Textual Inference by Combining Multiple Logic Programming Paradigms

Chitta Baral
Dept. of C.S. & Eng.
Arizona State University
Tempe, Az 85287
chitta@asu.edu

Gregory Gelfond
Dept. of C.S.
Texas Tech University
Lubbock, Tx 7409
gelfond@redwood.cs.ttu.edu

Michael Gelfond
Dept. of C.S.
Texas Tech University
Lubbock, Tx 7409
mgelfond@redwood.cs.ttu.edu

Richard B. Scherl
C.S. Dept.
Monmouth University
West Long Branch, N.J.
rscherl@monmouth.edu

Abstract

The goal of our approach to textual inference is to answer queries about events and dates reported in texts; queries that demand inferencing with the relevant background knowledge. Our primary knowledge representation language is AnsProlog. The core inference engine is a combination of AnsProlog and Constraint Logic programming. AnsProlog is particularly useful for the representation of defaults, causal relations, and other types of common-sense knowledge. Constraint Logic Programming is needed to solve constraints involving the relationship between the sequence of actions and the dates on which the actions may have occurred. The combination of the two creates a useful tool for answering quries over texts and in particular queries that involve relatively deep reasoning.

Introduction and Motivation

We envision a query answering system which is given a text of some sort. This text may be typed into the system by the user or it may have been retrieved by a search engine looking for sources of information relevant to a query.

After the text is processed by the system, the user may then ask particular queries pertaining to the information in the text. Often the text will describe a series of actions or events and the queries that the user wishes to pose will concern the truth of various facts about the world after these events have taken place or while they are occurring. Answering those questions demands commonsense reasoning and the use of commonsense knowledge about the world. Generally, this commonsense knowledge is not contained in the actual text.

The approach is to initially translate the natural language text (which we assume is in English) and the user's query (also in English) into an appropriate knowledge representation language. A knowledge base, also in the same knowledge representation language, is available. The knowledge base needs to contain both commonsense and perhaps some expert knowledge about the relevant domains. Then the inference engine answers the user's query by reasoning with both the representation of the information contained in the text and the background knowledge base. Finally, an answer is returned to the user.

Copyright © 2005, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Here we propose a knowledge representation language and an inference engine suitable for such a query answering system. As the core knowledge representation language, we propose AnsProlog - a language of logic programs under the answer set semantics (Gelfond & Lifschitz 1988; 1991). This language is especially useful if answering the query requires sophisticated kinds of reasoning such as default, causal, 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 & Simons 1997b) and others as well(Niemela & Simons 1997a; Lierler & 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. There is also a well developed methodology for representing dynamic domains (Baral & Gelfond 2000; Turner 1997). In addition, it supports the 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. This is because 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.

Here we are concentrating on reasoning about texts that involve actions and time. We not only have to reason about the sequence of steps (individual actions) and what facts are true after each step, but we also need to reason about dates and times. Since the number of dates is relatively large, the use of several rules that contain date variables quickly becomes prohibitively inefficient. The same problem that occurs in general with numerical computations occurs in particular with reasoning about dates and times.

We propose an architecture that combines the use of AnsProlog with (CLP) Constraint Logic Programming(Jaffar & Lassez 1987; Van Hentenryk 1989; Marriott & Stuckey 1998; Frühwirth & Abdennadher 2002). CLP allows the use of efficient solvers over large and possibly infinite domains while still preserving the declarative properties of logic programming. The CLP portion determines which action steps correspond to which dates.

Our implementation makes use of this combination of AnsProlog and CLP. We have used the SMODELS implementation of AnsProlog and the CLP module contained in Sicstus[1] Prolog. We have also built a background theory in AnsProlog sufficient to reason about a travel domain. This paper describes the architecture of our reasoning system and the background travel domain. It does not cover the translation of natural language text and queries into the AnsProlog, nor does it cover the translation of the AnsProlog output into natural language. These are the topics of our current and future work.

Examples

Here is a simple example (Scenario 1) to illustrate the problem with which we are concerned.

> John is in Paris. On March 15th he packs his laptop in the carry-on luggage and takes a plane to Baghdad. Was his laptop in Baghdad on March 16th?

Depending on the time of the day, that John leaves Paris, he may arrive in Baghdad on either March 15th or March 16th. But it is a reasonable default assumption that in the absence of additional information John and his laptop remained in Baghdad after arriving. So, the desired answer is yes.

But consider a variation on the above, Scenario 2.

> On March 15th John took the plane from Paris to Baghdad. On the way the plane stopped in Rome, where John was arrested. Is John in Baghdad on March 16th?

