Automatic Translation of Logic Grid Puzzles into a Typed Logic

Jens Claes

***Abstract*—This paper extends and evaluates the semantical framework of Blackburn and Bos for knowledge representation. Specifically, the framework is used to translate logic grid puzzles into logic.**

We introduce types into the framework to translate to a typed logic and to allow the system to infer the domains of a logic grid puzzle from its sentences in natural language. This extension also enables type checking of sentences in natural language. Therefore, it can reject sentences that are grammatical but without meaning.

I. Introduction

KNOWLEDGE base systems have been around for a while. They take a knowledge base as input. This knowledge base describes the world, i.e. it is a specification of how something should work. Based on this knowledge base, different kinds of inference can be applied. De Cat et al. [1] give the example of an university course-management system. The knowledge base contains rules like "In every auditorium at a any time, there should be at most one course.". The different kinds of inferences that can be applied, include (the examples are from De Cat et al. [1]): *propagation* (e.g. automatically selecting required prerequisites when constructing an individual study program), *model expansion* (e.g. getting a full individual study program from a partial one) and *querying* (e.g. getting the schedule for a specific student).

These knowledge base systems have triggered a lot of research into formal languages and their expressivity. These languages are often hard to read, write and learn.

One method to ease the writing of formal theories is to develop languages that are somewhere in between natural language and formal languages. Such languages are called Controlled Natural Languages (CNL). A CNL is a subset of a natural language that, for example, allows to write specifications in a more consistent language. Kuhn [2] made an overview of 100 CNL's. Some of these languages have formal semantics and can be translated automatically into a formal logic. They can be used as a knowledge representation language within a knowledge base system. They are often easy to read. Unfortunately, the translations of such CNL's into logic are not well documented which makes it hard to expand the language.

This paper therefore constructs a new CNL based on the language used in a small domain, namely logic grid puzzles. It is then tested on unseen puzzles. The first goal of this paper is to show that the framework introduced by Blackburn and Bos [3], [4] can be used to automatically translate such a CNL into a formal logic. The second goal is to show that adding types to this framework is useful. It makes it possible to translate to a typed logic and allows more inference. E.g. The system can automatically infer the domains of a logic grid puzzle.

Therefore, we take the next step for bridging the gap between natural language and logic. We extend the well documented framework of Blackburn and Bos with types. With this extension, we constructed the first typed CNL with a formal semantics in a typed logic.

The result of this research is a fully automated tool that can understand a logic grid puzzle and reason about it (e.g. the system can solve the puzzle automatically).

II. Logic Grid Puzzles

Logic grid puzzles are logical puzzles. The most famous such puzzle is probably the Zebra Puzzle [5], sometimes also known as Einstein's Puzzle. These puzzles consist of a number of sentences or clues in natural language. In the clues, a number of domains appear (e.g. nationality, color, animal, ...), each with a number of domain elements (e.g. Norwegian, Canadian, blue, red, cow, horse, ...). Between each domain there is a bijection. The goal of such a puzzle is to find the value of these bijections, i.e. to find which domain elements belong together. E.g. The Norwegian lives in a blue house and keeps the horse. Every puzzle has an unique solution.

Logic grid puzzles can be considered as small specifications. Moreover, they can be expressed in fairly simple logical statements. Finally, it is easy to find numerous examples of these types of puzzles. It is these three properties that make logic grid puzzles the ideal domain to test our assumptions. Namely that the framework of Blackburn and Bos allows to construct a CNL with formal semantics that can be used for knowledge representation (within a small domain).

III. Motivation

Since knowledge representation languages can be hard to read, write and learn, an expert in formal languages is required to write and maintain the specification. A formal language that is more accessible for non-experts, like a CNL with formal semantics, can solve this issue without compromising on the strength of knowledge base systems. It is still possible to apply different kind of inferences on a specification that is more accessible for non-experts.

