Lossless Semantic Round-Tripping in PENG^{ASP}

Rolf Schwitter
Macquarie University, Sydney, Australia
Rolf.Schwitter@mq.edu.au

Abstract

The PENG^{ASP} system supports the writing of textual specifications with the help of a smart text editor that possesses knowledge about the structure of the specification language. Specifications written in PENG^{ASP} are incrementally translated into executable answer set programs and vice versa. That means the system allows for lossless semantic round-tripping between a human-readable specification and an answer set program. This functionality is achieved by a single bi-directional logic grammar that serves at the same time as a text processor and a text generator. We demonstrate that the PENG^{ASP} system can be used to bridge the gap between a (seemingly) informal specification and an executable answer set program.

1 Introduction

Thou shalt choose an appropriate notation.
[Bowen and Hinchey, 2006]

Formal specifications are ideal for software developers, but not for domain specialists who are often not familiar with formal notations [Fuchs *et al.*, 2008]. Virtually any initial document for a system specification is written in natural language [Berry and Kamsties, 2004], since these documents need to be understood by domain specialists as well as software developers. Natural language looks like a good candidate for writing specifications [Dalpiaz *et al.*, 2018]; however, natural language is ambiguous and vague [Parnas, 2010], if not used in a restricted and precise form [Kuhn, 2014].

The PENG^{ASP} system tries to balance the tension between an informal and a formal notation by providing a controlled natural language (CNL) as a specification language and by supporting the writing process of a specification with a smart text editor [Guy and Schwitter, 2017]. The PENG^{ASP} system focuses on writing specifications in CNL that can be translated into executable answer set programs (ASP) and vice versa [Schwitter, 2018]. The latest version of the PENG^{ASP} system supports the writing of temporal specifications and uses an ASP-based adaptation of the event calculus [Kowalski and Sergot, 1986; Lee and Palla, 2012; Mueller, 2015; Schwitter, 2019] for automated reasoning with the ASP system *clingo* [Gebser *et al.*, 2019].

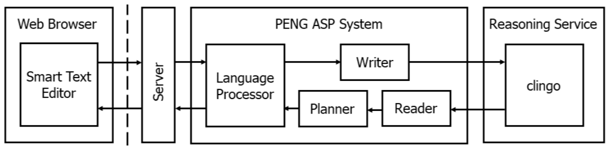

Figure 1: Architecture

2 PENG^{ASP} System Architecture

The smart text editor of the PENG^{ASP} system runs in any modern web browser and communicates with a Prolog HTTP server via JSON objects. These JSON objects are translated into Prolog terms and are then processed by the language processor of the PENG^{ASP} system (see architecture Figure 1).

The language processor consists of four main components: a unification-based grammar, a chart parser, an anaphora resolution module, and a lexicon for function words and one for content words. In the case of text processing, the language processor communicates via a writer module with the reasoning service (*clingo*) supported by a linguistically-motivated ASP implementation of the event calculus [Schwitter, 2019]. The writer translates the internal ASP format that the language processor constructs into an executable ASP program. In the case of text generation, the reasoning service communicates via a reader module and a planner module with the language processor. The reader reconstructs the internal ASP format and the planner aggregates the internal ASP format for optimised linguistic rendering. It is important to note that the unification-based logic grammar is bi-directional and can be used for text processing as well as for text generation.

3 PENG^{ASP} Interface

The user of the PENG^{ASP} system is guided by a smart text editor that is tightly integrated with the language processor that provides lookahead information about how a CNL sentence can be constructed and completed. The user can define new content words via a pull-down menu during the writing of a specification and requires only minimal linguistic knowledge, since the language processor contains the relevant information that is necessary to automatically classify new content words. The user can also use all accessible anaphoric expressions of an emerging specification via another pull-down menu; these anaphoric expressions are continuously updated

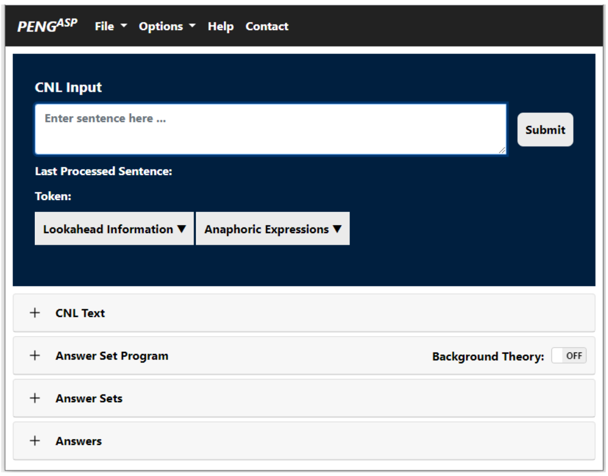

Figure 2: PENG$^{\text{ASP}}$ Interface

during the writing process. The interface (Figure 2) displays the processed text as well as the generated ASP program as output together with the answer set(s) and answers to questions. The user can also ask for a verbalisation of the ASP program which is semantically equivalent to the original specification.

