of a logic program Π assigns to Π a collection of *answer sets* – consistent sets of ground literals corresponding to beliefs which can be built by a rational reasoner on the basis of rules of Π. In the construction of these beliefs the reasoner is guided by the following informal principles:

- He should satisfy the rules of Π, understood as constraints of the form: *If one believes in the body of a rule one must belief in its head.*
- He should adhere to the *rationality principle* which says that *one shall not believe anything he is not forced to believe*.

The precise definition of answer sets is first given for programs whose rules do not contain default negation. Let Π be such a program and X a consistent set of ground literals. Set X is *closed* under Π if, for every rule (4.1) of Π, $l_0 \in X$ whenever for every $1 \leq i \leq m$, $l_i \in X$ and for every $m+1 \leq j \leq n$, $l_j \notin X$.

Definition 1 *(Answer set – part one)*
A state X of $\sigma(\Pi)$ is an *answer set* for Π if X is minimal (in the sense of set-theoretic inclusion) among the sets closed under Π.

To extend this definition to arbitrary programs, take any program Π, and consistent set X of ground literals. The *reduct*, Π^X, of Π relative to X is the set of rules

$$l_0 \leftarrow l_1, \ldots, l_m$$

for all rules (4.1) in Π such that $l_{m+1}, \ldots, l_n \notin X$. Thus Π^X is a program without default negation.

Definition 2 *(Answer set – part two)*
X is an answer set for Π if X is an answer set for Π^X.

Definition 3 *(Entailment)*
A program Π entails a literal l ($\Pi \models l$) if l belongs to all answer sets of Π.
The Π's answer to a query l is *yes* if $\Pi \models l$, *no* if $\Pi \models \bar{l}$, and *unknown* otherwise.

5 Summary

In conclusion, we feel that the features of AnsProlog are well suited to form the foundations for an inference engine supporting a QAS. Our future work will develop the support tools and implementation needed to demonstrate this hypothesis.

References

C. Baker, C. Fillmore, and J. Lowe. The Berkeley FrameNet project. In *Proceedings of the COLING-ACL, Montreal, Canada*, 1998.

C. Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003.

C. Baral and M. Gelfond. Reasoning agents in dynamic domains. In J Minker, editor, *Logic Based AI*. pp. 257–279, Kluwer, 2000.

P. Blackburn and J. Bos. Computational Semantics. *Theoria*, 18(1), pages 365–387, 2003.

N. Chang, S. Narayanan, R. Miriam, and L. Petruck. From frames to inference. In *Proceedings of the First International Workshop on Scalable Natural Language Understanding, Heidelberg, Germany*, 2002.

C. Fellbaum (ed). *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Logic Programming: Proc. of the Fifth Int'l Conf. and Symp.*, pages 1070–1080. MIT Press, 1988.

M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, pages 365–387, 1991.

D. Gildea and D. Jurafsky. Automatic Labeling of Semantic Roles. In *Proceedings of ACL 2000, Hong Kong, China*, 2000.

Lierler Yu., and Maratea M. Cmodels-2: SAT-based Answer Sets Solver Enhanced to Non-tight Programs, In *Proc. of LPNMR-7*, pp. 346, 2004.

J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh, 1969.

T. Eiter, N. Leone, C. Mateis., G. Pfeifer and F. Scarcello. A deductive system for nonmonotonic reasoning, *Proceedings of the 4rd Logic Programming and Non-Monotonic Reasoning Conference – LPNMR '97*, LNAI 1265, Springer-Verlag, Dagstuhl, Germania, Luglio 1997, pp. 363–374.

I. Niemela and P. Simons. Smodels – an implementation of the stable model and well-founded semantics for normal logic programs. In *Proc. 4th international conference on Logic programming and non-monotonic reasoning*, pages 420–429, 1997.

R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 119–140. Plenum Press, New York, 1978.

H. Turner. Representing actions in logic programs and default theories. *Journal of Logic Programming*, 31(1-3):245–298, May 1997.