Uhura: An Authoring Tool for Specifying Answer-Set Programs Using Controlled Natural Language

Tobias Kain(✉) and Hans Tompits

Institute of Logic and Computation, Knowledge-Based Systems Group, Technische Universität Wien, Favoritenstraße 9-11, 1040 Vienna, Austria
{kain,tompits}@kr.tuwien.ac.at

Abstract. In this paper, we present the tool Uhura for developing answer-set programs by means of specifying problem descriptions in a controlled natural language which then are translated into answer-set programming (ASP) rules. The tool is aimed for supporting users not familiar with answer-set programming—or logic-based approaches in general—for developing programs. Uhura is based on a new controlled natural language called L^{U}, which is in turn an adaption of $\mathsf{PENG}^{\mathrm{ASP}}$, a controlled natural language employed in the PENG ASP system, developed by Guy and Schwitter, for solving computational problems by translating $\mathsf{PENG}^{\mathrm{ASP}}$ statements into answer-set programs. In contrast to $\mathsf{PENG}^{\mathrm{ASP}}$, L^{U} allows for a more natural translation into ASP rules and provides also a broader set of pre-defined sentence patterns. Uhura is implemented in Java and employs DLV as backend answer-set solver.

Keywords: Answer-set programming · Program development · Controlled natural language

1 Introduction

In the past couple of years, *answer-set programming* (ASP), i.e., logic programming under the answer-set semantics [15], has evolved into a viable approach for declarative problem solving. Although applications in many diverse fields using ASP have been developed, ASP is rarely used by developers who do not work in an academic environment. This is attributable, among other things, to a lack of support tools designed for ASP novices, although work on integrated development environments for ASP [2,9], as well as methods for debugging and testing answer-set programs are available [5,10,16,20,25]. Especially those who have never before used any logical programming languages have a hard time getting acquainted with ASP.

In this paper, we present a tool which addresses this issue. Our tool, Uhura, supports ASP novices as well as professional ASP developers in developing answer-set programs by providing an editor in which a problem description can be specified by means of a *controlled natural language* (CNL) which is then translated into an answer-set program which is displayed vis-à-vis the CNL specification. The obtained ASP code can be executed using the solver DLV [23] and the computed answer sets can accordingly be displayed in a tab of Uhura as well. Uhura employs the ASP-Core-2 [3] syntax, which is supported by DLV as well as other ASP solvers. The name of our tool derives from the fictional Star Trek character Lieutenant Nyota Uhura, who serves as communication officer at the Starship Enterprise, because our tool and Lt. Uhura have somewhat similar tasks, viz. acting as a facilitator between a familiar and an unfamiliar language.

The CNL employed in Uhura, called L^{U}, is a novel CNL which is based on $\mathsf{PENG}^{\mathrm{ASP}}$ [30], a CNL whose primary purpose is to be a CNL which can unambiguously be translated into ASP. However, compared to $\mathsf{PENG}^{\mathrm{ASP}}$, the translation of L^{U} sentences into ASP is more direct and L^{U} offers a broader set of predefined sentence patterns. Following Kuhn [22], 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 is often used as a bridge between a highly ambiguous natural language and a less human-readable formal language.

The idea of building a tool for translating a problem description expressed in a (controlled) natural language into an answer-set program is not a new one. For example, the system based on $\mathsf{PENG}^{\mathrm{ASP}}$ [17,18], as well as LOGICIA [24] and BioQuery-ASP [6,7] are systems that translate a CNL into ASP in order to achieve various goals. Furthermore, there is also an approach [8] to translate answer sets in more human-readable form based on a CNL using the ASP annotation language Lana [4].

However, what makes our approach unique is that the aim of Uhura is not only to translate CNL sentences into ASP rules but to perform the translation in such a way that the user can learn how to write answer-set programs. To increase the learning progress, we decided to design L^{U} in such a way that correspondence between the CNL sentences and the ASP rules is clearly evident. While most of the other available tools primarily focus on solving the described problem, the goal of translating a CNL problem description into an intuitive-to-read answer-set program comes at the expense of expressibility. Moreover, our tool does not aim for an optimised translation (concerning time resources). Uhura can also help to improve the communication between those people who provide the required knowledge to solve a certain problem (i.e., the domain experts) and those who put the knowledge into code (i.e., the knowledge engineers), since the domain expert can read and understand the CNL sentences the knowledge engineer has provided.

Our paper is organised as follows: Sect. 2 provides background information about answer-set programming and $\mathsf{PENG}^{\mathrm{ASP}}$, the CNL our language is based upon. Section 3 gives an overview of the system Uhura, including a description of a typical workflow in Uhura, details about L^{U}, the CNL underlying our system, and some implementational details. The paper concludes with Sect. 4, containing a discussion on related approaches and future work.