For example, when the clues of a logic grid puzzle are expressed in a CNL (with formal semantics), it is possible to automatically construct a complete specification such that a knowledge base system can reason about the puzzle. Such a system can

- Solve the puzzle automatically.
- Given a (partial) solution by the user, indicate which clues have already been incorporated in the solution, which clues still hold some new information and which clues have been violated.
- Given a partial solution by the user, automatically derive a subset of clues that can be used to expand the solution. This can be part of a "hint"-system for the user.
- Given a set of clues, indicate which solutions are still possible. This can help the author while writing new puzzles. Based on the solutions that are still possible, the author can construct a new clue to eliminate some of the possibilities until only one solution remains.

IV. An overview of the system

We created such a system. It can reason about a logic grid puzzle starting from the clues in (controlled) natural language and a puzzle-specific lexicon. The system uses the following steps

1) It translates the clues into logic using the extended framework of Blackburn and Bos. While doing so, it gathers type information about the words in the lexicon.
2) The system uses this type information to automatically deduce the domains of the puzzle. If necessary, it asks the user questions.
3) The system then constructs a formal vocabulary based on these domains.
4) Thereafter, it formulates the necessary axioms to model the implicit assumptions of a logic grid puzzle
5) Finally, the system passes the complete specification (consisting of the formal vocabulary and the theory with the axioms and the translation of the clues) to the IDP system [1] to get a solution to the puzzle (or to apply another kind of inference)

For this system, we extended the framework of Blackburn and Bos, we constructed a grammar and a set of lexical categories that can be used in this framework for the translation of logic grid puzzles into logic. We devised an inference system to deduce the different domains from the type information. We contrived how to express these domains in a formal vocabulary and finally, we listed the axioms necessary to complete the specification.

V. A semantical framework

Blackburn and Bos constructed a framework [3], [4] to capture the meaning of natural language by translating it into a logic. It consists of four parts: the lexicon, the grammar, the semantics of the lexicon and the semantics of the grammar. The framework is based on the λ-calculus and Frege's compositionality principle. Every word has a λ-expression as its meaning. The meaning of a group of words is a combination of the meaning of the words that are part of the group. In this framework, λ-application is used to combine the meaning of words.

The grammar defines which sentences are grammatically correct (and thus allowed) and which are not. E.g. a correct sentence consists of a noun phrase and a verb phrase that agree in number. This process results in a parse tree of every grammatically correct sentence, in which the words are the leaves. The framework then combines the meaning of the leaves (the words) upwards in the tree to the meaning of the root node (the sentence).

```
s([]) -->
  np([num:Num]),
  vp([num:Num]).
```

```
4  np([num:sg]) -->
5    pn([]).
6  np([num:Num]) -->
7    det([]),
8    n([num:Num]).
9  vp([num:Num, sem:Sem]) -->
10   iv([num:Num, sem:Sem]).
11 vp([num:Num, sem:Sem]) -->
12   tv([num:Num, sem:]),
13   np([num:_]).
```

Grammar 1: A simple grammar

The lexicon consists of the enumeration of the different words that can be used along with some linguistic features. E.g. "loves" is a transitive verb in the present tense, "love" is a transitive verb and an infinitive. A lot of words also have a feature *Symbol* which is used in the translation as the name of a constant or predicate.

The goal of the framework is to lexicalize as much of the meaning as possible. The semantics of a grammar rule should be limited to λ-applications, if possible. Table I gives the semantics for grammar 1.