4 Working with PENG$^{\text{ASP}}$

Let us assume the following scenario: a person picks up an object from a first location and drives to second location where she delivers the object; afterwards, she drives back to the first location. Commonsense knowledge tells us that the object moves from the first location to the second location but not back to the first location after delivery. The indirect effect of a person driving from a location to another location is that the object also changes its location. This kind of problem is known as the ramification problem [McCain and Turner, 1995]; it has been shown that the event calculus can be used to reason about indirect effects [Mueller, 2015; Shanahan, 1999].

We can directly express the factual information as well as the required ontological knowledge, the effect axioms, constraints and the relevant questions in CNL.

Here is the factual information for our scenario:

1. The parcel is in Epping on 2020-02-16 at 08:00.
2. Rona is in Epping at 08:50 and picks up the parcel at 09:00.
3. Rona drives from Epping to Eastwood at 09:05 and delivers the parcel at 09:20.
4. Rona drives from Eastwood to Epping at 09:30.

Below is the relevant ontological knowledge for the scenario that can be expressed on the level of the CNL. This means that the language PENG$^{\text{ASP}}$ can also be used as an ontology specification language:

5. Rona is a person.
6. Epping and Eastwood are suburbs.
7. Every suburb is a location.
8. Every parcel is an object.

This ontological knowledge is used – for example – to connect the factual information with the positive and negative effect axioms that have the form of conditional sentences:

9. If a person drives from a location A to a location B at a time point then the person will be in B afterwards.
10. If a person drives from a location A to a location B at a time point then the person will no longer be in A afterwards.
11. If a person is in a location at a time point and an object is in the same location at the same time point and the person picks up the object at that time point then the person will be holding the object afterwards.
12. If a person is holding an object at a time point and the person delivers that object at the same time point then the person will no longer be holding the object afterwards.

We use the consequences of the effect axioms (9) and (10) as constraints to represent the indirect effects of driving from one location to another location:

13. If a person is holding an object at a time point and the same person will be in a suburb after that time point then the object will be in that suburb afterwards.
14. If a person is holding an object at a time point and the same person will no longer be in a suburb after that time point then the object will no longer be in that suburb afterwards.

Additionally, we specify that an object is located in only one location at a time point using a constraint:

15. It is not the case that an object is in a location A at a time point and the same object is in a location B at the same time point and the location A is not the same as the location B.

We can now formulate a question with a time point that has not been mentioned explicitly in the specification:

16. Where is the parcel at 09:25?

Our specification is automatically translated into an ASP program. Figure 3 shows the simplified ASP clauses for the declarative sentence (2), the two conditional sentences (9) and (10), the constraint (15), and the question (16). The declarative sentence (2) leads to a fluent that holds after a specific time point and an event that happens at a later time point. The conditional sentence (9) results in an ASP rule that specifies the conditions under which a given event initiates a particular fluent that holds afterwards. Similarly, the conditional sentence (10) results in an ASP rule that specifies the conditions under which an event terminates a given fluent so that it does no longer hold afterwards. The constraint (15) specifies that two particular fluents cannot hold in a given context. The question (16) leads to a rule with an answer literal in the head of the rule. The answer set system *clingo* will return Eastwood as the correct answer for the question. This is achieved with the help of an ASP-based implementation of the event calculus that has been designed for the CNL.

```
class(1, parcel).
named(2, epping).
named(4, rona).
holds_at(fluent(4, 2, located), 43000).
happens(event(4, 1, pick_up), 43600).

initiated_at(fluent(C, D, located), E) :-
  class(C, person),
  happens(event(C, drive, F), E),
  modifier(G, from, F), class(G, location),
  modifier(D, to, F), class(D, location),
  class(H, time_point),
  data_prop(H, E, date_time).

terminated_at(fluent(I, J, located), K) :-
  class(I, person),
  happens(event(I, drive, L), K),
  modifier(J, from, L), class(J, location),
  modifier(M, to, L), class(M, location),
  class(N, time_point),
  data_prop(N, K, date_time).

:- class(H1, object),
   holds_at(fluent(H1, I1, located), J1),
   class(I1, location),
   class(K1, time_point),
   data_prop(K1, J1, date_time),
   holds_at(fluent(H1, L1, located), J1),
   class(L1, location),
   I1 != L1.

answer(named(M1, N1)) :-
  holds_at(fluent(1, M1, located), 45100),
  named(M1, N1).
```

