An Approach for Representing Answer Sets in Natural Language

Hans Tompits

Vienna University of Technology
Institute of Information Systems
Knowledge-Based Systems Group

Joint work with Min Fang

Context

➤ We deal with *answer-set programming (ASP)*, an approach for declarative problem solving based on logic programming.

- Fully declarative semantics in terms of *answer sets* (Gelfond & Lifschitz, 1991).

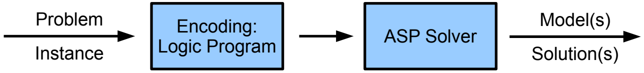
➤ Characteristic feature of ASP:

- The solutions of encoded problems are given by the answer sets of the corresponding logic program.
- Thus, models of logic programs provide the "the answers" to the encoded problems.

☞ This is in contrast to traditional knowledge representation which focuses on *inference relations* and *proofs*.

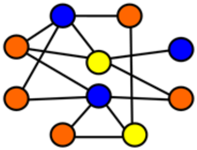
Context (ctd.)

- ASP as a programming paradigm:



- Example:
  - 3-colouring problem (3COL):
    - Given a graph and three colours, assign one colour to each node such that adjacent nodes have different colours.
  - Uniform encoding of 3COL:

```
asgn(N,red) v asgn(N,yellow) v asgn(N,blue) :- node(N).
:- edge(X,Y), asgn(X,C), asgn(Y,C)
```



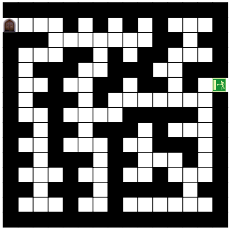
Motivation

- Reading the output of an answer-set solver is often a *tedious and difficult* task.
- Example: Maze generation problem.
  - Benchmark for the second and third *ASP competition*.
  - A maze is a grid where every cell contains either a wall or is empty and there is one entry and one exit.

```
entrance(1,2) exit(5,1) row(1) row(2) row(3) row(4) row(5)
row(6) row(7) ... adjacent(2,11,2,12) adjacent(2,12,2,13)
adjacent(2,13,2,14) adjacent(2,14,2,15) adjacent(3,1,3,2)
... wall(3,10) wall(3,11) wall(3,12) wall(3,13) wall(4,3)
wall(4,7) wall(4,10) wall(4,12) wall(5,3) wall(5,4)
wall(5,5) wall(5,9) wall(5,10) wall(5,12) ... empty(13,14)
empty(14,2) empty(14,3) empty(14,4) empty(14,5) empty(14,6)
empty(14,7) empty(14,8) empty(14,9) empty(14,10)
empty(14,12) empty(14,14)
```

Motivation (ctd.)

- Wanted: a more intuitive representation!
- One possibility: visualising answer sets.
  - This is realised by the `Kara` system (Kloimüllner et al., WLP 2011) and incorporated in `SeaLion` (Busoniu et al., TPLP 2013), an integrated development environment (IDE) for ASP.
  - For the Maze example, such a visualisation may look as follows:



- However, the effectiveness of such visualisations are problem-dependent.
  - Sometimes, other kinds of representations would be more suitable.

Main Contributions

- We propose an approach for a *natural-language representation* of answer sets.
- Basic Idea:
  - Translation is based on annotations in the ASP code which provide meta-information about the atoms in the program.
  - For expressing this meta-information, we
    - introduce a dedicated *controlled natural language* (*CNL*), in which the atom annotations are formulated, and
    - use the ASP annotation language `Lana` (De Vos et al., TPLP 2012) as the underlying annotation framework.
- The approach is also implemented as an Eclipse plug-in for the IDE `SeaLion`.
  - It allows to make further ad-hoc changes to a generated natural-language interpretation and export it, e.g., in PDF format.