Grammatical rule	**Semantics**
S (line 1-3)	$\llbracket s \rrbracket = \llbracket vp \rrbracket\left(\llbracket np \rrbracket\right)$
NP1 (line 4-5)	$\llbracket np \rrbracket = \llbracket pn \rrbracket$
NP2 (line 6-8)	$\llbracket np \rrbracket = \llbracket det \rrbracket\left(\llbracket n \rrbracket\right)$
VP1 (line 9-10)	$\llbracket vp \rrbracket = \llbracket iv \rrbracket$
VP2 (line 11-13)	$\llbracket vp \rrbracket = \llbracket tv \rrbracket\left(\llbracket np \rrbracket\right)$

TABLE I: The semantics of grammar 1

The meaning of the words themselves is the only thing still missing. Blackburn and Bos assume that the meaning of a word is only dependent on its linguistic features, most importantly its lexical category. Constructing these λ-expressions can be hard. Therefore, it can help to analyze the signature of these expressions. Blackburn and Bos suggest a λ-calculus with two types for this: e represents entities and t truth-values. The signature of a noun is $\tau(n) = e \rightarrow t$, given an entity, the noun says whether or not the entity can be described with the noun. E.g. $\llbracket n_{man} \rrbracket = \lambda x \cdot man(x)$

A noun phrase represents one or more entities. E.g. "a man", "every woman". A sentence is true if enough entities of the noun phrase satisfy a certain property of the verb phrase. The signature of a noun phrase is thus $\tau(np) = (e \rightarrow t) \rightarrow t$. Given a property P with signature $e \rightarrow t$, the semantics of the noun phrase tell us if the noun phrase satisfies the property P of the verb phrase. The simplest noun phrase is a proper noun like John. Its semantics is $\llbracket pn_{John} \rrbracket = \lambda P \cdot P\left(John\right)$. I.e. when we say something about "John", the property should hold for $John$. The meaning of the noun phrase "every man" is $\llbracket np_{every\ man} \rrbracket = \lambda P \cdot \forall x \cdot man(x) \Rightarrow P(x)$. The property P should hold for all men for the sentence to be true. From this, we conclude the meaning of "every" as $\llbracket det_{universal} \rrbracket = \lambda R \cdot \lambda P \cdot \forall x \cdot R(x) \Rightarrow P(x)$. We call R the restriction by the noun. We can generalize this to other determiners like "two": $\llbracket det_{binary} \rrbracket = \lambda R \cdot \lambda P \cdot \exists_{>2} x \cdot R(x) \wedge P(x)$.

A verb phrase should say whether the sentence is true or not, given a noun phrase as its subject. An intransitive verb like "sleeps" is the simplest verb phrase. Its meaning is $\llbracket iv_{sleeps} \rrbracket = \lambda S \cdot S\left(\lambda x \cdot sleeps(x)\right)$. The sentence is true if the subject S holds the property $\lambda x \cdot sleeps(x)$. In other words, the sentence is true, if the semantics of the subject S (a function from a property with signature $e \rightarrow t$ to a truth value) evaluates to $true$ for the property $\lambda x \cdot sleeps(x)$.

The meaning of the sentence "John sleeps" is then

$$\llbracket s \rrbracket = \llbracket vp \rrbracket\left(\llbracket np \rrbracket\right) =$$

$$\left(\lambda S \cdot S\left(\lambda x \cdot sleeps(x)\right)\right)\left(\lambda P \cdot P\left(John\right)\right)$$

$$= sleeps(John)$$

Finally, the meaning of a transitive verb like "loves" is

$$\llbracket tv_{loves} \rrbracket = \lambda O \cdot \lambda S \cdot S\left(\lambda x_S \cdot O\left(\lambda x_O \cdot loves(x_S, x_O)\right)\right)$$

An entity x_S from the subject S satisfies the property if there are enough corresponding x_O's from the object O such that $loves(x_S, x_O)$.

For example for the sentence "Every man loves a woman", a man x_S satisfies the verb phrase ("loves a woman") if for this man x_S, there is a woman x_O such that $loves(x_S, x_O)$. The subject (as a whole) satisfies the verb phrase if *every* man x_S satisfies the verb phrase, i.e. if there is a woman x_O for every man x_S such that the man loves the woman. This woman can be different for every man.

This translation therefore decides how quantifier scope ambiguities are resolved, namely the translation follows the order of the natural language. This is how all quantifier scope ambiguities are resolved within this paper. Blackburn and Bos explain in their books how all readings can be obtained.