Here we want to say no, because the arrest action would generally prevent John from continuing on to Baghdad. The location of the laptop is more complex, but a good answer is no. Of course to obtain this answer (rather than unknown), it is necessary to represent the background rule (a defeasible rule) that when someone is arrested on a plane, the person's carry-on luggage is taken off.

We now introduce, AnsProlog and show how these examples can be represented in that language.

Syntax and Semantics of AnsProlog

An AnsProlog knowledge base consists of rules of the form:

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

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.

[1] http://www.sics.se/sicstus

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 (*) 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 (*) 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.

Given, the translation of a text and a background theory, the initial task of inferencing in AnsProlog is to compute all of the answer sets or models of the text and background theory. To determine whether a fact follows from our text and background theory, it is necessary to have a definition of entailment.

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.

Given an AnsProlog program Π, the output of an implementation of AnsProlog such as SMODELS is a set of the answer sets of the program Π. Each of the answer sets is represented by a listing of the ground literals true in that answer set. The output can be quite large, but the user can restrict the output to see the specific results that are relevant to his/her purposes.

Note that the language of AnsProlog includes both negation as failure (not), and logical negation (-). This is important as the two together are used to represent defaults. Many examples of this combination occur in the next sections.

Representing Travel Events

It is necessary to represent in AnsProlog the information contained in the scenarios as well as the needed commonsense background information about people, places, traveling, and the effects of traveling from one place to another. Since facts about the world will be true at some points and false in others, a mechanism to distinguish between points in time is needed. Dates can not be used here as we would then end up with AnsProlog atoms containing variables with very large domains. This is not feasible given the grounding mechanism of AnsProlog inference systems such as SMODELS. The reasoning about dates is reserved for the CSP program.

We use integers to represent step points between the basic actions that can be performed. A limit does need to be set on the number of step points needed. For the examples discussed in this paper, the limit was set to 5, but for more complicated examples a higher limit would be needed.

A basic object of this module is a *trip*, i.e., a short journey over a set route. We also need *locations*. The clauses `trip(J)` and `location(L)` are used to specify that a term is the name of a trip or a location. The name can be anything, but generally, we use a term constructed out of the starting city and the destination, e.g., `j(boston,paris)`. We also need the special position `en_route` to indicate that a trip is in transit or between stops.

A description of a trip's planned route is then given simply by a list of atoms such as $\mathtt{stop}(\mathtt{j},\mathtt{l}_0,0), ..., \mathtt{stop}(\mathtt{j},\mathtt{l}_k,\mathtt{k})$ which specifies the stops of the trip `j` at location $\mathtt{l}_\mathtt{i}$. Intermediate stops may be unspecified, but we do require the specification of the location of the origin ($\mathtt{l}_0$) and the destination ($\mathtt{l}_\mathtt{k}$).

We view a trip as an object that is "performing" actions. There are two main actions a trip can perform. One is departing from its current location `depart(j)` and the other is stopping at a particular location `stop(j,L)`.

Properties of the world that change over time (as actions occur) are called *fluents*. For example, the fluent `at` as in `at(J,Pos)` represents that a particular trip `J` is at a particular position `Pos`. Positions include both locations (i.e, cities) and the special position `en_route`. Therefore our theory of trips now contains several types; actions, fluents, steps, locations, trips and also people, luggage, travel documents, and modes of transportation. Our Smodels implementation declares typed variables with all of these types.

We use the relations `h` (for `holds`) and `o` (for `occurs`) as the basis for our action theory. The first argument of `h` is a fluent and the second is a step point. For example, `h(at(john,en_route),3)` asserts that at step 3, John is at position `en_route`. Similarly, the first argument of `o` is an action term and the second is a step point. So, `o(arrest(john),3)` asserts that at step 3 John was arrested.

Background Information

A background module using the conventions developed above is needed to represent information about the effect that various actions in this travel domain have on the truth of the fluents as well as the default behavior of trips. The following are a number of our dynamic causal laws that represent the inferences (but defeasible inferences) that may be drawn in this domain. We must omit many of the details here, but the examples below should give an idea of the capability of AnsProlog to represent our commonsense knowledge of the travel domain.

After the departure, trip `J` is `en_route`:

```
h(at(J,en_route),S+1) :- o(depart(J),S).
```

Normally after stops, trips continue on to their next stop, unless the destination has been reached.

```
o(stop(J,C),T) :-
  h(at(J,en_route),T),
  next_stop(J,C),
  not -o(stop(J,C),T).
o(depart(J),T+1) :-
  o(stop(J,C),T),
  not dest(J,C),
  not -o(depart(J),T+1).
```

The position of an object is unique:

```
-h(at(O,Pos1),S) :-
  h(at(O,Pos2),S),
  neq(Pos1,Pos2).
```

The following axioms specify inertia, that things tend to stay as they are (unless of course something occurs to change them).

```
h(F1,S+1) :- h(F1,S),
      not -h(F1,S+1).
-h(F1,S+1) :-  -h(F1,S),
      not h(F1,S+1).
```

These axioms allow us to solve the frame problem(McCarthy & Hayes 1969).