Remarks

Possible benefit of a natural-language representation:

- In developing ASP code, one may deal with domain experts unfamiliar with ASP or logical formalisms in general.
- Yet, these experts need to validate the output of ASP encodings.
- ➥ A natural-language representation of answer sets is easier to understand for them.

Controlled Natural Language

- Observation:
  - For realising a method like ours, information other than the pure ASP code is required.
- Indeed, a mechanism which allows to describe "what certain ASP predicates mean" is needed.
  - ➥ This is achieved in terms of *user-specified atom descriptions*.
    - From these, the natural-language-like interpretations for the answer sets are built.
- Note:
  - In order to parse the atom descriptions properly and deduce syntactic information from them, descriptions *conform to a strict syntactic form*, provided by a controlled natural language.

Controlled Natural Language (ctd.)

- A CNL is a constructed language resembling a natural language but being more restrictive concerning
  - lexicon, syntax and/or semantics
  - while preserving most of its natural properties.
- In effect, it is essentially a *formal language* and can therefore (usually) be specified in terms of a grammar.
- Examples of general-purpose (i.e., not restricted to the vocabulary of a specific domain), English-based CNLs are, e.g.,
  - Attempto Controlled English (ACE) (Fuchs et al., 1999) and
  - PENG/PENG Light (Schwitter, 2002).

Controlled Natural Language (ctd.)

Applications of CNL techniques in the context of answer-set programming include:

- `BioQuery-CNL`, developed by Erdem and Yeniterzi (2009), a domain-specific CNL used to express biomedical queries over predefined ontologies.
  - ☞ Queries are translated into answer-set programs using the `BioQuery-ASP` system (Erdem & Öztok, 2015).
- Methods for solving search problems by representing them into a CNL and processing these representations by translations to ASP (Schwitter, 2012, 2013; Guy and Schwitter, 2017).

The Lana Annotation Language

➤ The annotation language `Lana` ("**L**anguage for **AN**notating **A**nswer-set programs"; De Vos et al., 2012) defines a standardised apparatus for specifying meta-information for answer-set programs.

- Reminiscent of Java annotations, with the "`@`" symbol preceding each keyword.
- Annotations are code comments and invisible to the solver.
- Offers support for structuring the code, documentation, testing, and type checking.

➤ An array of different annotations are available in `Lana`, like:

- `@block`, for grouping certain rules together;
- each block can declare predicates using the `@atom` annotation, and define its input and output signature with `@input` and `@output`;
- `@assert`, `@precon`, and `@postcon` define logical conditions for testing purposes.

➥ For our approach, `@atom` and `@output` are mainly relevant.

Example

```
%**
@block WorkTable {
 assign projects and jobs to employees
 @atom employee(E)          E is an employee.
 @atom skill(S)             S is a skill.
 @atom project(P)           P is a project.
 @atom hasSkill(E,S)        Employee E has skill S.
 @atom requiresSkill(P,S) Project P requires skill S.
 @atom works(E,P,S)         Employee E works on project
     P with skill(s) S.
 @atom projectLeader(P,E) Employee E is project
    leader for project P.

 @input employee/1, skill/1, project/1, hasSkill/2,
    requiresSkill/2
 @output works/3, projectLeader/2
*%
% ASP rules feeding the grounder/solver
...
%** } % end of block WorkTable *%
```

A CNL for `Lana` Atom Descriptions

- Note:
  - Our CNL for the `Lana` atom descriptions is neither domain-specific nor general purpose.
- Rather, our restriction lies in the *syntactic structure of its sentences*.
- Indeed, we constrain our grammar to a very small number of rules.
  - This is possible because our CNL is not general-purpose.
    - We know its application context (stated below an `@atom` annotation describing the meaning of this atom)
    - and its purpose (generating human-readable interpretations of answer sets)
    - despite being uninformed about the semantics of its words and the domain of the program.
- However:
  - Imposing a rigid structure on the sentences yields *enough structural information* for the generation of the textual interpretations.

