Working with Defaults in a Controlled Natural Language

Rolf Schwitter
Department of Computing
Macquarie University
Sydney NSW 2109, Australia
Rolf.Schwitter@mq.edu.au

Abstract

In this paper, we discuss how statements about defaults and various forms of exceptions to them can be incorporated into an existing controlled natural language. We show how these defaults and exceptions are translated and represented in the answer set programming paradigm in order to support automated reasoning.

1 Introduction

Defaults are statements in natural language that contain words such as *generally*, *normally*, or *typically* and generalise over what a particular kind of objects does. These kinds of statements are very useful in human communication, since we often do not have complete information about the world, but must be able to draw conclusions based on incomplete information. These conclusions are preliminary, and we may be forced to withdraw them later when new information becomes available. In this paper, we investigate how statements about defaults and exceptions to them can be incorporated into an existing controlled natural language (White and Schwitter, 2009) and what kind of formal machinery is required to process these defaults and to reason with them in the answer set programming paradigm (Gelfond and Lifschitz, 1988; Lifschitz, 2008; Gebser et al., 2012). Answer set programming (ASP) has its roots in logic programming and non-monotonic reasoning and is well suited for solving problems which involve commonsense reasoning (Eiter et al., 2009).

It is important to note that we are working here with a controlled natural language (for a survey see (Kuhn, 2013)). Our controlled natural language (CNL) consists of a well defined subset of English and has been designed to serve as a knowledge representation language with automated reasoning support (White and Schwitter, 2009). The CNL allows domain specialists to write a textual specification using the vocabulary of the application domain. The writing process of the CNL is guided by an intelligent authoring tool, and there is no need for the human author to formally encode the knowledge since the language processor takes care of this process.

2 CNL and Answer Set Programming

Our CNL processor translates a specification written in CNL with the help of a discourse representation structure (DRS) (Schwitter, 2012) in the spirit of (Kamp and Reyle, 1993; van Eijck and Kamp, 2011) into an executable ASP program.

An ASP program looks similar to a Prolog program but relies on a completely different computational mechanism. Instead of deriving a solution from a program specification using resolution like Prolog does, finding a solution in the ASP paradigm corresponds to computing one or more stable models (Gelfond and Lifschitz, 1988) that in principle always terminate. Stable models are also known as answer sets (Lifschitz, 2008).

The building blocks of an ASP program are atomic formulas (atoms), literals and rules. A rule is an expression of the following form:

$$L_0 \textit{ or } ... \textit{ or } L_k \leftarrow L_{k+1}, ..., L_m, \textit{ not } L_{m+1}, ..., \textit{ not } L_n.$$

where L_i's are literals. A literal is either an atom a or its classical negation $\neg a$. The symbol *not* stands for negation as failure; *not* L_i means that L_i is not known. The symbol $\leftarrow$ stands for an implication. The expression on the left-hand side of this symbol is called the *head* of the rule and may consist of a disjunction (*or*) of literals. The expression on the right-hand side is called the *body* of the rule. If the body of a rule is empty, then the rule is called a *fact*, and if the head of a rule is empty, then the rule is called a *constraint*. Constraints are not important in the following discussion, but they can be expressed in our CNL (Schwitter, 2012).

Rolf Schwitter. 2013. Working with Defaults in a Controlled Natural Language. In *Proceedings of Australasian Language Technology Association Workshop*, pages 106−110.

Our CNL processor takes, for example, the following text as input:

1. Sam is a child.
2. John is the father of Sam and Alice is the mother of Sam.
3. Every father of a child is a parent of the child.
4. Every mother of a child is a parent of the child.
5. Every parent of a child cares about the child.

and translates it via a DRS into an ASP program. In our case, the resulting ASP program is a positive logic program (without any form of negation or disjunction) and consists of a set of facts and rules:

```
child(sam).
father(john,sam).
mother(alice,sam).
parent(X,Y) :- father(X,Y), child(Y).
parent(X,Y) :- mother(X,Y), child(Y).
care(X,Y)   :- parent(X,Y), child(Y).
```

This program derives the following unique answer set with the help of an answer set solver:

```
{ child(sam) father(john,sam)
  mother(alice,sam) parent(john,sam)
  parent(alice,sam) care(alice,sam)
  care(john,sam) }
```

It contains – among other literals – the two literals `care(alice,sam)` and `care(john,sam)`.

