Model Generation for Generalized Quantifiers via Answer Set Programming

Yuliya Lierler
University of Texas at Austin, Computer Science Department
yuliya@cs.utexas.edu

Günther Görz
University of Erlangen-Nürnberg, Computer Science Institute /8
goerz@informatik.uni-erlangen.de

Abstract. For the semantic evaluation of natural language sentences, in particular those containing generalized quantifiers, we subscribe to the generate and test methodology to produce models of such sentences. These models are considered as means by which the sentences can be interpreted within a natural language processing system. The goal of this paper is to demonstrate that answer set programming is a simple, efficient and particularly well suited model generation technique for this purpose, leading to a straightforward implementation.

1 Introduction

In Natural Language Processing researchers made considerable progress in the fields of speech recognition and syntactic analysis for practical applications. On the other hand the area of semantic analysis has so far mainly concentrated on representational and semantic construction issues. Several logical systems have been developed that are suited for the task of discourse meaning representation and in particular for anaphora binding, e.g., Discourse Representation Theory (DRT). At the same time the area of semantic analysis, i.e., possibility of inference based on discourse semantic information and world knowledge, still is the area where much work remains to be done.

Inference has long been admitted as a key concern in the design of natural language processing systems. It plays an important role in assigning meaning to utterances as well as verifying the consistency of an utterance within the world knowledge of the system or implementing commands encoded in the utterance. [3, 7, 10, 11, 4, 5] propose model generation as a key methodology for implementing inference in different areas of natural language processing. At the same time psycholinguistic studies support model-generation based methods from a conceptual view. Their studies show that during discourse comprehension humans build a logical form of a text as well as construct the state of affairs described by the discourse, i.e., models of the representation, so called mental models.

Nowadays there are two main research directions in the area of applying inference for natural language interpretation. One of them introduces the use of first order logic (FOL) as a metalanguage for encoding utterance information, world and situation knowledge as for instance demonstrated in [4, 5]. The main advantage of using first order logic is the availability of inference tools, such as first order logic model builders and theorem provers. Using first order logic for the task of inference within natural language processing brings several problems. On one hand first order logic is unable to express such language phenomena as generalized quantifiers like *most*, *many*. On the other hand expressing domain knowledge in first order logic is a very tedious task where the underlying theory is undecidable. Another way of applying inference for language interpretation is to design an inference engine specifically for a certain type of semantic representation language as for instance in [7, 10, 11]. The approach of developing specialized inference algorithms for semantic interpretation are not satisfactory in the sense that they are system and logic specific.

We see the possibility of using tools that are especially designed for knowledge representation and reasoning for the tasks of natural language processing inference be a compromise between two available directions of the research in semantic interpretation. We investigate the potential of using a logic programming paradigm called *answer set programming (ASP)* for model generation approach within natural language processing. Answer set programming is a form of the declarative programming paradigm [14, 15] related to logic programming languages, such as Prolog, where the solutions to a problem are represented by answer sets, and not by answer substitutions produced in response to a query as in conventional logic programming. As opposed to Prolog, this programming method uses specialized answer set solvers, such as for example SMODELS^A [6]. This approach is similar to propositional satisfiability checking, where a propositional formula encodes the problem and models of the formula correspond to the solutions of the problem. The model generation approach in place of query evaluation is the most characteristic feature of answer set programming. This methodology allows new ways of solving problems occurring in artificial intelligence. Within this paper we argue that it is also useful in providing solutions to the natural language processing interpretation problem. By now the answer set programming paradigm was successfully applied in various domains including space shuttle control [17], planning [12], and the design and implementation of question answering systems [1]. We investigate the applicability of the answer set programming methodology in the area of natural language processing on the task of providing the interpretation to sentences with generalized quantifiers. We argue that the method is competitive with other available tools in this field and may bring new insights in the area of model generation for the semantic interpretation of sentences.