2 Preliminaries

2.1 Answer-Set Semantics

We assume the reader familiar with the basic elements of answer-set programming (ASP) [1,12,13]. To briefly recapitulate the relevant elements of ASP, by an *answer-set program* (or *program* for short) we understand a set of rules of form

$$a_1 \vee \cdots \vee a_m \;:-\; b_1, \ldots, b_k, \text{not } b_{k+1}, \ldots, \text{not } b_n, \tag{1}$$

where $a_1, \ldots, a_m$ and $b_1, \ldots, b_n$ are *literals* over a first-order vocabulary, i.e., atoms possibly preceded by the *strong negation* symbol "$-$", "$\vee$" denotes *disjunction*, and "not" stands for *default negation*. The intuitive meaning of rule (1) is that if $b_1, \ldots, b_k$ are derivable and $b_{k+1}, \ldots, b_n$ are *not* derivable, then at least one of $a_1, \ldots, a_m$ is asserted.

We refer to $a_1 \vee \cdots \vee a_m$ as the *head* of (1) and $b_1, \ldots, b_k, \mathit{not}\ b_{k+1}, \ldots, \mathit{not}\ b_n$ as the *body*. The latter in turn is subdivided into the *positive* and the *negative* body, given by $b_1, \ldots, b_k$ and $\mathit{not}\ b_{k+1}, \ldots, \mathit{not}\ b_n$, respectively.

Both the head and the body of a rule might be empty. In the former case, the resulting rule is referred to as a *fact*, whilst in the latter case, the rule is called a *constraint*.

The semantics of a program, P, is given in terms of *answer sets*, which are defined as minimal models of the usual *Gelfond-Lifschitz reduct* [14,15]. Prominent solvers for computing answer sets are, e.g., clasp [26] and DLV [23]. We also employ the aggregate function $\#\mathit{count}$ as used in DLV for counting elements satisfying certain properties.

2.2 The Controlled Natural Language PENG$^{\text{ASP}}$

The controlled natural language (CNL) underlying Uhura, L$^{\text{U}}$, is based on the CNL PENG$^{\text{ASP}}$ [30], which itself is based on PENG Light [29] which in turn is the

Table 1. Word categories of PENG$^{\text{ASP}}$.

Word category	Example
PNoun (proper noun)	Vienna, Roberta, Spock
CNoun (common noun)	person, animal, university
Adjective	female, male, mortal
Verb	do, work, play
OrdNumber (order number)	first, second, third
CardRest (cardinality restriction)	exactly, at most, at least

Table 2. Simple sentences of PENG$^{\text{ASP}}$.

Pattern	Example
PNoun is a *CNoun*.	Roberta is a person.
PNoun is *Adjective*.	Roberta is female.
There is a *CNoun*.	There is a job.
A *CNoun* is *Adjective*.	A person is female.
A *OrdNumber CNoun* is a *CNoun* of a *OrdNumber CNoun*.	A first person is a husband of a second person.
A *CNoun Verb* a *CNoun* as *PNoun*.	A person holds a job as nurse.
CardRest CNoun Verb a *CNoun*.	Exactly one person holds a job.
A *CNoun Verb CardRest CNoun*.	A person holds exactly two jobs.

successor of PENG [28], a computer-oriented controlled natural language similar to Attempto Controlled English [11]. The goal of PENG Light is to provide a CNL that can be translated unambiguously into first-order logic. In contrast to its predecessor, PENG Light can be processed by a bidirectional grammar, meaning that first-order translations of sentences expressed in PENG Light can be used to create answers to these sentences. In what follows, we give a brief overview about PENG$^{\text{ASP}}$.

Like PENG Light, PENG$^{\text{ASP}}$ differentiates between *simple* and *complex sentences*. Simple sentences are built on six word categories, which are given in Table 1 along with corresponding examples. A simple sentence, then, is one of the eight patterns depicted in Table 2 (similar to Table 1, each pattern is adjoined by a corresponding example). Depending on the application, more sentence patterns and word categories can be added.

Note that not all simple sentences listed in Table 2 can be translated immediately into ASP. Indeed, only the first two sentences of Table 2 are factual statements, meaning that they can be translated into an ASP rule. All the other simple sentences contained in Table 2 can only be used as part of a complex sentence, which are defined in PENG$^{\text{ASP}}$ as follows:

$$\text{If } \mathit{SimpleSentence}1 \ \{\text{and } \mathit{SimpleSentenceN}\} \text{ then } \mathit{SimpleSentenceM}. \quad (2)$$

$$\text{Exclude that } \mathit{SimpleSentence}1 \ \{\text{and that } \mathit{SimpleSentenceN}\}. \quad (3)$$

Here, "$\{\ldots\}$" means that the argument surrounded by the curly brackets can be used 0 to n times, where $n \in \mathbb{N}$.

A sentence that fulfills pattern (2) corresponds to an ASP rule which has a non-empty head and a non-empty body. On the other hand, a sentence that matches pattern (3) corresponds to an ASP constraint.

For illustration of the translation of PENG$^{\text{ASP}}$ into ASP, consider the following two complex sentences and their respective ASP representations (for a detailed description of the translation process, cf. Schwitter [30]):

– "If a person holds a job as teacher then the person is educated":

$$\mathit{educated}(X)\ :-\ \mathit{person}(X), \mathit{job}(\mathit{teacher}), \mathit{hold}(X, \mathit{teacher}). \tag{4}$$

– "Exclude that Roberta is female and that Roberta is a man":

$$:-\ \mathit{female}(\mathit{roberta}), \mathit{man}(\mathit{roberta}). \tag{5}$$

Sentence (4) shows that PENG$^{\text{ASP}}$ allows that the simple sentences which are part of a complex sentence can depend on each other. Therefore, it is not possible in general to translate the simple sentences of a complex sentence individually.

