Logic Programs vs. First-Order Formulas in Textual Inference

Yuliya Lierler and Vladimir Lifschitz
Department of Computer Sciences
University of Texas at Austin

{yuliya,vl}@cs.utexas.edu

Abstract

In the problem of recognizing textual entailment, the goal is to decide, given a text and a hypothesis expressed in a natural language, whether a human reasoner would call the hypothesis a consequence of the text. One approach to this problem is to use a first-order reasoning tool to check whether the hypothesis can be derived from the text conjoined with relevant background knowledge, after expressing all of them by first-order formulas. Another possibility is to express the hypothesis, the text, and the background knowledge in a logic programming language, and use a logic programming system. We discuss the relation of these methods to each other and to the class of effectively propositional reasoning problems. This leads us to general conclusions regarding the relationship between classical logic and answer set programming as knowledge representation formalisms.

1 Introduction

In the problem of recognizing textual entailment, the goal is to decide, given a text T and a hypothesis H expressed in a natural language, whether a human reasoner would call H a consequence of T. The following example is No. 115 in the collection of problems proposed as the Second PASCAL Recognizing Textual Entailment Challenge [Bar-Haim *et al.*, 2006]:

> T: The World Bank has also been criticized for its role in financing projects that have been detrimental to human rights and the natural environment.
>
> H: The World Bank is criticized for its activities.
>
> *Expected answer*: Yes.

Recognizing textual entailment is a special case of a more general and practically important problem, textual query answering.

To recognize the fact that H is "entailed" by T, we often need to use some background commonsense knowledge. For instance, in the example above it is essential that financing is an activity.

The approach to recognizing textual entailment employed in [Bos and Markert, 2005] and implemented in the system Nutcracker[1] can be summarized as follows:

(i) T and H are represented first by discourse representation structures [Kamp and Reyle, 1993] and then by first-order formulas,

(ii) potentially relevant background knowledge is identified and expressed by a first-order formula BK,

(iii) an automated reasoning system is used to check whether the implication

$$T \wedge BK \rightarrow H \tag{1}$$

is logically valid.

Related work is described in [Akhmatova, 2005, Fowler *et al.*, 2005].

The approach to the problem proposed in [Baral *et al.*, 2005, Tari and Baral, 2005, Nouioua and Nicolas, 2006] is similar, except that it relies on logic programs as the representation language instead of first-order formulas, and on logic programming systems as computational tools instead of first-order reasoners. The following example comes from the introduction to [Baral *et al.*, 2005]:

> T: In Paris, on March 15th, John packed his laptop in the carry-on luggage and took a plane to Baghdad.
>
> H: His laptop was in Baghdad on March 16th.
>
> *Expected answer*: Yes.

Here again some background commonsense knowledge is needed to recognize that *yes* is the correct answer: a trip from Paris to Baghdad by air does not normally take more than a day; a person and his carry-on luggage are normally in the same city. Baral *et al.* represent the text T by a set of rules,[2] along with background knowledge BK; H is represented by a ground

[1] `http://www.cogsci.ed.ac.uk/~jbos/RTE/` .

[2] This is done manually; automating the translation is mentioned in the paper as future work.

atom. Then an answer set solver and a constraint logic programming system are used to establish the fact that H is entailed by logic program $T \cup BK$.

Each of the two knowledge representation languages—first-order formulas and logic programs—has its own advantages. A first-order formula may have a complex, nested form; this is essential because discourse representation structures are often deeply nested. On the other hand, the semantics of logic programs is nonmonotonic; this is crucial when background commonsense knowledge is expressed by defaults (note the word "normally" in the examples above).

In this paper we argue, however, that these two versions of the logic approach to textual entailment have more in common than meets the eye. A large part of the work done by Bos and Markert can be understood in terms of the logic programming methodology advocated by Baral *et al.* Many textual entailment problems used to test the Nutcracker system can be solved by the answer set solver DLV[3] instead of the first-order theorem prover VAMPIRE[4] and the model builder PARADOX[5] that were actually employed in Nutcracker experiments.

The first-order reasoning problems that can be naturally expressed by logic programs have a distinctive syntactic feature: they belong to the "effectively propositional," or "near-propositional" formulas [Schulz, 2002]. The relationship between effectively propositional reasoning (EPR) and answer set programming (ASP) [Lifschitz, 1999, Marek and Truszczyński, 1999, Niemelä, 1999] is one of the topics discussed in this paper. This will bring us, at the end of the paper, to some general conclusions regarding the relationship between classical logic and answer set programming as knowledge representation formalisms.