Syntactical Structure of the CNL

- We allow three sentence types in our CNL, depeding on the number of arguments the main verb combines in the sentence (referred to as the *valency* of the verb):
  1. *sentences with an intransitive verb*
     - verb with a subject only (valency 1);
  2. *sentences with a transitive verb*
     - verbs having a subject and a direct object (valency 2) or verbs combining a subject, a direct object, and an indirect object (valency 3);
  3. *sentences with a copula*
     - copula = linking verb, combining the subject with the subject complement (the so-called *predicative*), like the "equals" sign in mathematics.
- We assume a fixed set of *function words*:
  is, are, be, has, have, do, does, must, can, cannot, an, a, A, An, the, The, There, maximally, minimally, at least, at most, or more, or less, or fewer, to, with, of, for, as, by, at, in, on, not.

Examples

- Sentences with an intransitive verb:
  1. Employee **X** works.
  2. Employee **X** does not work.
  3. **N** employees must work on project **P**.
  4. Employee **X** must not work on project **P**.
- Sentences with a transitive verb:
  1. Employee **X** heads project **P**.
  2. Employee **X** does not head project **P**.
  3. Employee **X** heads project **P** with skill **S**.
- Sentences with a copula:
  1. Employee **X** is a project leader.
  2. Employee **X** is project leader for project **P**.
  3. An employee is project leader **L** for project **P**.
  4. There are **N** project leaders on project **P**.

Interpreting Answer Sets

- Based on the `@atom` annotations provided by `Lana` in an ASP code conforming to our CNL, we can generate adapted natural-language sentences.
- Translation uses methods to contract sentences in order to reduce redundant information.
  - Redundancies occur because an atom usually has many instantiations in one answer set.

Interpreting Answer Sets (ctd.)

- For instance, consider the following answer set of the program involving the `Lana` annotations mentioned earlier:

```
    works(boris,p2,planning) works(lisa,p2,planning)
    works(boris,p2,marketing) works(boris,p2,design)
    works(lisa,p2,design) works(lisa,p2,modelling)
    works(sarah,p2,modelling) works(boris,p1,marketing)
    works(boris,p1,design) works(sarah,p1,modelling)
    works(peter,p1,planning) works(hans,p1,modelling)
    works(sarah,p1,coding) works(peter,p1,coding)
    projectLeader(p1,sarah) projectLeader(p2,lisa)
```

- A direct translation, e.g., of the atoms involving Boris yields the following sentences:

```
Employee boris works on project p1 with skill design.
Employee boris works on project p1 with skill marketing.
Employee boris works on project p2 with skill design.
Employee boris works on project p2 with skill marketing.
Employee boris works on project p2 with skill planning.
```

Interpreting Answer Sets (ctd.)

- Noun phrases within the same prepositional phrases can be coordinated:

  ```
  Employee boris works on project p1 with skills design
  and marketing.
  Employee boris works on project p2 with skills design,
  marketing, and planning.
  ```

- Using the *topic-rheme dichotomy*, sentences with a common topic ("Boris") can be coordinated:

  ```
  Employee boris works on project p1 with skills design
  and marketing and on project p2 with skills design,
  marketing, and planning.
  ```

- We implemented our approach in Eclipse, used in connection with the integrated development environment `SeaLion` for ASP.

Conclusion

- We presented a CNL approach for generating interpretations for answer sets in which the user can specify meta-information about the predicates used in an answer-set program.
  - The descriptions in the `Lana` annotations are restricted accorded to the CNL we defined.
- Approaches complementary to our work include:
  - methods which provide *justifications* for the inclusion or non-inclusion of ground atoms in answer sets (Pontelli et al., 2009);
  - methods giving explanations why a program has no answer sets at all (Syrjänen, 2006).
- An issue for future work is to consider the question to translate answer-set programs themselves into natural language.
  - However, this task requires a more dedicated syntactic and semantic analysis of user-specified sentences.