3 The System Uhura

Uhura is a tool for translating a problem definition expressed in the CNL L$^{\text{U}}$ (to be detailed below) into an answer-set program. The development of Uhura was driven by the motivation to provide a system which supports ASP novices doing their first steps in answer-set programming. By specifying a problem in a language similar to English, but influenced by the syntax of ASP, users can learn how to express domain knowledge declaratively. As well, Uhura makes it possible to directly see how domain knowledge is translated into ASP rules, by observing the generated ASP rules from the specified CNL sentences. Indeed, Uhura converts CNL sentences in such a way that the user recognises which sentence is translated into which ASP rule(s). Our aim was also to keep the ASP rules as natural as possible, meaning that the resulting answer-set program should be easy to understand by developers unexperienced in ASP.

Uhura, as well as its source code, is available for download at: https://github.com/TobiasKain/uhura.

In what follows, we first describe a typical workflow when using Uhura. Afterwards, we give details about L$^{\text{U}}$, the CNL underlying Uhura. Finally, we briefly discuss aspects of the implementation of Uhura.

3.1 Workflow in Uhura

The Jobs Puzzle. To demonstrate how a typical workflow in Uhura looks like and how the system works, we use the following puzzle:

1. There are four people: Roberta, Thelma, Steve, and Pete.
2. Among them, they hold eight different jobs.
3. Each holds exactly two jobs.
4. The jobs are chef, guard, nurse, clerk, police officer (gender not implied), teacher, actor, and boxer.
5. The job of nurse is held by a male.

6. The husband of the chef is the clerk.
7. Roberta is not a boxer.
8. Pete has no education past the ninth grade.
9. Roberta, the chef, and the police officer went golfing together.

This puzzle is the so-called *jobs puzzle*, which was first introduced in 1984 along with other puzzles designed for automated reasoning [31].

The goal of this puzzle is to find out which person holds which jobs. Clearly, the solution to this puzzle cannot be found only by analysing the explicit information stated in the puzzle. Instead, we have to extract the implicit information of the puzzle and use this knowledge together with the explicit information to solve the puzzle.

Controlled Natural Language Problem Description. In order to solve the jobs puzzle using Uhura, the user has to phrase the implicit and explicit knowledge of the puzzle in the controlled natural language L^{U}, which is similar to $\mathsf{PENG}^{\mathrm{ASP}}$. However, compared to $\mathsf{PENG}^{\mathrm{ASP}}$, the controlled natural language we are working with comprises slightly different sentence patterns from those in $\mathsf{PENG}^{\mathrm{ASP}}$ as well as additional ones. Furthermore, Uhura also allows to define individual sentence patterns. We give more details about L^{U} below; a full description of the language can be found in the thesis of the first author [21].

The first sentence of the jobs puzzle states that our problem domain contains four different people. We express this fact by using the pattern

$$PNoun \text{ is a } CNoun.$$

In our case, we instantiate *PNoun*, referring to a proper noun, with *Roberta*, *Thelma*, *Steve*, and *Pete*, and *CNoun* (common noun) with *person*, obtaining the following sentences:

Roberta is a person. Thelma is a person. Steve is a person. Pete is a person.

Furthermore, we have to express that a person cannot be both male and female simultaneously. This means we drop all solutions which contain both genders for some person. We do this by using the following pattern:

$$\text{Exclude that } SimpleSentence \ \{\text{and that } SimpleSentence\}.$$

Since we want that this constraint is applied to every person in our problem domain, we use a variable instead of proper nouns.

$$\text{Exclude that person } X \text{ is male and that person } X \text{ is female.}$$

Next, we guess all the possible solution candidates. In order to do that, we use the following sentence pattern:

$$\text{If } SimpleSentence \ \{\text{and } SimpleSentence\} \text{ then } SimpleSentence.$$

To guess whether a person holds a specific job or not, we use a disjunctive sentence:

> If there is a person X and there is a job Y then person X holds job Y or person X does not hold job Y.

The second sentence of the jobs puzzle tells us that our problem domain includes eight different jobs and that every job is held by exactly one person. To formulate this, we use the following pattern as part of a constraint:

$$CNoun\ Variable\ Verb \text{ (more/less) than } Number\ CNoun\ Variable.$$

Since we want to express that a job is held by *exactly* one person, we have to use this pattern twice:

> Exclude that there is a job Y and that person X holds more than one job Y.
> Exclude that there is a job Y and that person X holds less than one job Y.

The third sentence of the job puzzle states that each person holds exactly two jobs. As before, we use the cardinality pattern twice to formulate this statement:

> Exclude that there is a person X and that person X holds more than two jobs Y.
> Exclude that there is a person X and that person X holds less than two jobs Y.