2 Representing EPR Formulas by Logic Programs

We consider here first-order formulas that may contain equality and object constants, but not function constants of arity > 0. An *EPR formula* is the universal closure of a quantifier-free formula in conjunctive normal form. We will show how to turn any EPR formula F into a logic program $\pi(F)$ such that $\pi(F)$ has a stable model iff F is satisfiable.

In the definition of π we assume that every clause in F is written as an implication with a conjunction of atoms (possibly empty) in the antecedent, and a disjunction of atoms (possibly empty) in the consequent:

$$A_1 \wedge \cdots \wedge A_m \rightarrow A_{m+1} \vee \cdots \vee A_n. \tag{2}$$

[3] `http://www.dbai.tuwien.ac.at/proj/dlv/` .

[4] `http://en.wikipedia.org/wiki/Vampire_theorem_prover` .

[5] `http://www.math.chalmers.se/~koen/paradox/` .

Besides the predicate constants occurring in F, the program $\pi(F)$ will contain two new predicate constants: the unary constant u (for "universe") and the binary constant eq (for "equals"). For any atomic formula A, by A^{eq} we denote $eq(t_1, t_2)$ if A is an equality $t_1 = t_2$, and A otherwise.

Program $\pi(F)$ consists of

(i) the facts $u(c)$ for all object constants c occurring in F;

(ii) the rules

$$\begin{array}{l} eq(X,X) \leftarrow u(X) \\ eq(Y,X) \leftarrow eq(X,Y) \\ eq(X,Z) \leftarrow eq(X,Y), eq(Y,Z); \end{array}$$

(iii) the rules

$$p(Y_1, \dots, Y_k) \leftarrow p(X_1, \dots, X_k), eq(X_1, Y_1), \dots, eq(X_k, Y_k)$$

for all predicate constants p occurring in F;

(iv) the disjunctive rules

$$A_{m+1}^{eq}; \dots; A_n^{eq} \leftarrow A_1^{eq}, \dots, A_m^{eq}, u(X_1), \dots, u(X_k)$$

corresponding to the conjunctive terms (2) of F, where $X_1, \dots, X_k$ are the variables that occur in the consequent $A_{m+1} \vee \dots \vee A_n$ of (2) but do not occur in its antecedent $A_1 \wedge \dots \wedge A_m$.

If, for instance, F is

$$\forall X (p(a) \wedge p(b) \wedge \neg p(c) \wedge (p(X) \rightarrow X = a)) \tag{3}$$

then $\pi(F)$ consists of the rules

$$\begin{array}{l} u(a) \\ u(b) \\ u(c) \\ \\ eq(X,X) \leftarrow u(X) \\ eq(Y,X) \leftarrow eq(X,Y) \\ eq(X,Z) \leftarrow eq(X,Y), eq(Y,Z) \\ \\ p(Y) \leftarrow p(X), eq(X,Y) \\ \\ p(a) \\ p(b) \\ \leftarrow p(c) \\ eq(X,a) \leftarrow p(X). \end{array} \tag{4}$$

As usual in answer set programming, by a model (stable model) [Gelfond and Lifschitz, 1988, Gelfond and Lifschitz, 1991] of $\pi(F)$ we understand a model (stable model) of the corresponding ground program. It is clear that $\pi(F)$ is a (possibly disjunctive) program without negation. Its stable models are simply minimal models. Each rule of $\pi(F)$ is safe—every variable occurring in its head occurs also in its body. The stable models of this program can be generated by the answer set solver DLV.[6]

Theorem *For any EPR formula F, the following conditions are equivalent:*

(i) F is satisfiable,

(ii) $\pi(F)$ has a model,

(iii) $\pi(F)$ has a stable model.

If F does not contain equality then this assertion remains valid if all rules containing eq are dropped from $\pi(F)$.

For instance, formula (3) is satisfiable; accordingly, program (4) has a stable model:

$$\{u(a),\ u(b),\ u(c),\ p(a),\ p(b),\ eq(a,a),\ eq(b,b),\ eq(c,c),\ eq(a,b),\ eq(b,a)\}.$$

Proof We begin by proving the second claim, that is, assume that F does not contain equality and that the rules containing eq are dropped from the program $\pi(F)$. *From (i) to (ii).* F is the universal closure of a quantifier-free formula, and it is satisfiable. Consequently F has an Herbrand model. By adding to this model the atoms $u(c)$ for all object constants c we get a model of $\pi(F)$. *From (ii) to (iii).* The result of grounding $\pi(F)$ is a finite program without negation; since it has a model, it has a minimal model. *From (iii) to (i).* By removing the atoms $u(c)$ from a model of $\pi(F)$ we get an Herbrand model of F.