A. A lexicon for logic grid puzzles

To use the framework for translating logic grid puzzles into logic, we constructed a set of 12 lexical categories. These categories are determiner, number, noun, proper noun, preposition, relative pronoun, transitive verb, auxiliary verb, copular verb, comparative, *some*-words (somewhat, sometime, ...) and conjunction. Because of space constraints, we refer the reader to the full master thesis for the λ-expression of these categories.

The determiners in logic grid puzzles we studied are simple in that only existential quantifiers are necessary. Universal quantification is not needed as the puzzles always fully classify their noun phrases. This is a consequence of the fact that all relations described in these puzzles are bijections between the different domains. There is always exactly one person who, for example, drinks tea. For the same reason there is also no negative quantifier. Sentences like “No person drinks tea” do not occur in the puzzles we studied.

Proper nouns play an important role in our translation of logic grid puzzles into logic. We assume that every domain element (of a non-numeric) domain is represented by a proper noun (and only one). Even when the words are adjectives, they must be declared as proper nouns to the system. Proper nouns can also be numeric, e.g. “March”. In that case the user has to encode it into a number.

There are three words that introduce a predicate in their translation in logic. These predicates express a relation between two entities. A transitive verb links its subject and its object. A preposition (in a prepositional phrase) links the noun phrase of the prepositional phrase to the noun phrase to which it is attached. Finally, an adjective phrase as object of a copular verb, links the subject with the noun phrase that is part of the adjective phrase. This is the only place an adjective can occur in our grammar.

Finally there are two lexical categories which can be considered special to logic grid puzzles: comparatives and *some*-words. The latter are, in the puzzles we studied, only used as an unspecified integer and always appear as a quantity in a comparison (e.g. “somewhat more than John”).

B. A grammar for logic grid puzzles

A grammar was constructed based on the first ten logic grid puzzles from Puzzle Baron’s Logic Puzzles Volume 3 [6].

In total there are 49 grammar rules: 3 grammar rules for sentences, 20 related to nouns and noun phrases, 8 for verb phrases and 18 rules that connect the grammar with the lexicon.

Again, because of space constraints, we cannot show the full grammar. We limit ourselves to a short discussion.

Most grammar rules are quite general. They can be used outside logic grid puzzles as well. Others, like the grammar rule for a sentence with template “Of ... and ..., one ... and the other ...”, are specific for these types of puzzles. The introduction to [6] explicitly mentions this template and explains how this template should be interpreted. This interpretation is incorporated in the semantics of the grammar rule covering this template.

Some other more specific rules include an *all-different* constraint (“A, B, and C are three different persons”) and a numerical comparison (“John scored 3 points higher than Mary”) which introduces an addition or subtraction.

The semantics of most rules consists only of λ-applications. Some rules are more complex. Those exceptions are either because the scope of variables and negations would otherwise be incorrect or because the grammar rule is specific to logic grid puzzles and can not be easily explained linguistically. This linguistic shortcut is then visible in the semantics.

There are two type of noun phrases which relate two entities without a reference to this relation. Namely “the 2008 graduate” and “John’s dog”. In the current framework, it is impossible to derive which relation holds between “the graduate” and “2008”. In Section VII-A, we discuss a solution to this problem.

VI. Types

In natural language, it is possible to construct sentences that are grammatically correct but without meaning. E.g. “The grass is drinking the house”. The grass is not something that can drink and a house is not something that can be drunk. We say the sentence is badly typed. Based on grammar alone, we can never exclude these sentences. Therefore, we add types to the framework.

Concretely, we add a feature *type* to most words and phrases to indicate their type. Similarly to the feature *number* that some words and phrases have. The feature *number* indicates whether a word or phrase is singular or plural. In a sentence, the *number* of the noun phrase and the verb phrase must unify, i.e. the subject and the verb must agree in number. Similarly to this, the *type* of the noun phrase and the verb phrase must unify.