The next sentence of the jobs puzzle enumerates all jobs our problem domain refers to. We formulate these facts using the same pattern we applied to phrase the first sentence:

> Chef is a job. Guard is a job. Nurse is a job. Clerk is a job.
> Police officer is a job. Teacher is a job. Actor is a job. Boxer is a job.

Chef, guard, nurse, clerk, police officer, teacher, and boxer are gender neutral jobs, meaning that they can be held by both genders. However, the job of an actor can only be done by a male, since a female actor is called an actress.

Furthermore, due to the fifth sentence of the jobs puzzle, we know that the job of a nurse is held by a male:

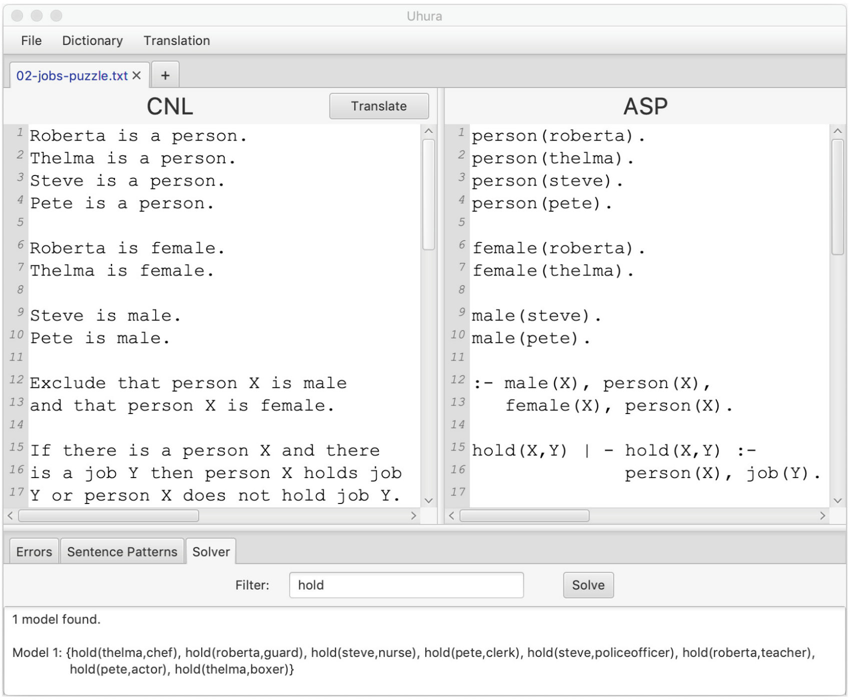
> If a person X holds a job as actor then person X is male.
> If a person X holds a job as nurse then person X is male.

According to the sixth sentence of the jobs puzzle, the husband of the chef is the clerk:

> If a person X holds a job as chef and a person Y holds a job as clerk then person Y is a husband of person X.

The previous sentence implies that the clerk is male since he is the husband of another person and that the chef is female since she has a husband:

> If a person X is a husband of a person Y then person X is male.
> If a person X is a husband of a person Y then person Y is female.



```
Roberta is a person.
Thelma is a person.
Steve is a person.
Pete is a person.

Roberta is female.
Thelma is female.

Steve is male.
Pete is male.

Exclude that person X is male
and that person X is female.

If there is a person X and there
is a job Y then person X holds job
Y or person X does not hold job Y.
```

```
person(roberta).
person(thelma).
person(steve).
person(pete).

female(roberta).
female(thelma).

male(steve).
male(pete).

:- male(X), person(X),
   female(X), person(X).

hold(X,Y) | - hold(X,Y) :-
              person(X), job(Y).
```

Filter: hold

```
1 model found.

Model 1: {hold(thelma,chef), hold(roberta,guard), hold(steve,nurse), hold(pete,clerk), hold(steve,policeofficer), hold(roberta,teacher),
         hold(pete,actor), hold(thelma,boxer)}
```

Fig. 1. The user interface of Uhura for the jobs puzzle.

Since the seventh sentence of the jobs puzzle states that Roberta is not a boxer, we exclude all solutions which claim that Roberta is a boxer:

Exclude that Roberta holds a job as boxer.

The eighth sentence of the jobs puzzle specifies that Pete is not educated past the ninth grade. Therefore, we drop all solutions that assert that Pete is educated past the ninth grade:[1]

Exclude that Pete is educated.

Furthermore, we can assume that a person has to be educated past the ninth grade to hold a job as a nurse, police officer, or teacher:

If a person X holds a job as nurse then person X is educated.
If a person X holds a job as police officer then person X is educated.
If a person X holds a job as teacher then person X is educated.