We can also make some typical default assumptions about travelers. For example, unless otherwise noted, the traveler already has his/her passport and luggage.

```
h(has(P,passport(P)),0) :-
  not -h(has(P,passport(P)),0).
h(has(P,Luggage),0) :-
  owns(P,Luggage),
  not -h(has(P,Luggage),0).
```

When someone, a person `P` embarks on a trip he/she becomes a participant.

```
h(participant(P,J),T+1) :-
     o(embark(P,J),T).
```

The status of being a participant persists by default. It ends when the trip ends:

```
-h(participant(P,J),T+1) :-
     o(disembark(P,J),T).
```

Also, it may be terminated if certain actions occur.

```
-h(participant(P,F),T+1) :-
     o(arrest(P),T).
```

Being arrested during the course of the trip is one such event.

The status of being a participant is important since participants share the location of the trip.

```
h(at(P,D),T) :-
   h(participant(P,J),T), h(at(J,D),T).
```

Personal Possessions are attached to a person and objects in containers share the location with the container.

```
h(at(PP,D),T) :-
  h(has(P,PP),T), h(at(P,D),T).
h(has(P,PP),T) :-
   h(inside(PP,Container),T),
   h(has(P,Container),T).
```

Representation of the Examples

Scenario 1 is translated into AnsProlog as follows: The translation of *John is in Paris* is

```
h(at(john,paris),0).
```

It simply states that the fluent `at(john,paris)` holds at step 0. The sentence *On March 15th he packs his laptop in the carry-on luggage and takes a plane to Baghdad* is represented by the following three AnsProlog clauses.

```
o(pack(john,laptop(john),
           carry_on(john)),0).
h(trip_by(j(paris,baghdad),plane),1).
o(go_on(john,j(paris,baghdad)),1).
```

The first clause states that the trip named `j(paris,baghdad)` begins at step point 1 and is by plane. The second clause states that john goes on that trip at step point 1.

For the time being we are ignoring the information about dates in the text. This information will be incorporated in the next section.

When we take the translation of Scenario 1, along with the background travel theory, the output from an AnsProlog inference engine such as Smodels is a list of all of the true ground literals in the single answer set of the program (translation of the scenario and the background travel theory). These will include the following:

```
         .
         .
o(disembark(john,j(paris,baghdad)),4)
o(stop(j(paris,baghdad),baghdad),3)
h(at(laptop(john),baghdad),5)
-o(stop(j(paris,baghdad),baghdad),5)
         .
         .
```

Although the full listing of the answer set is very large it does contain the literal `h(at(laptop(john),baghdad),5)` which indicates that John's laptop is in Baghdad at step 5. Since in this case we only have one answer set, the answer to our query *Was his laptop in Baghdad on March 16th?* is true as long as step 5 can be identified with the correct date.

Constraints about Dates and Time Points

It is now necessary to represent the information that we do have about the correspondence of step points with dates. This information is represented as Prolog clauses as it will be used by the CLP program to be discussed next. For example, we may know the day and month of step point 0. This can be expressed with `time(0,d,15)` and `time(0,m,3)`. These clauses state that the time of step point 0 in units `d` (day) is 15 and in units `m` (month) is 3 (i.e., March). When we have a full date (including the year), we can convert the full date from our calendar (Gregorian calendar) into a fixed day number (Rata Die) as described by (Reingold & Dershowitz 2001). This conversion simplifies reasoning about leap years and the number of days in months. Reasoning about dates (year, month, day) reduces to reasoning about days. Additionally, other calendrical systems can also be converted into fixed day numbers. In this case March 15, 2004 would be indicated as `time(0,rd,731655)`. We have a small Prolog program that allows us to convert in both directions between Gregorian dates and fixed day numbers.

Additionally, it is necessary to represent information about event duration. For example.

```
duration(Action,d,0) :-
   one_day_action(Action).
one_day_action(pack(_,_,_)).
one_day_action(embark(P,J)) :-
   person(P), trip(J).
duration_interval(get(P,passport(P)),
                             d,3,21).
```

The CLP program has to assign dates (e.g. days and months) to step points while still satisfying constraints involving the above information. We have written a CLP program incorporating the following constraints:

Constraint 1 The assignment must respect the assertion of facts of the form *time(I,Unit,X)* as in `time(0,d,15)`.