In this paper we use a very basic type system. There are a number of basic types and a *pair* type constructor that takes two basic types. E.g. a noun has a basic type, a transitive verb a pair of types: one for its subject and one for its object. Also phrases get types. A noun phrase gets a basic type. The same goes for a verb phrase. In the grammar we then express that the type of the noun phrase and the verb phrase should unify when they make a sentence. This excludes badly typed sentences from being accepted.

We also use these types in the semantics to indicate the types of the logical variables and of predicates. This allows the system to translate to a typed logic.

As indicated earlier, most words (like noun, verb, ...) also get a type in the lexicon. In this paper we explored a form of type inference. We assume that every word has exactly one type but that is not explicitly given to the system. We then search the types of all words. Based on these types, we can group the domain elements (represented by a proper noun) per domain (represented by a type), i.e. we can infer that "France" and "Italy" are from the same domain (without necessarily knowing that they are countries).

The search for the types of all words gets as input the number of domains in the puzzle (related to the number of base types) and some type constraints (from the translation of the clues into logic, based on the principle of one type per word). If many synonyms are used, the system enters an interactive mode and asks the user some linguistic questions that can be rewritten as "Is ... a meaningful sentence?" until the types are unified enough.

This process fails if (and only if) a certain domain always occurs as part of noun phrases with an unknown relation (e.g. "the 2008 graduate"). In such a scenario, the system cannot fallback on a linguistic question and therefore asks the user if two proper nouns are of the same type. This is the exact question the system was supposed to solve.

The described system can thus correctly derive which domain elements belong together based on type constraints and the answers to some linguistic questions. Only when they do not provide enough information, is the user asked to help identify the elements that belong together.

VII. A complete specification

In step 3 and 4 of the system described in IV, the system completes the specification with a formal vocabulary and the formulation of the implied axioms. We now describe these steps.

A. The formal vocabulary

We can construct the formal vocabulary based on linguistic information and type information.

A type that corresponds to a non-numerical domain of the puzzle is translated to a constructed type. A constructed type consist of a set of constants with two extra axioms implied: Domain Closure Axiom (DCA) and Unique Names Axiom (UNA). DCA states that the set of constants are the only possible elements of the domain. UNA states that all constants are different from each other.

The different proper nouns that belong to this type are the constants of the constructed type. However, it is possible that one constant is missing (i.e. it didn't occur in any of the clues). In that case, the system adds an extra constant. It is not possible that two elements are missing, because every puzzle has an unique solution and these two elements would be interchangeable.

A type that corresponds to a numerical domain is translated to a subset of the natural numbers. In that case, the system asks the user the exact subset as it often cannot be automatically inferred from the clues.

It is also possible to have an intermediate type that links two domains, e.g. *tour* in "John follows the tour with 54 people". These types are translated to constructed types with as many constants as there are domain elements in a domain. Symmetry-breaking axioms are added to the theory to link every constant with one domain element of another domain.

Every transitive verb and preposition introduces a predicate. The type signature of the predicate corresponds with the type pair of the word. Based on these signatures, we can infer the relation behind noun phrases like "the 2008 graduate". The predicate that links the two entities is the predicate that links the corresponding types.

B. The implied axioms

There are some axioms implied for a logic grid puzzle. They are not expressed in the clues because they hold for all logic grid puzzles.

The first type of axioms is the symmetry-breaking axioms mentioned in VII-A for intermediate types (E.g. $with(T1, 54) \wedge with(T2, 64) \wedge with(T3, 74)$). Another type is the bijection axioms to state that every predicate is a bijection (E.g. $\forall x \cdot \exists y \cdot with(x, y) \wedge \forall y \cdot \exists x \cdot with(x, y)$).

Finally there are three types of axioms to express that there is an equivalence relation between domain elements. Two elements are equivalent if they are linked through a predicate or if they are equal. In second order logic $x \sim y \Leftrightarrow \exists P \cdot P(x, y) \vee x = y$. The reflexivity is satisfied by definition.