As the main goal of this paper is to advocate ASP as a new and efficient model generation technique, having semantic evaluation in mind, we will not deal with general problems of semantic construction or problems of representing generalized quantifiers in a DRT framework. So, research problems in the area of generalized quantifiers such as scope problems, vague quantification, non-monotonicity, etc., as well as questions of scalability are beyond the scope of the present paper. Instead, we assume an appropriate semantic construction procedure as given and focus just on the model generation aspects using ASP. To investigate our central thesis, i.e. the feasibility of ASP for model generation and semantic evaluation in NLP, we begin with rather simple cases. It is important to notice that these simple examples are not meant as to provide a particular motivation for our approach. But with their help, we will show that ASP is in fact an appropriate approach, being simple, efficient and of suitable expressiveness at the same time, and that it supplies good reasons to be taken as the basis for the treatment of more complex cases — a research task still to be done.

So, in the following we present our first results to resolve the meaning of the natural language generalized quantifiers "all", "some", "two", and "most". We adopt the idea proposed in [2], that generalized quantifiers are relations between sets, where members of the sets are given by the semantics of the sentence. Note that we use the term set, while linguistically these sets can be seen as extensions. Consider the simple example sentence *All trains are fast*. In a view adopted from [2] the sentence expresses the relation *all* between set of trains and the set of fast trains. The relation *all* is satisfied only if the two sets are identical.

Of course, there are different issues within generalized quantifier interpretation. Unsurprisingly, their interpretations are often ambiguous. For example, in Discourse Representation Theory [9] several algorithms are being proposed for constructing the representation for the same sentence that contains a generalized quantifier. But, as pointed out above, these questions are beyond the scope of the present paper.

The way we approach the interpretation of generalized quantifiers is by applying the *generate and test* methodology. First, we encode the problem of generating prospective models by means of logic rules in the answer set program. Intuitively, prospective models are sets that could correspond to the meaning of a sentence, i.e. the sets that might satisfy the relation on the sentence provided by the generalized quantifier. For example, when the domain consists of three trains $\{t1, t2, t3\}$ and the sentence to be analyzed is *All trains are fast*, then the one possible and hence prospective model corresponding to the sentence, i.e. the model that contains all trains, is $\{t1, t2, t3\}$ such that they are fast. The second part of the logic program encodes the test on prospective models on their plausibility by using pragmatic, world, or domain knowledge. Within the paper we present only the outline of our idea on interpreting generalized quantifiers. We demonstrate the approach using sample sentences with generalized quantifiers.

The paper is organised as follows. First, we introduce the formal concepts of the answer set programming sufficient to understand the further encodings in the paper. Second, the specifications and encoding of sample domain are given. Then, we present the analysis of simple example sentences that contain generalized quantifiers, and specify the details of their processing within the generate and test framework.

2 Answer Set Programming

A *rule* is an expression

$$a_0 \leftarrow a_1, \ldots, a_m, not\ a_{m+1}, \ldots, not\ a_n \tag{1}$$

where a_0 is an atom or the symbol $\bot$ for falsehood, and $a_1, \ldots, a_n$ are atoms for $0 \leq m \leq n$. Atom a_0 is called *head* of the rule whereas $a_1, \ldots, a_m, not\ a_{m+1}, \ldots, not\ a_n$ is the *body*. If the head of a rule is $\bot$ then the rule is called a *constraint* and it is written

$$\leftarrow a_1, \ldots, a_m, not\ a_{m+1}, \ldots, not\ a_n.$$

If the body of a rule is empty then the rule is called a *fact*. A *logic program* is a finite set of rules.

We interpret logic programs via answer set semantics [8, 16, 18]. Let Π be a logic program comprising rules with $n = m$ (i.e. Π is a program without any occurrence of not) and let X be a set of atoms; we say that X is *closed* under Π if, for every rule in Π, $a_0 \in X$ whenever $\{a_1, \ldots, a_m\} \subseteq X$. We say that X is an *answer set for* Π if X is the smallest set closed under Π. Now let Π be an arbitrary logic program and let X be a set of atoms. The *reduct* Π^X *of* Π *relative to* X is the set of rules $a_0 \leftarrow a_1, \ldots, a_m$ for all rules in Π such that $X \cap \{a_{m+1}, \ldots, a_n\} = \emptyset$. Thus Π^X is a program without not. Set X is an *answer set for* Π if X is an answer set for Π^X.