The last sentence of the jobs puzzle contains the implicit information that Roberta is neither a chef nor a police officer. Furthermore, this sentence also

[1] Here, and henceforth, in CNL sentences, we simply write "educated" to refer to the property of being educated past the ninth grade.

```
person(roberta).
person(thelma).
person(steve).
person(pete).

female(roberta).
female(thelma).
male(steve).
male(pete).

:- male(X), person(X), female(X), person(X).

hold(X,Y) | -hold(X,Y) :- person(X), job(Y).

:- job(Y), #count{X : hold(X,Y)} > 1.
:- job(Y), #count{X : hold(X,Y)} < 1.
:- person(X), #count{Y : hold(X,Y)} > 2.
:- person(X), #count{Y : hold(X,Y)} < 2.

job(chef).
job(guard).
job(nurse).
job(clerk).
job(policeofficer).
job(teacher).
job(actor).
job(boxer).

male(X) :- hold(X,actor), person(X), job(actor).
male(X) :- hold(X,nurse), person(X), job(nurse).
husband(Y,X) :- hold(X,chef), person(X), job(chef),
                hold(Y,clerk), person(Y), job(clerk).
male(X) :- husband(X,Y), person(X), person(Y).
female(Y) :- husband(X,Y), person(X), person(Y).
educated(X) :- hold(X,nurse), person(X), job(nurse).
educated(X) :- hold(X,policeofficer), person(X),
               job(policeofficer).
educated(X) :- hold(X,teacher), person(X), job(teacher).

:- hold(roberta,boxer), job(boxer).
:- educated(pete).
:- hold(roberta,chef), job(chef).
:- hold(roberta,policeofficer), job(policeofficer).
:- hold(X,chef), person(X), job(chef), hold(X,policeofficer),
   person(X), job(policeofficer).
```

Fig. 2. ASP translation of the jobs puzzle L^{U} specification.

implies that the person who works as a chef and the person who works as a police officer are two different individuals:

> Exclude that Roberta holds a job as chef.
> Exclude that Roberta holds a job as police officer.
> Exclude that a person X holds a job as chef and that person X holds a job as police officer.

Composing the CNL Problem Description Using Uhura. As the previous discussion illustrates, the user has to work precisely to extract all the implicit information contained in the puzzle. Especially for inexperienced users, the task of extracting such implicit information is arguably one of the most demanding tasks in this context. Now, to support the user as good as possible during the process of composing the problem description, our tool provides a separate text field where the user writes down the domain knowledge (see Fig. 1). Alternatively, our tool also allows importing text files composed in a different text editor.

Since remembering all the different sentence patterns is unrewarding, the user can look them up by switching in the lower tab pane to the tab called *Sentence Patterns*.

In case the user enters a sentence that does not match any pattern, Uhura responds with an error message that explains where exactly in the sentence an error occurred. For example, assume the user types in the sentence "Roberta is not a lovely". Then, the system responds with the following error message, which tells the user which sentence pattern was detected and which word caused the error:

```
Error in sentence "Roberta is not a lovely.":
     "lovely" is not a common noun.
     (detected sentence-pattern: 'PNoun is [not] a CNoun.')
```

ASP Translation and Solving. Once the user clicks the *Translation* button, the system starts translating the entered CNL problem description into ASP rules. The answer-set program resulting from translating the CNL problem description for the jobs puzzle given above is depicted in Fig. 2. Note that the syntax used by Uhura is the ASP-Core-2 syntax, which is supported by DLV as well as other ASP solvers.

As can be seen in Fig. 1, the result of the translation process is displayed in the text editor next to the CNL problem description. Also, the (single) solution of the problem is shown in the *Solver* tab of the editor, which is situated in the bottom bar. Since the goal of the jobs puzzle is to find out which person holds which job, we are only interested in the predicate *hold* and therefore the resulting model is filtered by this predicate.

3.2 The Controlled Natural Language L^{U} of Uhura

We now describe the CNL of Uhura, L^{U}, in more detail.

Table 3. Word categories supported by L^{U}.

Word category	Example
PNoun (Proper Noun)	Vienna, Roberta, Spock
CNoun (Common Noun)	person, animal, university
Adjective	female, male, mortal
Verb	do, work, play
Variable	X, Y, Z
Preposition	at, to, in
Number	one, two, three

Table 4. Examples of default sentences which are supported by L^{U}.

Sentence	Translation
Birds normally fly.	$\mathit{fly}(X) \ {:}{-}\ \mathit{bird}(X),\ \mathit{not}\ {-}\mathit{fly}(X)$
Dogs normally do not fly.	${-}\mathit{fly}(X) \ {:}{-}\ \mathit{dog}(X),\ \mathit{not}\ \mathit{fly}(X)$
Birds normally are beautiful.	$\mathit{beautiful}(X) \ {:}{-}\ \mathit{bird}(X),\ \mathit{not}\ {-}\mathit{beautiful}(X)$
Birds normally are not ugly.	${-}\mathit{ugly}(X) \ {:}{-}\ \mathit{bird}(X),\ \mathit{not}\ \mathit{ugly}(X)$
Students normally are afraid of exams.	$\mathit{afraid\_of}(X, \mathit{exam}) \ {:}{-}\ \mathit{student}(X),$ $\mathit{not}\ {-}\mathit{afraid\_of}(X, \mathit{exam})$
Students normally are not afraid of homework.	${-}\mathit{afraid\_of}(X, \mathit{homework}) \ {:}{-}\ \mathit{student}(X),$ $\mathit{not}\ \mathit{afraid\_of}(X, \mathit{homework})$

To begin with, the word categories of L^{U}, which are slightly different from $\mathsf{PENG}^{\mathrm{ASP}}$, are given in Table 3. Furthermore, L^{U} comprises, besides simple and complex sentences, also some other groups of sentence patterns. The set of sentence patterns supported by L^{U} comprises:

- *simple sentences* (divided into *factual simple sentences* and *non-factual simple sentences*),
- *complex sentences*,
- *default sentences*, and
- *categorical propositions*.

