First order logic and commonsense reasoning: a path less travelled

Tanel Tammet[1], Dirk Draheim[2], Priit Järv[3] and Martin Verrev[4]

[1] *Tallinn University of Technology, Tallinn, Estonia*
[2] *Tallinn University of Technology, Tallinn, Estonia*
[3] *Tallinn University of Technology, Tallinn, Estonia*
[4] *Tallinn University of Technology, Tallinn, Estonia*

Abstract

The context of the paper is developing logic-based components for hybrid – machine learning plus logic – commonsense question answering systems. The paper presents the main principles and several lessons learned from implementing an automated reasoner able to handle both undecidable exceptions and numerical confidences for full first order logic. Although the described reasoner is based on the resolution method, some of these lessons may be useful for the further development of ASP systems.

Keywords

Commonsense reasoning, default logic, automated reasoning

1. Introduction

The focus of the paper is on several aspects of devising symbolic systems for commonsense reasoning, particularly for problems stated in natural language. Although it is commonly accepted that during the last decade the machine learning systems have achieved significantly more successes than symbolic reasoning systems, it is often argued (see [1], [2]) that hybrid systems have more potential for the future than either pure symbolic or pure machine learning systems. In order to research and experiment with hybrid systems, we need symbolic systems which are suited for the task.

We are developing a symbolic reasoning system GK for exactly this purpose: to obtain a symbolic reasoning component suitable for commonsense reasoning tasks, which could then be integrated with the machine learning components. As a complement to GK we are developing a semantic parser for natural language, which outputs logical statements and questions suitable for GK. The semantic parser first uses the Stanza parser [3] producing a Universal Dependencies graph for natural language sentences and then converts this

2nd Workshop on Goal-directed Execution of Answer Set Programs (GDE'22), August 1, 2022
EMAIL: tanel.tammet@taltech.ee (T. Tammet); dirk.draheim@taltech.ee (D. Draheim); priit.jarv1@taltech.ee (P. Järv); martin.verrev@taltech.ee (M. Verrev)
ORCID: 0000-0003-4414-3874 (T. Tammet); 0000-0003-3376-7489 (D. Draheim); 0000-0001-7725-543X (P. Järv)

graph to first order predicate calculus enhanced with constructions for "softening" the logic.

The logical statements thus created and then handled by GK are currently softened in two separate ways. First, in [4] we introduce a CONFER framework were each clause obtained from the statements can have a probability-like confidence number 0...1 attached. Second, in [5] we describe the mechanisms for handling clauses which contain blocker atoms: atoms whose derivability is checked for each finished proof containing such atoms, and the provability of which disqualifies the proof containing the atom. The blocker atoms are essentially a mechanism of implementing default logic as introduced by R. Reiter in [6].

2. Standard and nonstandard reasoning by GK

The GK reasoning system available at https://logictools.org/gk/ is a significantly extended version of the high performance resolution prover GKC [7] for classical logic. GK, as well as GKC, is written in C. A tutorial and a set of default logic and confidence calculation example problems along with proofs from GK are also available at http://logictools.org/gk, along with the large integrated knowledge bases built from WordNet, Quasimodo [8] and ConceptNet [9] knowledge bases, usable by GK.

GKC, in turn, is optimized for large problems. Some of these optimization principles are:

- Pervasive use of hash indexes instead of tree indexes commonly used in automated reasoners.
- Highly efficient forward subsumption algorithms using feature vectors based on hashes.
- Focus on using goal-directed search, i.e. the variations of the set-of-support algorithm.
- Clause selection queues aimed at spreading the clause selection during search relatively uniformly over different important categories of derived clauses, like the query, assumptions, more relevant rules/facts, less relevant rules/facts.

The nonstandard features of GK are:

- Finding answer substitutions by using a variation of the well-known answer literal mechanism.
- Surviving inconsistent knowledge bases by discarding proofs not containing goal clauses.
- Searching for derivations of both positive and negative instances of queries/answers.
- Calculating the diminishing-or-equal confidences of the results of rule applications.
- Calculating the increasing-or-equal cumulative confidences from different derivations of a clause.
- Handling and checking the blocker literal expressing exceptions to rules: essentially, default logic.

Consider a typical example for default logic: birds can normally fly, but penguins cannot fly. The classical logic part

$$penguin(p) \,\&\, bird(b) \,\&\, \forall x.penguin(x) \Rightarrow bird(x) \,\&\, \forall x.penguin(x) \Rightarrow \neg fly(x).$$

is extended with the default rule $bird(x) : fly(x) \vdash fly(x)$.