3 Extending the CNL with Defaults

Now, let's assume that we learn the subsequent new information via the CNL sentence (6) and (7):

6. John does not care about Sam.
7. Alice is absent.

In everyday human reasoning this new information does in general not cause problems, since humans seem to able to revise their beliefs with ease. However, the addition of the formal representation `-care(john,sam)` derived from sentence (6) to the above-mentioned ASP program results in an inconsistent answer set. And the addition of the formal representation `absent(alice)` derived from sentence (7) does not have any impact on the conclusion `care(alice,sam)` (humans might have a least some doubts here).

In order to deal with this situation, we have to replace sentence (5) that results in a strict rule by a sentence such as (5') that expresses a default using the keyword *normally*[1] and builds the starting point for non-monotonic reasoning:

5'. Parents of a child normally care about the child.

As we will see, defaults can have two types of exceptions: strong exceptions and weak exceptions (Gelfond and Kahl, 2014). Strong exceptions refute a default's conclusion and derive the opposite of the default as sentence (6) should do. Weak exceptions render a default inapplicable and do not support certain conclusions as sentence (7) should do (the reasoner should not conclude that Alice cares about Sam).

In order to achieve this form of non-monotonic reasoning, we need to translate sentence (5') via a DRS into a suitable rule in ASP. Before we show how this can be done, we discuss in the next section what the target representation for defaults looks like in the ASP paradigm.

4 Representing Defaults in ASP

ASP is well suited for representing defaults since it distinguishes between two kinds of negation: classical negation and negation as failure. Combining both forms of negation allows us to express, for example, the closed world assumption, i.e., the assumption that a literal that is currently not known to be true is false. The closed world assumption is an example of a default (Reiter, 1978). For instance, the following ASP program includes a closed world assumption rule that combines classical negation (`-`) and negation as failure (`not`):

```
r(1). r(2). s(3). s(4). q(1,3). q(2,3).
-q(X,Y) :- r(X), s(Y), not q(X,Y).
```

This ASP program has a unique answer set that includes the two negative literals `-q(2,4)` and `-q(1,4)`:

```
{ r(1) r(2) s(3) s(4) q(1,3) q(2,3)
  -q(2,4) -q(1,4) }
```

It is interesting to note that an ASP program that combines strong negation and weak negation can apply the closed world assumption rule to some of its literals and leave other literals in the scope of the open world assumption. The same technique of combining classical negation and negation as failure can be used in our context.

[1] (Pelletier and Asher, 1997) convincingly argue that there exists no univocal (probabilistic-oriented) quantifier (like **most** *parents*) that characterises all defaults.

A default that states that most elements *X* of a class *c* have property *p* can be represented by the following rule in ASP (Gelfond and Kahl, 2014).

```
p(X) :- c(X), not ab(d(X)), not -p(X).
```

That means `p(X)` holds if `c(X)` holds and it cannot be shown (`not`) that `X` is abnormal (`ab`) with respect to a default `d` and that it cannot be shown (`not`) that `-p(X)` does hold. Note that `X` might be abnormal and that `-p(X)` might hold but we currently cannot find any evidence that this is the case.

We can use the same technique to represent sentence (5') that results in a default (`d(care(X,Y))`) with the help of the following rule:

```
care(X,Y) :-
  parent(X,Y), child(Y),
  not ab(d(care(X,Y))),
  not -care(X,Y).
```

The subsequent ASP program that uses this default rule and represents the information derived from sentence (6) and (7) finally leads to a consistent answer set:

```
child(sam).
father(john,sam).
mother(alice,sam).
parent(X,Y) :- father(X,Y), child(Y).
parent(X,Y) :- mother(X,Y), child(Y).

-care(john,sam).
absent(alice).
care(X,Y) :-
  parent(X,Y), child(Y),
  not ab(d(care(X,Y))),
  not -care(X,Y).
```