- **Synonymy** There is exactly one bijection between every two domains. Therefore, two predicates with the same types are always synonyms. E.g. $\forall x \forall y \cdot livesIn(x, y) \Leftrightarrow from(x, y)$.
- **Symmetry** Predicates with a reversed signature are each other inverse. E.g. $\forall x \forall y \cdot gave(x, y) \Leftrightarrow givenBy(y, x)$.
- **Transitivity** Finally there are axioms linking the different predicates. They express the transitivity of the equivalence relation. E.g. $\forall x \forall y \cdot spokeFor(x, y) \Leftrightarrow \exists z \cdot gave(x, z) \wedge lastedFor(z, y)$.

VIII. Evaluation

A grammar was constructed based on the ten first logic grid puzzles from Puzzle Baron's Logic Puzzles Volume 3 [6] and evaluated on the next ten puzzles. Some rare and difficult constructs in the training set were modified to make it easier to parse. To fit the unseen puzzles to the constructed grammar, some clues of the test set were modified as well. The question arises how many of these modifications are necessary to represent the unseen puzzles in the constructed grammar. Another question is whether or not it is possible to deduce the types of a logic grid puzzle or not. Finally, we want to know if all puzzles are represented correctly such that the solution can automatically be derived with the IDP system [1].

It turns out that once modified to the grammar, the types of nine unseen puzzles can be derived automatically and the clues can be translated correctly into logic. For one puzzle the system asks the user if certain domain elements belong to the same domain. Thereafter, it does translate the puzzle correctly into logic.

Table II gives an overview of the different modifications. In total 65 modifications were necessary, 30 of which were purely grammatical. They used grammatical structures that didn't appear in the training set. The next 5 modifications were modifications similar to those from the training set. There are 19 modifications due to badly typed words. E.g. a verb "to order" that was used to order both food and drinks. The assumption of one type per word wasn't satisfied in that case. There was one modification to a proper noun because the same domain element appeared in two different word forms. We assumed there is only one word form per domain element. Finally, there were 10 cases where an extra word was added or removed.

Problem	**Count**
"the one"	15
Wrong use of copular verb	6
Badly structured subordinate clause	6
Passive sentence	1
Possessive pronoun	1
Badly structured noun phrase	1
Rational numbers	3
Superlative	1
"Of the two ..."	1
Badly typed verb	14
Badly typed noun phrase	5
More than one word form for a domain element	1
Redundant word	7
Missing word (ellipse)	3

TABLE II: An overview of the different modifications

We refer to the full master thesis for a detailed overview of all the modifications. However, most of the modifications are small and they were always necessary to fit them in the grammar. No modifications were necessary to avoid a parse that resulted in a wrong translation.

IX. Related work

In this section we discuss some related work. We begin with three CNL's, two for knowledge representation and one with types. Finally, we discuss another framework that is based on λ-calculus.

Two of the most advanced controlled natural languages that can be used for knowledge representation are Attempto Controlled English (ACE) and Processable English (PENG). ACE [7] is a general purpose CNL. The language contains a large built-in vocabularium. This has as advantage that the user doesn't have to provide the vocabularium. However, for specifications in small domains, we usually do want to provide the vocabularium to make sure the specification only talks about the modelled domain. PENG [8] does ask the user to provide its own content words. Moreover, there is a tool named ECOLE which gives suggestions while writing PENG sentences, namely about which linguistic constructs can follow the words already entered. This makes it easier to write correct PENG sentences. These languages mainly show that we can translate English into logic, but the literature on both these CNL's lacks a description of this translation process. Therefore, extending these CNL's is hard.

Another important CNL is RuleCNL [9]. It is a CNL that translates (a subset of) English into business rules. It's notable because it is the only CNL we could find that uses types. Interestingly, the paper lacks a description of the type system that was used and of the inferences that are supported. However figure 6 from [9] indicates that type checking of business rules in RuleCNL is supported.