Another difference between L^{U} and $\mathsf{PENG}^{\mathrm{ASP}}$ is that L^{U} does not allow cross references between the simple sentences used in a complex sentence. Therefore, simple sentences can be translated individually, which makes it easier to translate sentences expressed in L^{U}. A full listing of all sentence patterns of L^{U}, along with their ASP translations, can be found in the thesis of the first author [21]; here, we describe only some key aspects.

Simple Sentences. The group of simple sentences is divided into *factual simple sentences* and *non-factual simple sentences*. The difference between those two

Table 5. Examples of categorical propositions which are supported by L^{U}.

Sentence	Translation
All humans are mortal.	$mortal(X)\ :-\ human(X)$
No humans are perfect.	$-perfect(X)\ :-\ human(X)$
Some humans are bad.	$:-\ \#count\{X : human(X), bad(X)\} = 0$
Some humans are not bad.	$:-\ \#count\{X : human(X), not\ bad(X)\} = 0$

sentence types is that factual simple sentences can be directly transformed into an ASP rule, where, on the other hand, non-factual simple sentences can only be used as part of complex sentences.

For example, the factual simple sentence "Roberta is a person" can be directly transformed into the fact:

$$person(roberta)\ :-\ .$$

On the other hand, a non-factual simple sentence like "X is a person" cannot be directly transformed into an meaningful ASP rule. However, this non-factual simple sentence can be used, e.g., as part of a conditional sentence like "If X is a person then X is mortal", which can be translated into the ASP rule:

$$mortal(X)\ :-\ person(X).$$

The reason why L^{U} separates the group of simple sentences is because of the following complex sentence, which also serves as a factual simple sentence:

$$FactualSimpleSentence\ \{\text{ or } FactualSimpleSentence\}. \tag{6}$$

Complex Sentences. L^{U} supports the same complex sentences that are defined by $\mathsf{PENG}^{\mathrm{ASP}}$, as well as complex sentences of form (6). A sentence of the latter form, like, e.g., "Roberta is a person or Roberta is a dog", can either be used as a factual simple sentence, which leads to the translation

$$person(roberta) \vee dog(roberta)\ :-\ ,$$

or as part of a complex sentence, like, e.g.:

If Roberta is mortal then Roberta is a person or Roberta is a dog.

This, in turn, corresponds to the ASP rule

$$person(roberta) \vee dog(roberta)\ :-\ mortal(X).$$

Default Sentences. *Default sentences* are used to describe circumstances that hold *typically*. L^{U} supports the following three default sentences as well as their negations:

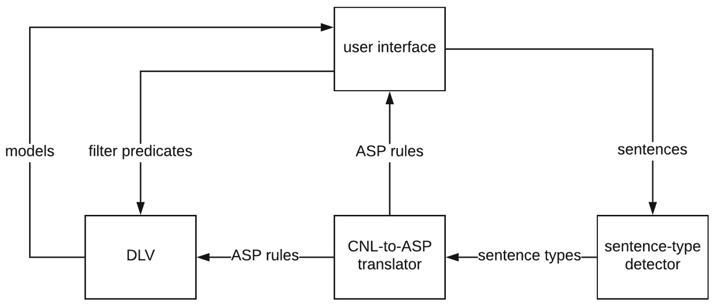


Fig. 3. The system architecture of Uhura.

- $CNoun$ normally $Verb$.
- $CNoun$ normally not $Verb$.
- $CNoun$ normally are *Adjective*.
- $CNoun$ normally are not *Adjective*.
- $CNoun$ normally are *Adjective Preposition* $CNoun$.
- $CNoun$ normally are not *Adjective Preposition* $CNoun$.

Table 4 shows an example sentence for each sentence pattern as well as its ASP translation.

Categorical Propositions. L^{U} also supports the four categorical propositions as used in Aristotle's system of syllogisms:

- All A are B.
- No A are B.
- Some A are B.
- Some A are not B.

An A has to be replaced by a *CNoun* and a B either by a *CNoun* or an *Adjective*. Table 5 illustrates each categorical proposition along with its translation. Note that the last two propositions (referred to as *particular statements* in the terminology of the syllogistic) employ the count aggregate of DLV.

3.3 Implementation

Uhura is implemented in Java. One of the main reasons why we chose Java is the availability of DLVWrapper [27], which is a Java library that provides all the functionalities of DLV. Another reason why we decided to implement our tool using Java is that we want to keep the option to integrate our tool into SeaLion [2], which is an integrated development environment for ASP.

Basically, Uhura consists of four different components:

- the user interface,

– a sentence-type detector,
– the CNL-to-ASP translator, and
– DLV (as underlying ASP solver).

Figure 3 shows how these components are connected to each other.

In what follows, we provide brief descriptions of these components.