We now extend the class of rules with *choice rules*, i.e. expressions of the form:

$$\{a_0\} \leftarrow a_1, \ldots, a_m, not\ a_{m+1}, \ldots, not\ a_n \tag{2}$$

where $a_0, \ldots, a_n$ are atoms. For the lack of space here we do not provide the precise definition of the semantics of logic programs with choice rules[1]. However, for the purpose of this presentation, it suffices to give the following, informal explanation. We say that the body of rules of the form (1) or (2) is satisfied by X if $\{a_1, \ldots, a_m\} \subseteq X$ and $\{a_{m+1}, \ldots, a_n\} \cap X = \emptyset$. On the one hand, rule (1) prescribes that if the body is satisfied by the answer set then its head must be in the answer set too. Choice rule (2), on the other hand, prescribes that if its body is satisfied by the answer set, its head *may be* in the answer set.

To illustrate, let us consider the program composed by the rules

$$a.\ \ \{b\} \leftarrow a.\ \ c \leftarrow b.$$

It has two answer sets, $\{a\}$ and $\{a, b, c\}$. The body of the first rule is satisfied by any set therefore a shall be part of all answer sets. The body of the second choice rule is also satisfied by all answer sets hence b may be in the answer set. The satisfaction of the body of the last rule depends on whether b is in the answer set or not. By adding the constraint $\leftarrow c.$ to the example program we can eliminate the second answer set.

We finally extend the class of rules with *aggregate rules*, i.e. expressions of the form:

$$a_0 \leftarrow aggr,\ a_1, \ldots, a_m, not\ a_{m+1}, \ldots, not\ a_n \tag{3}$$

where $a_0, \ldots, a_n$ are atoms, and $aggr$ is an aggregate literal of the form $F(\{X, p(X)\})op\ Value$, F is an aggregate function (e.g., *count*, *sum*), *op* is a relational operator (e.g. $=, >$). Due to the lack of space we refer the readers to [18] for a precise definition of an answer set for logic programs with aggregate rules. However, we

[1] For a precise definition of an answer set for logic programs with choice rules please see [16].

illustrate the use of aggregate rules in the following example. Let us consider the program composed by the rules

$$set(a).\ set(b).\ set(c).$$
$$twoElementsSet \leftarrow count(\{X, set(X)\}) = 2.$$
$$threeElementsSet \leftarrow count(\{X, set(X)\}) = 3.$$

It has one answer set $\{set(a)\ set(b)\ set(c)\ threeElementsSet\}$. For simplicity, in the sequel we use the term rules in a broad sense so to encompass also choice and aggregate rules.

The answer set programming system SMODELS^A accepts the programs containing rules (1), (2), and (3) produced by the grounder LPARSE[2]. LPARSE allows variables in its syntax, and makes use of so called domain predicates in order to ground the program so that it contains only atoms. Later in the presentation we use the rules with variables but it shall be intuitive which ground rules they result in.

The common answer set programming style is to split the problem specification into two subproblems: *generate*, and *test* [13], where

1. the *generate* part of a program defines a potential set of solutions,
2. the *test* part consists of the constraints which "weed out" the answer sets generated in the *generate* part that don't satisfy the constraints.

We adopt this methodology for encoding the problem of interpreting the sentences with generalized quantifiers in the following.

3 Generate and Test via Answer Set Programming

3.1 Domain Specification

We demonstrate our way of interpreting generalized quantifiers with the help of examples with the assumption that a natural language processing system has a built-in world model — the domain knowledge. First, let us formalise our example domain knowledge and encode it as a logic program.