Note that sentence (6) is a strong exception to the default rule and refutes the conclusion of the default. So far, there is no information in the ASP program that states that the default `d` is not applicable to `care(X,Y)`. In order to ensure that the weak exception `absent(alice)` derived from sentence (7) is correctly processed and can render the default `d` inapplicable, we need to add a so-called cancellation axiom to the ASP program:

```
ab(d(care(X,Y))) :-
  parent(X,Y), child(Y),
  not -absent(X).
```

This cancellation axiom makes sure that an absent parent of a child can be viewed as a weak exception to the default. Adding this cancellation axiom to our ASP program results in a unique answer set where the conclusion `care(alice,sam)` is abnormal (`ab`) with respect to the default `d` and the literal `care(alice,sam)` is unknown to the answer set:

```
{ child(sam) father(john,sam)
  mother(alice,sam) absent(alice)
  -care(john,sam) parent(john,sam)
  parent(alice,sam)
  ab(d(care(alice,sam)))
  ab(d(care(john,sam))) }
```

Note that if our ASP program would contain the information `-absent(alice)` instead of `absent(alice)`, then the default rule would succeed and the answer set would contain the information that Alice cares about Sam. If none of these two literals is available in the ASP, then the default rule does not apply.

5 Translating the CNL with Defaults

Our existing CNL processor consists of a chart parser, a unification-based grammar and a domain-specific lexicon (White and Schwitter, 2009). The language processor takes a CNL text as input and generates an extended DRS (Schwitter, 2012) for that text. This DRS is then translated into an ASP program (Schwitter, 2013) that is executed by *clingo* (Gebser et al., 2011), an ASP tool.

In our case, a DRS is a term of the form `drs(U,C)`. The first argument `U` is a list of discourse referents (i.e. quantified variables), and the second argument `C` is a list of simple and complex conditions for these discourse referents. Simple conditions are logical atoms and complex conditions are built from other DRSs with the help of logical connectors. Our extended DRS uses a reified notation for logical atoms together with a small number of predefined predicates.

Since our existing CNL already distinguishes between classical negation and negation as failure, it is possible to express rules that enforce the closed world assumption (as introduced in the last section). For example, the conditional sentence:

8. If there is no evidence that a mother of a child is absent then the mother is not absent.

is translated during the parsing process into the following DRS:

```
[]
  [A,B]
    relation(mother,A,B)
    object(B,child)
    NAF
      []
      property(absent,A)
  ==>
  []
    NEG
      []
      property(absent,A)
```

This DRS consists of a complex implicative condition (==>). Note that the CNL expression *there is no evidence that* results in a negation as failure operator (NAF) in the antecedent of this DRS and the translation of the expression *does not* leads to a classical negation (NEG) in the consequent. This extended DRS is then further translated into a strict rule in ASP:

```
-absent(X) :-
  mother(X,Y), child(Y),
  not absent(X).
```

This works fine; however, we also have to guarantee that sentences such as (5') are correctly translated into a default rule in an ASP program.

In order to achieve this, this sentence is first translated into the following DRS that uses a new operator ∼∼> to mark this kind of default:

```
[]
  [A,B]
    relation(parent,A,B)
    object(B,child)
  ∼∼>
  []
    predicate(care,A,B)
```

This operator helps us to distinguish between a strict rule and a default rule. The subsequent translation process into an ASP program identifies the type of operator in the DRS and translates the DRS into the following ASP rule:

```
care(X,Y) :-
  parent(X,Y), child(Y),
  not ab(d(care(X,Y))),
  not -care(X,Y).
```

This translation is achieved with the help of a Prolog program that takes a DRS as input and applies templates of the following form to generate the default rule:

```
[ Predicate, ':-',
  Term,
  not ab(d(Predicate)),
  not -Predicate ]
```

It is interesting to see that our existing CNL has already all the ingredients that are necessary in order to paraphrase this default rule. But in contrast to sentence (5') that uses the keyword *normally*, we end up with a rather lengthy circumscription:

9. If there is no evidence that a parent abnormally cares about a child and there is no evidence that the parent does not care about the child then the parent cares about the child.

But note that the translation of sentence (5') and sentence (9) result in the same default rule. In the case of sentence (9), the expression *abnormally cares about* translates into the literal `ab(d(care(X,Y)))`.