Finally, Baral et al. [10], [11], [12] researched translating natural language into ASP programs. They do this based on λ-calculus and a Combinatorial Categorial Grammar (CCG). Each word gets one λ-ASP-expression per CCG category it can represent. The CCG then describes how to combine these expressions via λ-application. In their latest paper on this subject [12], Baral et al. explain how to learn both these λ-ASP-expressions and a probabilistic CCG grammar from a set of logic grid puzzles. They used a supervised machine learning method for this goal. Based on these methods they can solve unseen puzzles (without adapting the clues of the puzzles). They do not support other kind of inferences.

X. Conclusion

The framework of Blackburn and Bos can be used to translate a CNL into logic. This way, we assign semantics to such a CNL. Therefore, we can use such a CNL as a knowledge representation language within a knowledge base system. This allows different kinds of inference on the (constructed) natural language. For a logic grid puzzle, this can mean automatically solving the puzzle but also helping the user by giving hints or helping the author by giving the possible solutions.

Moreover, we expanded the framework with types. This allows us to translate the constructed CNL to a typed logic. We have proven that type inference is possible as well. More research into a typed CNL is definitely necessary. For example, what the value of a more complex type system is. Or if we can construct a CNL with a type system with multiple types per word. The type system can then derive the correct instance of the verb. E.g. it can differentiate between ordering drinks and ordering food.

References

[1] B. de Cat, B. Bogaerts, M. Bruynooghe, and M. Denecker, "Predicate logic as a modelling language: The IDP system," *CoRR*, vol. abs/1401.6312, 2014. [Online]. Available: http://arxiv.org/abs/1401.6312

[2] T. Kuhn, "A Survey and Classification of Controlled Natural Languages," *Computational Linguistics*, vol. 40, no. 1, pp. 121–170, mar 2014. [Online]. Available: http://www.mitpressjournals.org/doi/abs/10.1162/COLI_a_00168

[3] P. Blackburn and J. Bos, "Representation and inference for natural language," 2005. [Online]. Available: http://www.coli.uni-saarland.de/publikationen/softcopies/Blackburn:1997:RIN.pdf

[4] ——, "Working with discourse representation theory," *An Advanced Course in Computational Semantics*, 2006.

[5] Wikipedia, "Zebra Puzzle — Wikipedia, the free encyclopedia," http://en.wikipedia.org/w/index.php?title=Zebra%20Puzzleoldid=779870856, 2017, [Online; accessed 24-May-2017].

[6] S. Ryder, *Puzzle Baron's logic puzzles*. Indianapolis, Indiana: Alpha Books, 2016.

[7] N. E. Fuchs, K. Kaljurand, and T. Kuhn, "Attempto controlled english for knowledge representation," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5224 LNCS, pp. 104–124, 2008.

[8] R. Schwitter, "English as a Formal Specification Language," *Interpreting*, pp. 228–232, 2002. [Online]. Available: http://web.science.mq.edu.au/ rolfs/papers/n-lis2002.pdf

[9] P. B. F. Njonko, S. Cardey, P. Greenfield, and W. El Abed, "RuleCNL: A controlled natural language for business rule specifications," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8625 LNAI, pp. 66–77, 2014.

[10] C. Baral, J. Dzifcak, and T. C. Son, "Using Answer Set Programming and Lambda Calculus to Characterize Natural Language Sentences with Normatives and Exceptions," *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*, pp. 818–823, 2008.

[11] C. Baral, M. A. Gonzalez, and A. Gottesman, "The inverse lambda calculus algorithm for typed first order logic lambda calculus and its application to translating english to FOL," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7265, pp. 40–56, 2012.

[12] C. Baral and J. Dzifcak, "Solving Puzzles Described in English by Automated Translation to Answer Set Programming and Learning How to Do that Translation," *Knowledge Representation and Reasoning (KR)*, pp. 573–577, 2012.