User Interface. To design and create the user interface (UI) of Uhura, we used JavaFX. Since one of our goals was to provide an intuitive graphical user interface, we decided to design the UI of Uhura similar to the user interfaces of popular IDEs, like, e.g., IntelliJ, Eclipse, or VisualStudio.

Sentence-Type Detector. Once the user has typed in a sentence and clicked the *Translate* button, the sentence-type detector tries to find out which sentence pattern the entered sentence fulfills. Therefore, the sentence-type detector contains for each sentence pattern a regular expression ("regex") pattern. For example, the regex pattern of the sentence pattern "*PNoun* is a *CNoun*" is defined as follows:

```
.* is(n't | not | )(a|an) .*\\.$.
```

Note that also the negated sentence pattern ("*PNoun* is not a *CNoun*.") matches this regex pattern. Furthermore, note that there is no space between the two alternations because the options of the first alternation (i.e., "`(n't | not | )`") end with a space.

To find out of which type the entered sentence is, the sentence-type detector sequentially checks if the sentence matches one of the regex patterns. In case a match is found, the sentence-type detector initiates the CNL-to-ASP translation of the sentence.

CNL-to-ASP Translator. The CNL-to-ASP translator is the heart of Uhura. It is responsible for translating the CNL sentences entered by the user into ASP rules. The basic idea of the CNL-to-ASP translator is to filter out the key words of a sentence and remove those words that are not used to put together the ASP rule. The CNL-to-ASP translator also checks if the words are of the correct category (e.g., *PNoun*, *CNoun*, adjective, verb, etc.). To do so, the translator makes use of the Stanford Parser[2], a natural-language parser that uses statistical data to parse sentences. In case a word is of a different type than expected, an exception is thrown, which tells the sentence-type detector to try if the sentence matches one of the remaining sentence patterns.

4 Conclusion and Discussion

In this paper, we presented the tool Uhura for translating sentences expressed in a controlled natural language into an answer-set program. Rather than aiming for

[2] https://nlp.stanford.edu/software/lex-parser.html.

a maximally expressive CNL and optimised ASP encodings, our goal was to have a system which helps users inexperienced in ASP or logical formalisms in general for developing programs by specifying problems in a language which is close to a natural one. Therefore, we introduced a new CNL, called L^{U}, which allows the user to unambiguously specify a problem. Furthermore, L^{U} is designed in such a way that the correspondence between the CNL sentences and the resulting ASP rules is clearly evident.

The idea of building a tool for translating a problem description expressed in a (controlled) natural language into an answer-set program is not a new one. For example, Schwitter and Guy presented a web-based predictive editor for $\mathsf{PENG}^{\mathrm{ASP}}$ [17,18]. The editor suggests for every word that is typed in a selection of words from which the user can choose. Thus, sentences entered by the user are always valid $\mathsf{PENG}^{\mathrm{ASP}}$ sentences. Compared to Uhura, this system can handle more complicated sentences but its focus lies more on problem solving rather than on obtaining natural-looking ASP encodings.

Furthermore, Baral and Mitra [24] developed the system LOGICIA for solving logic grid puzzles. To solve those puzzles, LOGICIA translates the puzzles, described in natural language, into ASP which is then solved by an answer-set solver. The system automatically learns how to translate the information given by the puzzle description. Obviously, the aim of LOGICIA differs from the goal of Uhura quite strongly. While our tool focuses on generating answer-set programs that can be easily understood by developers inexperienced in ASP, LOGICIA deals with translating sentences expressed in natural language. To do so, this tool represents answer-set rules in a more complicated and unintuitive way.

Other approaches in the context of ASP using CNLs is the BioQuery-ASP system [6,7] for expressing biomedical queries over predefined ontologies and translating them into ASP. Moreover, Min and Tompits [8] provided an approach to translate answer sets into natural language which is also based on a CNL using the annotation language Lana [4] for ASP.

As regards future work, one possibility would be to incorporate the approach mentioned last into Uhura such that not only the programs can be specified in a natural-language-looking way, but also the output of the programs, i.e., the answer sets.

Another possible enhancement of Uhura would be to integrate the system into SeaLion [2], which is an integrated development environment for answer-set programming. In this case, the only unit that has to be changed is the user interface. Furthermore, this integration would also allow providing a second ASP solver to solve the answer-set programs generated by Uhura since SeaLion supports two answer-set solvers, viz. DLV and clingo.

Another promising future work would be to use Uhura for specifying test cases for software projects. In particular, based on previous work [6,19], in which ASP is used to specify sequence-covering arrays, the idea would be to use Uhura to allow the user to specify a sequence-covering array in the controlled natural language offered by our system and then to translate this definition into ASP.