Constraint 2 The assignment must respect the assertion of facts of the form *time_interval(I, Unit, First, Last)*. These facts express the constraint that a particular step I must occur within the open interval *First* . . . *Last*, measutred in unit *Unit*.

Constraint 3 The assignment must respect facts of the form *duration(Action, Unit, D)* which assert that action *Action* takes *D* duration when measured in *Unit* units.

Constraint 4 The assignment must respect all assertions of the form *duration_interval(Action, Unit, Min, Max)*.

Constraint 5 If step I $<$ step J, then time I $\leq$ time J.

The output of the CLP module is an assignment of dates to time steps. It is possible that there are multiple legal assignments.

Putting the Pieces Together

The final step is to pose the query to the query module (written in Prolog). The output of SMODELS is transformed into a set of Prolog clauses that can be simply loaded into Prolog for access by the query module as well as the CSP module. Finally, the query module matches the date to the time point and then queries the answer sets.

Examples Completed

It is necessary to add the information about dates and times for the two scenarios. This information is needed by the CLP program and the constraints listed above must take it into account. Part of the translation of the text is the fact that step points 0 and 1 both correspond to March 15. If we include the year, March 15, 2004 converts to fixed day number 731655. We need to add `time(0,d,15)` and `time(0,m,3)` or `time(0,rd,731655)`. It is also necessary to add `duration(j(paris,baghdad),d,1)` to indicate that traveling from Paris to Baghdad takes 1 day. (A more accurate axiomatization would represent the duration in time and allow the possibility that a traveler would leave earlier in the day and arrive on the same day.) In Scenario 1, step 5 is matched to March 15th. Therefore the answer to our query is *yes*, since in the only answer set for Scenario 1, the laptop is in Baghdad at step 5. Note that if the fixed day numbers are used, step 5 would match to 731656 and this day number converts to March 16th 2005 in the Gregorian calendar.

For Scenario 2, a variation on Scenario 1, the translation is as follows:

```
h(at(john,paris),0).
h(trip_by(j(paris,baghdad),plane),1).
o(go_on(john,j(paris,baghdad)),1).
```

The translation of *On the way the plane stopped in Rome, where John was arrested.* is translated into the following two clauses.

```
o(stop(j(paris,baghdad),rome),2)
o(arrest(john),3)
```

Note that the beginning of the translation of Scenario 2, is effectively identical to that for Scenario 1. But the answer to the query is *no*, because John is not in Baghdad at any step in the scenario. This is because the trip has been interrupted. A defeasible conclusion drawn in Scenario 1, has been defeated in Scenario 2. The labeling of the step points here is not significant and works essentially in the same fashion as in Scenario 1.

In general, *unknown* or *possible* may also be an answer when there are models or labelings of step points in which the query is true and also in which the query is false. We have also extened the approach to work with time (i.e., hours, minutes and seconds) in addition to dates.

Summary

In conclusion, we feel that the combination of AnsProlog and CLP provides a powerful representation language and inference engine for domains involving reasoning about actions and time. Our implementation combines Prolog, SMODELS (for AnsProlog inferencing) and the CSP library of Sicstus Prolog (for reasoning over the domain of dates). The two scenarios used to illustrate the method could not have been handled by either of the methods alone.

The following are extensions to this work which form the subject of our current and future research.

- Automate the translation of English to the AnsProlog representation for both texts and queries.
- Extend the approach to handle queries involving temporal relations (e.g. precedence, containment) of actions and of periods when particular fluents are true.

Acknowledgements

This work has been supported by ARDA.

References

Baral, C., and Gelfond, M. 2000. Reasoning agents in dynamic domains. In Minker, J., ed., *Logic Based AI*. Kluwer.

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

Frühwirth, T., and Abdennadher, S. 2002. *Essentials of Constraint Programming*. Berlin, Germany: Springer.

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

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9:365–387.

Jaffar, J., and Lassez, J.-L. 1987. Constraint logic programming. In *Proceedings of the 14th ACM Principles of Programming Languages Conference*, 111–119.

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

Marriott, K., and Stuckey, P. J. 1998. *Programming with Constraints*. Cambridge, Massachusetts: The MIT Press.

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

Niemela, I., and Simons, P. 1997a. A deductive system for nonmonotonic reasoning. In *Proceedings of the 4th Logic Programming and Non-Monotonic Reasoning Conference – LPNMR '97, LNAI 1265*, 363–374. Dagstuhl, Germany: Springer-Verlag.

Niemela, I., and Simons, P. 1997b. 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*, 420–429.

Reingold, E., and Dershowitz, N. 2001. *Calendrical Calculations: The Millennium Edition*. Cambridge University Press.

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

Van Hentenryk, P. 1989. *Constraint Satisfaction in Logic Programming*. Cambridge, MA: MIT Press.