From here we can derive that an arbitrary bird b can fly, but a penguin p cannot. The default rule cannot be applied to p, since a contradiction is derivable from $fly(p)$. For example, the "birds can fly" default rule is represented as a clause

$$\neg\texttt{bird(X)} \vee \texttt{fly(X)} \vee \texttt{block(0,neg(fly(X)))}$$

where `X` is a variable and `neg(fly(X))` encodes the negated justification. The first argument of the blocker (0 above) encodes priority information covered later.

Since first order logic is not decidable, it is in general impossible to guarantee that some blocker literal is not derivable. Hence, the standard approach for handling default logic has been creating a large ground instance KB_g of the KB, and then performing decidable propositional reasoning on the KB_g.

Almost all the existing implementations of default logic like DeReS [10], DLV2 [11] or CLINGO [12], with the exception of s(CASP) [13], follow the same principle. More generally, the field of *Answer Set Programming* (ASP), see [14], is devoted to this approach. As an exception, the s(CASP) system [13] solves queries without the grounding step and is thus better suited for large domains.

The approach implemented in GK accepts the lack of logical omniscience. Justification checking of blockers (i.e. exceptions) is delayed until a first-order proof is found; after that recursively deepening checks are performed with diminishing time limits. Thus, GK first produces a potentially large number of different candidate proofs and then enters a recursive checking phase. The results produced by GK thus depend on the time limits and are not stable.

The search algorithm integrates exception checking with confidence calculations and searching for derivations of both a positive version and a negated version of a query or a particular answer substitution. A blocker is considered to be proved in case the resulting confidence is over a pre-determined configurable threshold, by default 0.5. Thus, the whole search tree for a query consists of two types of interleaved layers: positive/negative confidence searches and blocker checking searches, the latter type potentially making the tree arbitrarily deep up to the minimal time limit threshold.

3. Priorities

The concept of *priorities* for default rules has been well investigated, with several mechanisms proposed: see [15]. Typically an ordering of defaults is introduced, based on specificity: default rules for a more specific class of objects should take priority over rules for more general classes. For example, since birds (who typically do fly) are physical objects and physical objects typically do not fly, we have contradictory default rules

describing the flying capability of arbitrary birds. Since birds are a subset of physical objects, the flying rule of birds should have a higher priority than the non-flying rule of physical objects.

The usage of priorities in proof search by GK is simple: when checking a blocker with a given priority, it is not allowed to use default rules with a lower priority. We encode priority information as a first argument of the blocker literal, offering several ways to determine priority: either as an integer, a taxonomy class number, a string in a taxonomy or a combination of these with an integer. For automatically using specificity we employ taxonomy classes: a class has a higher prirority than those above it on the taxonomy branch. To enable more fine-grained priorities, an integer can be added to the term like $(”`bird`”, 2) generating a lexicographic order.

From our experiments with the commonsense reasoning tasks it has become clear that the question of encoding and assigning priorities is critical for any nontrivial questions posed. For example, consider “Birds can fly” and “Baby birds cannot fly”: there is no class taxonomy involved. Moreover, we clearly see that there is a need to be able to attach a potentially arbitrarily deep list of lecixographically comparable priority information to each exception. Consider “Birds can eat meat” and “Baby birds cannot eat raw meat”. As an another nontrivial example for encoding exceptions and priorities, consider the following commonsense observation: birds and airplanes without wings cannot fly. This can be generalized to a plausible rule: if a class has a component used for some capability, then an instance of the class without this component does not have this capability. A naive usage of priorities for encoding this rule will lead to unstable results.

4. A Comparison with the ASP approach

The following small example illustrates the fundamental difference of GK from the existing ASP systems for default logic. Although the example is favorable to GK, we argue that serious commonsense reasoning knowledge bases almost certainly have to contain both function terms and a large numbers of constants and facts/rules containing these constants, similarly to the following minimalistic examples. For example, a proper logical representation of even relatively simple natural language sentences tends to require interleaved nested general and existential aquantifiers, from which Skolemization generates function symbols. Also, axiomatizing uniqueness of the properties of objects is most naturally achieved with the help of function symbols. The standard penguins and birds example presented above in the ASP syntax is

```
bird(b1).
penguin(p1).
bird(X) :- penguin(X).
flies(X) :- bird(X), not -flies(X).
-flies(X) :- penguin(X).
```