Figure 3: Excerpt of the resulting ASP Program[1]

In the case of verbalising an ASP program, the language processor of the PENG^{ASP} systems generates semantically equivalent sentences; for example, (17) for (14):

17. If a person is holding an object at a time point and that person will no longer be in a suburb afterwards then that object will no longer be in that suburb afterwards.

We can formally prove that the original specification *S* is semantically equivalent to the verbalisation *S'* by showing via round-tripping that *S* and *S'* produce the same ASP program and the same solutions.

5 Conclusion

The PENG^{ASP} system supports the writing of textual specifications in CNL with the help of a smart text editor and incrementally translates these specifications into executable ASP programs. The system also allows the user to verbalise ASP programs, since the grammar is bi-directional, and makes semantic round-tripping possible between a CNL specification and an ASP program without the loss of information.

[1]Note that timestamps have been abbreviated in the clauses to save space, for example `43000` instead of `1581843000`.

References

[Berry and Kamsties, 2004] Daniel M. Berry and Erik Kamsties. Ambiguity in requirements specification. In *Perspectives on Software Requirements*, volume 753 of *SECS*, pages 7–44. Springer, 2004.

[Bowen and Hinchey, 2006] Jonathan Bowen and Michael Hinchey. Ten commandments of formal methods ... ten years later. *Computer*, 39:40–48, 2006.

[Dalpiaz *et al.*, 2018] Fabiano Dalpiaz, Alessio Ferrari, Xavier Franch, and Cristina Palomares. Natural language processing for requirements engineering: The best is yet to come. *IEEE Software*, 35(5):115–119, 2018.

[Fuchs *et al.*, 2008] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Attempto Controlled English for knowledge representation. In *Reasoning Web*, volume 5224 of *LNCS*, pages 104–124. Springer, 2008.

[Gebser *et al.*, 2019] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Marius Lindauer, Max Ostrowski, Javier Romero, Torsten Schaub, Sven Thiele, and Philipp Wanko. *Potassco User Guide, Version 2.2.0*, 2019.

[Guy and Schwitter, 2017] Stephen Guy and Rolf Schwitter. The PENG^{ASP} system: Architecture, language and authoring tool. *Journal of Language Resources and Evaluation, Controlled Natural Language*, 51:67–92, 2017.

[Kowalski and Sergot, 1986] Robert Kowalski and Marek Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–94, 1986.

[Kuhn, 2014] Tobias Kuhn. A survey and classification of controlled natural languages. *Computational Linguistics*, 40(1):121–170, 2014.

[Lee and Palla, 2012] Joohyung Lee and Ravi Palla. Reformulating temporal action logics in answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 786–792, 2012.

[McCain and Turner, 1995] Norman McCain and Hudson Turner. A causal theory of ramifications and qualifications. pages 1978–1984, 1995.

[Mueller, 2015] Erik T. Mueller. *Commonsense Reasoning: An Event Calculus Based Approach*. Morgan Kaufmann; Second Edition, 2015.

[Parnas, 2010] David L. Parnas. Really rethinking ‘formal methods’. *Computer*, 43:28–34, 2010.

[Schwitter, 2018] Rolf Schwitter. Specifying and verbalising answer set programs in controlled natural language. *Journal of Theory and Practice of Logic Programming*, 18:691–705, 2018.

[Schwitter, 2019] Rolf Schwitter. Augmenting an answer set based controlled natural language with temporal expressions. In *PRICAI 2019: Trends in Artificial Intelligence*, volume 11670 of *LNAI*, pages 500–513. Springer, 2019.

[Shanahan, 1999] Murray Shanahan. The ramification problem in the event calculus. In *IJCAI'99: Proceedings of the 16th International Joint Conference on Artifical Intelligence*, volume 1, pages 140–146, July 1999.