Example Specification: *There are three trains in the world. Constants* $t1,\ t2,\ t3$ *correspond to the train instances. The trains* $t1, t2$ *have the property of being* $fast$, $t3$ *has the property of being* $slow$. The encoding of the domain follows:

$$\begin{array}{l} num(1).\ num(2).\ num(3). \\ object(t1; t2; t3). \\ pred(train, t1).\ pred(train, t2).\ pred(train, t3). \\ pred(fast, t1).\ pred(fast, t2).\ pred(slow, t3). \end{array} \quad (4)$$

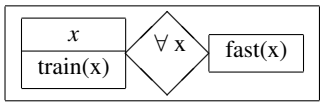
3.2 Sentence Representation

We choose four naive sentences to demonstrate the approach:

All trains are fast. [s-1]
Some trains are fast. [s-2]
Two trains are fast. [s-3]
Most trains are fast. [s-4]

Although these are simple sentences, they are sufficient for our presentation.

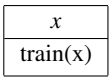
Within this paper we consider duplex-DRS [9] to denote a formal representation of a sentence. For instance, sentence [s-1] has the following duplex-DRS representation



(5)

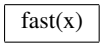
[2] http://saturn.hut.fi/pub/smodels/ .

where



(6)

is called a *restrictor*;



(7)

is called a *nuclear scope*; $\langle \forall x \rangle$ is the formal representation of the relation *all* and is called a *quantifier*. Restrictor, nuclear scope and (5) are Discourse Representation Structures (DRSs), where (5) and (7) contain only conditions, and (6) consists of discourse referents, and conditions listed respectively above and below the horizontal bar.

Duplex-DRS as a Logic Program. First we define constants *forall, exists, two, most* to correspond to duplex-DRS quantifiers $\langle \forall\ x \rangle$, $\langle \exists\ x \rangle$, $\langle two\ x \rangle$, $\langle most\ x \rangle$ respectively. The specifications of duplex-DRS contains logic rules specifying a restrictor, a nuclear scope, and a quantifier. For instance duplex-DRS (5) specification consists of

$$restrictor(train, x).\quad nuclearScope(fast, x). \quad (8)$$

and

$$quantifier(forall, x). \quad (9)$$

Within a representation of duplex-DRSes for sentences [s-2,s-3,s-4] rule (9) would be replaced by the following rules respectively:

$$quantifier(exists, x). \quad (10)$$
$$quantifier(two, x). \quad (11)$$
$$quantifier(most, x). \quad (12)$$

3.3 Prospective Models

As we mentioned in the introduction we take a view at generalized quantifiers as relations on sets. In this sense we can describe the relation *all* between two sets A and B as follows

- *All* A *are* B whenever $A \subseteq B$

Additionally, it was noted by Barwise and Cooper [2] that the relation $B \subseteq A$ always holds in case of all natural language generalized quantifiers. (There are exceptions as *only*, but we do not consider such cases in this work). Hence the relation that correspond to natural language quantifier *all* can be given as

- *All* A *are* B *whenever* $A = B$.

The generation of sentence models corresponding to its meaning proceeds as follows. First, we encode the *generate* part of the program that is able to enumerate the prospective models of the sentence. By a *prospective model* we identify a pair of sets $\langle A, B \rangle$ such that

i if and only if $a \in A$ then a satisfies the restrictor of duplex-DRS, i.e., A is an extension of the restrictor, e.g., for sentence [s-1] $A = \{a | train(a)\}$,
ii $B \subseteq A$,
iii sets A and B satisfy the quantifier condition of the duplex-DRS. In case of *all*-$\langle \forall x \rangle$ quantifier, its condition is $A \subseteq B$ $(A = B)$.

For example, there exists only one prospective model

$$\langle \{t1, t2, t3\}, \{t1, t2, t3\} \rangle$$

for sentence [s-1] within the domain described in section 3.1. Second, we encode the *test* part of the program that lists the constraints

on the solutions based on the sentence nuclear scope, domain knowledge, and situation. This part of the program verifies the plausibility of prospective models. Domain and sentence representation in logic program syntax plus the generate and test parts produce the logic program, whose answer sets represent the possible sentence interpretations.