References

1. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, Cambridge (2003)
2. Busoniu, P., Oetsch, J., Pührer, J., Skocovsky, P., Tompits, H.: SeaLion: An Eclipse-based IDE for answer-set programming with advanced debugging support. Theory Pract. Log. Program. **13**(4–5), 657–673 (2013)
3. Calimeri, F., et al.: ASP-Core-2: Input language format. ASP Standardization Working Group (2012)
4. De Vos, M., Kisa, D.G., Oetsch, J., Pührer, J., Tompits, H.: Annotating answer-set programs in Lana. Theory Pract. Log. Program. **12**(4–5), 619–637 (2012)
5. Dodaro, C., Gasteiger, P., Musitsch, B., Ricca, F., Shchekotykhin, K.: Interactive debugging of non-ground ASP programs. In: Calimeri, F., Ianni, G., Truszczynski, M. (eds.) LPNMR 2015. LNCS (LNAI), vol. 9345, pp. 279–293. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23264-5_24
6. Erdem, E., Erdogan, H., Öztok, U.: BioQuery-ASP: Querying biomedical ontologies using answer set programming. In: Proceedings of the 5th International RuleML2011@BRF Challenge. CEUR Workshop Proceedings, vol. 799. CEUR-WS.org (2011)
7. Erdem, E., Öztok, U.: Generating explanations for biomedical queries. Theory Pract. Log. Program. **15**(1), 35–78 (2015)
8. Fang, M., Tompits, H.: An approach for representing answer sets in natural language. In: Seipel, D., Hanus, M., Abreu, S. (eds.) WFLP/WLP/INAP 2017. LNCS (LNAI), vol. 10997, pp. 115–131. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00801-7_8
9. Febbraro, O., Reale, K., Ricca, F.: ASPIDE: Integrated development environment for answer set programming. In: Delgrande, J.P., Faber, W. (eds.) LPNMR 2011. LNCS (LNAI), vol. 6645, pp. 317–330. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20895-9_37
10. Febbraro, O., Leone, N., Reale, K., Ricca, F.: Unit testing in *ASPIDE*. In: Tompits, H., et al. (eds.) INAP/WLP 2011. LNCS (LNAI), vol. 7773, pp. 345–364. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41524-1_21
11. Fuchs, N.E., Schwitter, R.: Attempto Controlled English (ACE). In: Proceedings of the First International Workshop on Controlled Language Applications (CLAW 1996). University of Leuven (1996)
12. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers (2012)
13. Gelfond, M., Kahl, Y.: Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach. Cambridge University Press, Cambridge (2014)
14. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proceedings of the 5th International Conference and Symposium on Logic Programming (ICLP/SLP), pp. 1070–1080. MIT Press (1988)
15. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Gener. Comput. **9**, 365–385 (1991)
16. Greßler, A., Oetsch, J., Tompits, H.: Harvey: A system for random testing in ASP. In: Balduccini, M., Janhunen, T. (eds.) LPNMR 2017. LNCS (LNAI), vol. 10377, pp. 229–235. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61660-5_21

17. Guy, S., Schwitter, R.: Architecture of a web-based predictive editor for controlled natural language processing. In: Davis, B., Kaljurand, K., Kuhn, T. (eds.) CNL 2014. LNCS (LNAI), vol. 8625, pp. 167–178. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10223-8_16
18. Guy, S.C., Schwitter, R.: The PENG$^{\text{ASP}}$ system: Architecture, language and authoring tool. Lang. Resour. Eval. **51**(1), 67–92 (2017)
19. Irlinger, M.: Combinatorial testing using answer-set programming. Bachelor's thesis, Technische Universität Wien, Institute of Information Systems, E184/3 (2017)
20. Janhunen, T., Niemelä, I., Oetsch, J., Pührer, J., Tompits, H.: On testing answer-set programs. In: Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010), pp. 951–956. IOS Press (2010)
21. Kain, T.: Uhura: an authoring tool for translating controlled natural language into answer-set programs. Bachelor's thesis, Technische Universität Wien, Institute of Information Systems, E184/3 (2017)
22. Kuhn, T.: A survey and classification of controlled natural languages. Comput. Linguist. **40**(1), 121–170 (2014)
23. Leone, et al.: The DLV system for knowledge representation and reasoning. ACM Trans. Comput. Log. **7**(3), 499–562 (2006)
24. Mitra, A., Baral, C.: Learning to automatically solve logic grid puzzles. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP 2015), pp. 1023–1033. The Association for Computational Linguistics (2015)
25. Oetsch, J., Pührer, J., Tompits, H.: Stepwise debugging of answer-set programs. Theory Pract. Log. Program. **18**(1), 30–80 (2018)
26. Potassco. http://potassco.sourceforge.net
27. Ricca, F.: The DLV Java wrapper. In: Proceedings of the 8th Joint Conference on Declarative Programming (AGP 2003), pp. 263–274 (2003)
28. Schwitter, R.: English as a formal specification language. In: Proceedings of the 13th International Workshop on Database and Expert Systems Applications (DEXA 2002), pp. 228–232. IEEE (2002)
29. Schwitter, R.: Working for two: A bidirectional grammar for a controlled natural language. In: Wobcke, W., Zhang, M. (eds.) AI 2008. LNCS (LNAI), vol. 5360, pp. 168–179. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89378-3_17
30. Schwitter, R.: The jobs puzzle: Taking on the challenge via controlled natural language processing. Theory Pract. Log. Program. **13**(4–5), 487–501 (2013)
31. Wos, L., Overbeck, R., Lusk, E., Boyle, J.: Automated reasoning: Introduction and applications. Prentice Hall Inc., Upper Saddle River (1984)