Consider now the general case, when F may contain equality. Let F^* be the formula obtained from F by

- replacing each equality $t_1 = t_2$ with $eq^*(t_1, t_2)$, where eq^* is a new binary predicate constant;
- conjoining the result with the universal closures of the formulas

$$\begin{array}{l} eq^*(X,X), \\ eq^*(X,Y) \rightarrow eq^*(Y,X), \\ eq^*(X,Y) \wedge eq^*(Y,Z) \rightarrow eq^*(X,Z) \end{array}$$

[6]If we add atoms of the form $u(X)$ for "weakly restricted" variables X to the bodies of some rules, it will become also possible to find the stable models of $\pi(F)$ using two other answer set solvers—GNT (`http://www.tcs.hut.fi/Software/gnt/`) and CMODELS (`http://www.cs.utexas.edu/users/tag/cmodels`).

and

$$p(X_1, \ldots, X_k) \wedge eq^*(X_1, Y_1) \wedge \cdots \wedge eq^*(X_k, Y_k) \rightarrow p(Y_1, \ldots, Y_k)$$

for all predicate constants p occurring in F;

- converting the result to prenex form.

It is clear that F^* is an EPR formula that does not contain equality, and that it is satisfiable iff F is satisfiable. The rules of $\pi(F^*)$ that do not contain eq can be obtained from the rules of $\pi(F)$ by replacing each occurrence of eq with eq^*. Consequently $\pi(F^*)$ has a model (stable model) iff $\pi(F)$ has a model (stable model). It remains to apply the part of the theorem proved above to F^*.

This proof shows that, in the case when F does not contain equality, the models of $\pi(F)$ are essentially identical to the Herbrand models of F.

From the theorem we see that testing an EPR formula for satisfiability can be accomplished using an answer set solver. In the next section we investigate the applicability of this idea to the problem of recognizing textual entailment.

3 EPR Formulas in Experiments with Textual Entailment

Recall that Bos and Markert [2005] recognize textual entailment by determining whether implication (1) is logically valid.[7] In many cases, the negation

$$T \wedge BK \wedge \neg H \tag{5}$$

of formula (1) can be converted to a prenex form with all existential quantifiers in front of the universal quantifiers ("$\exists\forall$-prenex form"). Then the sentence F, obtained from this prenex form by Skolemization and then converting the quantifier-free part to conjunctive normal form, is an EPR formula. It is clear that (1) is logically valid iff F is unsatisfiable.

The possibility of converting (5) to $\exists\forall$-prenex form is essential because it guarantees that no function constants of arity > 0 are introduced in the process of Skolemization. It is clear that conjunction (5) can be written in $\exists\forall$-prenex form if every conjunctive term can be written in this form.

[7] Like other existing systems for recognizing textual entailment, Nutcracker is not completely reliable. Generally, formulas H and T only approximate the meanings of RTE sentence pairs. For instance, Nutcracker currently ignores the semantics of plurals, tense, and aspect. Also, formula BK may not adequately represent all relevant background knowledge.

How restrictive is this requirement? The website shown in Footnote 1 gives the first-order formulas corresponding to 799 textual entailment problems from the second PASCAL collection that were produced by Nutcracker. In 711 cases (89%), both T and $\neg H$ can be converted to $\exists\forall$-prenex form. The conjunctive terms of BK come from two sources: most are automatically extracted from WordNet, the others are hand-coded. All conjunctive terms of the first kind are universal sentences, so that each of them is $\exists\forall$-prenex. Among the hand-coded parts of BK that Bos and Markert chose to include we found several exceptions, but they are never essential: dropping the "difficult" hand-coded part of BK from any of the 711 implications (1) that we have studied never makes a logically valid formula invalid.

The model builder PARADOX terminates, in principle, on any EPR formula; it generates a model if the formula is satisfiable, and reports that it is unsatisfiable otherwise. For this reason, in each of the 711 cases (with the inessential "difficult" terms dropped) Bos and Markert would be able to test the formula for satisfiability by running PARADOX alone, without invoking also the theorem prover VAMPIRE.

In these 711 cases we could also test the logical validity of (1) by running the answer set solver DLV, as described in the previous section. Interestingly, the runtime of DLV was smaller than the runtime of the model builder PARADOX in most cases, even though DLV did some redundant work: whenever the program $\pi(F)$ has a model, it computed one of its *minimal* models.[8] We should add that all these runtimes are pretty small, usually less than a second.

The main computational difference between model builders, such as PARADOX, and answer set solvers, such as DLV, is that model builders typically start by looking for a finite model with a small universe (say, a singleton); if there is no such model then search is extended to larger universes. If the input is an EPR formula containing N object constants, then answer set solvers ground the given program with the universe of size N, which means essentially that they only look for a model of cardinality N. Good answer set solvers, such as DLV, perform grounding "intelligently" and sometimes introduce auxiliary predicates that make the size of the grounded program smaller.