Prospective Models for Some. We can describe the relationship *some* between two sets A and B as follows

- *Some* A *are* B whenever $A \cap B \neq \emptyset$

The condition (iii) of the prospective model definition in case of the natural language quantifier *some*-$\langle \exists x \rangle$ is $B \neq \emptyset$. Consider a sentence [s-2]. There are seven prospective models that correspond to this sentence and domain from section 3.1, where A is the set $\{t1, t2, t3\}$, while B may have seven corresponding assignments $\{t1, t2, t3\}$, $\{t1\}$, $\{t1, t2\}$, $\{t2, t3\}$, $\{t1, t3\}$, $\{t2\}$, $\{t3\}$.

Prospective Models for Two. First we may note the ambiguity of the word *two*. In some sentences it might mean *exactly two* in others *at least two*, or *at most two*. At the moment we abstract to the former meaning of *two – exactly two*.

We can describe the relationship *two* between the sets A and B by

- *Two* A *are* B whenever $|A \cap B| = 2$

The condition (iii) of the prospective model definition for the natural language quantifier *two*-$\langle$*two x*$\rangle$ is $|B| = 2$. Consider sentence [s-3]. In case of the domain described in section 3.1, and sentence [s-3] there are three prospective models, where A is set $\{t1, t2, t3\}$, while B may have three corresponding assignments $\{t1, t2\}$, $\{t2, t3\}$, $\{t1, t3\}$.

Prospective Models for Most. We can describe the relationship *most* between two sets A and B as follows

- *Most* A *are* B whenever $|A \cap B| > \frac{|A|}{2}$

The condition (iii) of the prospective model definition for the natural language quantifier *most*-$\langle$*most x*$\rangle$ is $|B| > \frac{|A|}{2}$. Consider sentence [s-4]. In case of the domain described in section 3.1 and sentence [s-4] there are four prospective models, where A is set $\{t1, t2, t3\}$, while B may have four corresponding assignments $\{t1, t2\}$, $\{t2, t3\}$, $\{t1, t3\}$, $\{t1, t2, t3\}$.

3.4 Generate

Our task now is to encode the generate part of the logic program that is able to enumerate the prospective models of a sentence, when domain specification and sentence duplex-DRS representation are provided as an input. The encoding *AB_Generate* follows:

$$\begin{array}{l} a_{el}(X, x) \leftarrow restrictor(Y, x), pred(Y, X). \\ \{b_{el}(X, x)\} \leftarrow a_{el}(X, x). \end{array} \quad (13)$$

$$\begin{array}{l} negAll \leftarrow a_{el}(X, x),\ not\ b_{el}(X, x). \\ forall \leftarrow\ not\ negAll. \\ \leftarrow quantifier(forall, x), not\ forall. \end{array} \quad (14)$$

$$\begin{array}{l} exists \leftarrow b_{el}(X, x). \\ \leftarrow quantifier(exists, x), not\ exists. \end{array} \quad (15)$$

$$\begin{array}{l} two \leftarrow count(\{X, b_{el}(X, x)\}) = 2. \\ \leftarrow quantifier(two, x),\ not\ two. \end{array} \quad (16)$$

$$\begin{array}{l} most \leftarrow count(\{X, b_{el}(X, x)\}) > K1, \\ \qquad assign(K1, div(K, 2)), numA(K),\ num(K). \\ numA(K) \leftarrow count(\{X, a_{el}(X, x)\}) = K, num(K). \\ \leftarrow quantifier(most, x),\ not\ most. \end{array} \quad (17)$$

where (13) corresponds to the code that generates the candidate members for the sets A and B, i.e., predicates $a_{el}(X, x)$, $b_{el}(X, x)$ state that X is a member of A, B respectively. (14,15,16,17) specify the conditions that candidate set B generated by (13) must satisfy when the quantifier is *all*, *some*, *two*, or *most* respectively. Consider specification (17). The first rule, *most*-rule, states that the number of objects that are elements of set B is greater than $K1$ where $K1 = \frac{|A|}{2}$ and $\frac{|A|}{2}$ is computed by the second rule of (17). The constraint $\leftarrow quantifier(most, x),\ not\ most.$ on the other hand forbids solutions where $quantifier(most, x)$ is provided by a duplex-DRS specification, but the requirement of *most*-rule is not satisfied.