Finally, we want to make sure that also cancellation axioms that implement a weak exception can be expressed in CNL and be translated into ASP rules. For example, the conditional sentence:

10. If there is no evidence that a parent of a child is not absent then the parent abnormally cares about the child.

represents a cancellation axiom and results in the following ASP rule:

```
ab(d(care(X,Y))) :-
  parent(X,Y), child(Y),
  not -absent(X).
```

Note that we could completely replace the expression *not absent* in our CNL specification by the positive expression *present* and we would end up with the same kind of inferences. That means strong negation is actually only a modelling convenience in ASP (Brewka et al., 2011) but does not increase the expressive power of the language.

6 Conclusion

Most of what we know about the world is normally true, with a few exceptions. Defaults allow us to draw conclusions based on knowledge that is common and normally the case. These defaults are sensitive to strong and weak exceptions and are important to non-monotonic reasoning that plays an important role in everyday human communication.

In this paper, we showed how an existing controlled natural language can be extended to accommodate statements about defaults and exceptions, how these statements can be translated via discourse representation structures into an answer set program, and how this answer set program can be used for automated reasoning. The general strategy for representing these defaults and exceptions to them in answer set programming is based on the work of (Gelfond and Kahl, 2014) and provides a clean and computationally elegant way to deal with these constructions.

To the best of our knowledge, our controlled natural language is the first one that supports the specification of defaults and exceptions in a well-defined subset of natural language and provides access to this form of non-monotonic reasoning via answer set programming.

References

Gerhard Brewka, Thomas Eiter, Miroslaw Truszczyński. 2011. Answer Set Programming at a Glance. In: *Communications of the ACM*, Vol. 54, No. 12, December.

Jan van Eijck and Hans Kamp. 2011. Discourse Representation in Context. In: J. van Benthem and A. ter Meulen (eds.), *Handbook of Logic and Language*, Second Edition, Elsevier, pp. 181–252.

Thomas Eiter, Giovambattista Ianni, Thomas Krennwallner. 2009. Answer Set Programming: A Primer. In: *Reasoning Web. Semantic Technologies for Information Systems*, LNCS, Vol. 5689, pp. 40–110.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Marius Schneider. 2011. Potassco: The Potsdam Answer Set Solving Collection. In: *AI Communications*, Vol. 24, No. 2, pp. 105–124.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Torsten Schaub. 2012. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan and Claypool Publishers.

Michael Gelfond and Vladimir Lifschitz. 1988. The stable model semantics for logic programming. In: R. Kowalski and K. Bowen (eds.), *Proceedings of International Logic Programming Conference and Symposium*, pp. 1070–1080.

Michael Gelfond and Yulia Kahl. 2014. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents.* The Answer-Set Programming Approach, Cambridge University Press.

Hans Kamp and Uwe Reyle. 1993. *From Discourse to Logic: Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory.* Kluwer, Dordrecht.

Tobias Kuhn. 2013. A Survey and Classification of Controlled Natural Languages. In: *Computational Linguistics*, MIT Press.

Vladimir Lifschitz. 2008. What Is Answer Set Programming? In: *Proceedings of AAAI'08*, Vol. 3, pp. 1594–1597.

Francis Jeffry Pelletier and Nicolas Asher. 1997. Generics and Defaults. In: J. van Benthem and A. ter Meulen (eds.), *Handbook of Logic and Language*, Elsevier Science, Chapter 20, pp. 1125–1177.

Raymond Reiter. 1978. On closed world data bases. In: H. Gaillaire and J. Minker (eds.), *Logic and Data Bases*, Plenum Press, New York, pp. 55–76.

Rolf Schwitter. 2012. Answer Set Programming via Controlled Natural Language Processing. In: T. Kuhn and N. E. Fuchs (eds.), *CNL 2012*, LNCS 7427, Springer, pp. 26–43.

Rolf Schwitter. 2013. The Jobs Puzzle: Taking on the Challenge via Controlled Natural Language Processing. In: *Journal of Theory and Practice of Logic Programming*, Vol. 13, Special Issue 4-5, pp. 487–501.

Coline White and Rolf Schwitter. 2009. An Update on PENG Light. In: L. Pizzato and R. Schwitter (eds.), *Proceedings of ALTA 2009*, Sydney, Australia, pp. 80–88.