Both GK and the ASP systems clingo 5.4.0, dlv 2.1.1 and s(CASP) 0.21.10.09 give an expected answer to the queries `flies(b1)` and `flies(p1)`. However, when we add the rules (not really used by these queries)

```
bird(father(X)) :- bird(X).
penguin(father(X)) :- penguin(X).
```

none of these ASP systems terminate for these queries, while GK does solve the queries as expected. Notably, as pointed out by the author of s(CASP), this system does terminate for the reformulation of the same problem with the two replacement rules

```
flies(X) :- bird(X), not abs(X).
abs(X) :- penguin(X).
```

while clingo and dlv do not terminate. When we instead add the following facts and rules (again, not actually used by the query)

```
father(b1,b2).
father(p1,p2).
...
father(bN-1,bN).
father(pN-1,pN).

ancestor(X,Y):- father(X,Y).
ancestor(X,Y) :- ancestor(X,Z), ancestor(Z,Y).
```

for a large N, s(CASP) does not terminate and clingo and dlv become slow for `flies(b1)`: ca 8 seconds for $N = 500$ and ca 1 minute for $N = 1000$ on a laptop with a 10-th generation i7 processor. GK solves the same question with $N = 1000$ under half a second and with $N = 100000$ under three seconds: the latter problem size is clearly out of scope of the capabilities of existing ASP systems.

References

[1] G. Marcus, The next decade in AI: four steps towards robust artificial intelligence, CoRR abs/2002.06177 (2020). URL: https://arxiv.org/abs/2002.06177. arXiv:2002.06177.

[2] G. Marcus, E. Davis, Rebooting AI: Building artificial intelligence we can trust, Vintage, 2019.

[3] P. Qi, Y. Zhang, Y. Zhang, J. Bolton, C. D. Manning, Stanza: A Python natural language processing toolkit for many human languages, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, 2020. URL: https://nlp.stanford.edu/pubs/qi2020stanza.pdf.

[4] T. Tammet, P. Järv, D. Draheim, Confidences for commonsense reasoning, in: S. G. Platzer A. (Ed.), Automated Deduction – CADE 28. CADE 2021., volume 12699 of *LNCS*, Springer, 2021, pp. 507–524.

[5] T. Tammet, P. Järv, D. Draheim, Gk: Implementing full first order default logic for commonsense reasoning (system description), in: IJCAR 2022: International Joint Conference on Automated Reasoning., LNCS, pending publication, Springer, 2022.

[6] R. Reiter, A logic for default reasoning, Artificial Intelligence 13 (1980) 81–132.
[7] T. Tammet, GKC: A reasoning system for large knowledge bases, in: P. Fontaine (Ed.), Proc. of CADE'2019 – the 27th Intl. Conf. on Automated Deduction, volume 11716 of *LNCS*, Springer, 2019, pp. 538–549.
[8] J. Romero, S. Razniewski, K. Pal, J. Z. Pan, A. Sakhadeo, G. Weikum, Commonsense properties from query logs and question answering forums, in: W. Zhu, D. Tao, X. Cheng, P. Cui, E. A. Rundensteiner, D. Carmel, Q. He, J. X. Yu (Eds.), Proc. of CIKM'19 – the 28th ACM Intl. Conf. on Information and Knowledge Management, ACM, 2019, pp. 1411–1420.
[9] R. Speer, J. Chin, C. Havasi, ConceptNet 5.5: An open multilingual graph of general knowledge, in: S. P. Singh, S. Markovitch (Eds.), Proc. of AAAI'2017 – the 31st AAAI Conf. on Artificial Intelligence, AAAI, 2017, pp. 4444–4451.
[10] P. Cholewinski, V. W. Marek, M. Truszczynski, Default reasoning system deres, KR 96 (1996) 518–528.
[11] M. Alviano, F. Calimeri, C. Dodaro, D. Fuscà, N. Leone, S. Perri, F. Ricca, P. Veltri, J. Zangari, The asp system dlv2, in: M. Balduccini, T. Janhunen (Eds.), Logic Programming and Nonmonotonic Reasoning, volume 11716 of *LNCS*, Springer, Cham, 2017, pp. 215–221.
[12] R. Kaminski, T. Schaub, P. Wanko, A tutorial on hybrid answer set solving with clingo, in: Reasoning Web. Semantic Interoperability on the Web. Reasoning Web 2017., volume 10370 of *LNCS*, Springer, 2017, pp. 167–203.
[13] J. Arias, M. Carro, E. Salazar, K. Marple, G. Gupta, Constraint answer set programming without grounding, Theory and Practice of Logic Programming 18 (2018) 337–354.
[14] V. Lifschitz, Answer set programming, Springer Berlin, 2019.
[15] G. Brewka, Adding priorities and specificity to default logic, in: European Workshop on Logics in Artificial Intelligence, Springer, 1994, pp. 247–260.