There exists a mapping between prospective models and answer sets of a program consisting of the domain, a sentence duplex-DRS specification, and AB_Generate. Let us consider the domain from section 3.1, and sentence [s-3]. The logic program corresponding to a prospective model generator for this example is a combination of the domain specification (4) with (8,11,13,16). There are three answer sets for this program that correspond to prospective models

$$\langle \{t1, t2, t3\}, \{t1, t2\} \rangle \langle \{t1, t2, t3\}, \{t2, t3\} \rangle \langle \{t1, t2, t3\}, \{t1, t3\} \rangle$$

At this point we are done with the generation part of our approach and proceed to the testing of prospective models.

3.5 Test

First let us explain what do we mean by prospective model testing. What are the criteria that would make us to accept one prospective model and reject another? We remind ourselves that our prototypical natural language processing system works within a specific domain possessing concrete knowledge about the surrounding world. The prospective models that we accept and call *SAT-models* are the ones that do not conflict with the domain knowledge of the natural language processing system. Although for some applications we argue and demonstrate by examples that prospective models might play an important role even when they are inconsistent with the domain knowledge.

It is easy to notice that deciding within a specific application if the model is acceptable might depend on such factors as whether the sentence is an imperative, a statement, or a question, or whether the speaker is trustworthy. In this section we would like to outline the flexibility of an answer set generate-and-test approach that may facilitate alternative interpretations of the sentences depending on specific natural language applications. For instance, natural language driven control system might evaluate a sentence differently from a question answering system. In the previous section we provided the means to generate prospective models that hold for the sentences with generalized quantifiers in all applications, while testing of the prospective models is application specific and often requires not only linguistic data, but also background, domain, and application knowledge. Therefore, in the following we only outline some possible scenarios for model testing that take into account types of the sentences and demonstrate that the answer set programming technique is sufficient to support these scenarios. Implementation of an approach in the real world natural language application will provide the new insights on the approach.

Let us consider domain from section 3.1 and the sentences:

Are all trains fast? (S-1)
Make all trains fast. (S-2)
Trains are all fast. (S-3)
Are most trains fast? (S-4)

Sentences (S-1, S-2, S-3) have only one prospective model $\langle\{t1, t2, t3\}, \{t1, t2, t3\}\rangle$. For these sentences the prospective model assumes that all trains are fast while in our sample domain we have one slow train. On the other hand sentence (S-4) has four prospective models

$$\langle\{t1, t2, t3\}, \{t1, t2\}\rangle, \langle\{t1, t2, t3\}, \{t2, t3\}\rangle,$$
$$\langle\{t1, t2, t3\}, \{t1, t3\}\rangle, \langle\{t1, t2, t3\}, \{t1, t2, t3\}\rangle$$

where, for instance, first model assumes that trains $t1$ and $t2$ are fast.

Consider question (S-1). Its only prospective model is not consistent with the domain knowledge hence no SAT-model shall be generated from it. No SAT-model found corresponds to a negative answer to the posed question. Consider question (S-4). Only one of the four prospective models $\langle\{t1, t2, t3\}, \{t1, t2\}\rangle$ is consistent with the domain knowledge. Therefore there shall be only one answer set found that corresponds to a SAT-model of (S-4) and states that indeed most of the trains are fast, i.e., trains $t1$ and $t2$ are fast.

On the implementational level what needs to be checked is whether the objects of B in the prospective model pair $\langle A, B\rangle$ possess the properties given in a nuclear scope of duplex-DRS corresponding to the question. Prospective models that satisfy such constraints are accepted as SAT-models. Answer set programming constraint

$$\leftarrow b_{el}(B, x), not\ pred(X, B), \\ nuclearScope(X, x). \tag{18}$$

forbids the answer sets where within the domain knowledge, some element of B does not possess the property specified by the nuclear scope. Remember that some element b in B corresponds to $b_{el}(b, x)$ predicate in our program.