It is interesting also that among the 711 EPR formulas from Nutcracker experiments that we have investigated, 514 are Horn—all their conjunctive terms (2) are definite ($n = m + 1$) or negative ($n = m$). If F is a Horn EPR formula then the program $\pi(F)$ is nondisjunctive; since there is no negation

[8]System DLV has the option `-OM-` that disables testing for minimality, but in our experiments it did not have a noticeable effect on runtimes.

in this program, it can have at most one stable model. Note that deciding whether a ground nondisjunctive program without negation has a model can be done in linear time.

4 Conclusion

The properties of the translation π established in this paper suggest that EPR reasoning can be described as the common part of classical first-order logic and logic programming under the stable model semantics (Figure 1). In terms of expressiveness, the availability of formulas more complex than

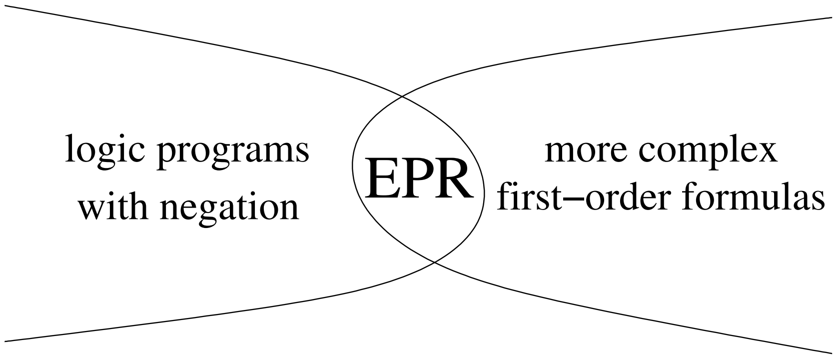

Figure 1: Logic programs vs. first-order formulas: a comparison

EPR is a strong side of classical logic; the availability of negation as failure is a strong side of declarative logic programming. Representations used in the existing work on the analysis of textual inference in terms of classical logic belong mostly to the common part of the two areas.

Acknowledgements

Many thanks to Günther Görz for introducing one of us (Yu.L.) to the exciting world of computational linguistics, to Michael Gelfond for many interesting conversations about textual query answering, to Johan Bos for helping us understand the findings reported on the Nutcracker web page, and to Gerald Pfeifer for expert advice on the use of DLV. Günther, Michael, Johan and Lenhart Schubert provided valuable comments on a draft of this note. This research was partially supported by the National Science Foundation under Grant IIS-0412907.

References

[Akhmatova, 2005] Elena Akhmatova. Textual entailment resolution via atomic propositions. In *Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment*, 2005.

[Bar-Haim *et al.*, 2006] Roy Bar-Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. The Second PASCAL Recognising Textual Entailment Challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*, Venice, Italy, 2006.

[Baral *et al.*, 2005] Chitta Baral, Gregory Gelfond, Michael Gelfond, and Richard Scherl. Textual inference by combining multiple logic programming paradigms. In *AAAI Workshop on Inference for Textual Question Answering*, 2005.

[Bos and Markert, 2005] Johan Bos and Katja Markert. Recognising textual entailment with logical inference. In *Proceeding of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 628–635, 2005.

[Fowler *et al.*, 2005] Abraham Fowler, Bob Hauser, Daniel Hodges, Ian Niles, Adrian Novischi, and Jens Stephan. Applying COGEX to recognize textual entailment. In *Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment*, 2005.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080, 1988.

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[Kamp and Reyle, 1993] Hans Kamp and Uwe Reyle. *From discourse to logic*, volume 1,2. Kluwer, 1993.

[Lifschitz, 1999] Vladimir Lifschitz. Action languages, answer sets and planning. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 357–373. Springer Verlag, 1999.

[Marek and Truszczyński, 1999] Victor Marek and Mirosław Truszczyński. Stable models and an alternative logic programming paradigm. In *The*

Logic Programming Paradigm: a 25-Year Perspective, pages 375–398. Springer Verlag, 1999.

[Niemelä, 1999] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999.

[Nouioua and Nicolas, 2006] Farid Nouioua and Pascal Nicolas. Using answer set programming in an inference-based approach to natural language semantics. In *Proceedings of the Fifth Workshop on Inference in Computational Semantics (ICoS)*, 2006.

[Schulz, 2002] Stephan Schulz. A comparison of different techniques for grounding near-propositional CNF formulae. In *Proceedings of the 15th International FLAIRS Conference*, pages 72–76, 2002.

[Tari and Baral, 2005] Luis Tari and Chitta Baral. Using AnsProlog with Link Grammar and WordNet for QA with deep reasoning. In *AAAI Workshop on Inference for Textual Question Answering*, 2005.