There is no answer set for the program Π_{S-1} constructed from domain specification (4), duplex-DRS corresponding to (S-N)[3]: (8, 9, 14), generate (13), and test (18). This corresponds to our intuition that the only prospective model of (S-1) is rejected that states a negative answer to the question (S-1). Note that this implementation rejects a prospective model even in case of incomplete knowledge when the system is not aware of the speed property of some train.

The answer set program constructed from domain specification (4), duplex-DRS corresponding to (S-4): (8, 12, 17), generate (13), and test (18). produces one answer set that corresponds to the SAT-model $\langle\{t1, t2, t3\}, \{t1, t2\}\rangle$. This meets our intuition that only one prospective model of (S-4) shall be accepted as a SAT-model that states a positive answer to the question (S-4).

In the case of the imperative sentence (S-2), its only prospective model is inconsistent with the domain description from section 3.1. Nevertheless rejecting the prospective model as in the case of question (S-1) might be not the optimal solution for all natural language processing systems. By means of answer set programming another possibility can be easily implemented. For instance, the answer set of a program may present a prospective model together with the list of objects in the domain required to possess some properties in order for prospective model to be consistent with the domain knowledge. Consider our sample domain and imperative sentence (S-2): train $t3$ shall possess a property of being *fast* in order for prospective model be consistent with the domain.

[3] S-N denotes S-1,S-2,S-3 due to the fact that duplex-DRS specifications corresponding to these sentences are identical.

The answer set program Π_{S-2} constructed from domain specification (4), the duplex-DRS corresponding to (S-N): (8, 9, 14), generate (13), and test rule

$$bPosses(B, X) \leftarrow b_{el}(B, x), not\ pred(X, B), \\ nuclearScope(X, x). \tag{19}$$

produces one answer set that contains the predicate $bPosses(t3,\ fast)$ and the prospective model $\langle\{t1, t2, t3\}, \{t1, t2, t3\}\rangle$. Such an answer set on the application level provides the information that in order to satisfy the requests of the imperative sentence, train $t3$ shall possess the property of being fast.

At the same time the answer set programming paradigm can easily encode such information as whether it is in general possible for a train from being slow to become fast. For instance, constraint

$$\leftarrow pred(slow, X), pred(fast, X). \tag{20}$$

states that if a train is slow, it in no way may be fast, i.e., the only prospective model for (S-2) can be rejected based on such a constraint of the domain knowledge. Intuitively, program Π_{S-2} extended with constraint (20) has no answer sets.

Consider statement (S-3). Its only prospective model assumes that all trains are fast while in our sample domain we have one slow train. Based on domain knowledge of a system and statement (S-3) inconsistency, for some natural language processing applications it might be expected a behaviour as in case of question (S-1) where no answer set is produced for a program Π_{S-1}. The natural language system may then conclude that newly received information is inconsistent with its knowledge and act upon that.

For some other applications, behaviour as in the case of the imperative sentence (S-2) might be more appropriate, where the answer set of Π_{S-2} conveys to a natural language processing system that the received information encoded by a statement differs from the domain knowledge w.r.t. certain information. For instance, for the case of our sample domain and statement (S-3), the answer set of Π_{S-2} states that domain object train $t3$ shall possess a property of being *fast* in order for statement (S-3) be consistent with the domain.

Here we conclude the presentation of some possible test scenarios. It is important to underline that the flexibility of the proposed answer set programming methodology allows more sophisticated testing mechanisms that may take into account such information as, e.g., background knowledge and system requirements.

4 Conclusions

We proposed the application of the generate and test methodology for finding the interpretations for sentences with generalized quantifiers. We see the generate part of the approach as the procedure that can be defined based on the generalized quantifier, the domain knowledge, and the sentence duplex-DRS. The test part is more sophisticated in a sense that also the type of the speech act, domain and situation knowledge, and further natural language processing system requirements may need to be taken into account in order to evaluate the meaning of the sentence.

We demonstrated a way of using the answer set programming paradigm for generating and testing the models corresponding to the meaning of sentences with generalized quantifiers. The encoding is modular and allows for flexible extensions. We see this feature of the approach as its biggest advantage, that may allow quick and sophisticated encoding of different strategies for testing the models with respect to the domain knowledge, the type of the speech acts, and further natural language processing system requirements.

Acknowledgements

We are grateful to Bernd Ludwig, Peter Reiss, Bernhard Schiemann, and Iman Thabet for discussions on the topic of this work, and Vladimir Lifschitz for the comments on a first draft of this paper.

REFERENCES

[1] Chitta Baral, Michael Gelfond, and Richard Scherl, 'Using answer set programming to answer complex queries', in *Workshop on Pragmatics of Question Answering at HLT-NAAC2004*, (2004).

[2] J. Barwise and R. Cooper, 'Generalized quantifiers and natural language', *Linguistics and philosophy*, (1981).

[3] Peter Baumgartner and Michael Kühn, 'Abductive Coreference by Model Construction', Technical Report 6–99, Universität Koblenz-Landau, Institut für Informatik, Rheinau 1, D-56075 Koblenz, (1999).

[4] P. Blackburn and J. Bos, *Representation and inference for natural language*, CSLI Publicatiosn, 2005.

[5] J. Bos, 'Exploring model building for natural language understanding', in *Proc. ICOS'4*, (2003).

[6] I. Elkabani, E. Pontelli, and T.C. Son, 'SMODELS^A — a system for computing answer sets of logic programs', in *Proc. LPNMR-8*, (2005). Available under `http://www.cs.nmsu.edu/ ielkaban/asp-aggr.html`.

[7] C. Gardent and K. Konrad, 'Interpreting definites using model generation', *Language and Computation*, **1**, 193–209, (2000).

[8] Michael Gelfond and Vladimir Lifschitz, 'The stable model semantics for logic programming', in *Logic Programming: Proc. Fifth Int'l Conf. and Symp.*, eds., Robert Kowalski and Kenneth Bowen, pp. 1070–1080, (1988).

[9] H. Kamp and U. Reyle, *From discourse to logic*, volume 1,2, Kluwer, 1993.

[10] M. Kohlhase, 'Model generation for discourse representation theory', in *Proc. 14th ECAI*, (2000).

[11] K. Konrad, *Model generation for natural language interpretation and analyses*, Ph.D. dissertation, University of Saarbruecken, 2000.

[12] Vladimir Lifschitz, 'Answer set planning', in *Proc. ICLP-99*, pp. 23–37, (1999).

[13] Vladimir Lifschitz, 'Answer set programming and plan generation', *Artificial Intelligence*, **138**, 39–54, (2002).

[14] V. Marek and M. Truszczyński, 'Stable models and an alternative logic programming paradigm', in *The Logic Programming Paradigm: a 25-Year Perspective*, 375–398, (1999).

[15] I. Niemelä, 'Logic programs with stable model semantics as a constraint programming paradigm', *Annals of Mathematics and Artificial Intelligence*, 241–273, (1999).

[16] Ilkka Niemelä and Patrik Simons, 'Extending the Smodels system with cardinality and weight constraints', in *Logic-Based Artificial Intelligence*, ed., Jack Minker, 491–521, Kluwer, (2000).

[17] Monica Nogueira, Marcello Balduccini, Michael Gelfond, Richard Watson, and Matthew Barry, 'An A-Prolog decision support system for the space shuttle', in *Working Notes of the AAAI Spring Symposium on Answer Set Programming*, (2001).

[18] T.C. Son, E. Pontelli, and I. Elkabani, 'A translational semantics for aggregates in logic programming', Technical Report NMSU-CS-2005-005, New Mexico State University, (2005).