User-Oriented Solving and Explaining of Natural Language Logic Grid Puzzles

Jens Claes[1], **Bart Bogaerts**[2], **Rocsildes Canoy**[2], **Tias Guns**[2]
[1] jensclaes33@gmail.com
[2] Vrije Universiteit Brussel, firstname.lastname@vub.be

Abstract

Inspired by the 2019 Holy Grail Challenge, we present ZEBRATUTOR, a mostly automated tool that solves a logic grid puzzle (also known as a Zebra puzzle) given the clues in natural language clues and a list of the entities in the puzzle. The challenge is to both handle the natural language input, and to produce a human-understandable explanation of how the solution is obtained. We achieve this by translating the natural language clues into logic using a typed version of the semantical framework of Blackburn and Bos. The logical representation is then used in a novel explanation-based reasoning procedure, on top of the IDP knowledge base system. A novel aspect of the explanation is that it is ordered by mental effort required to understand the reasoning step, which is estimated by the number of previously derived facts needed to derive new facts. The outcome is a stepwise visualisation of the clue(s) used and the resulting changes on the grid. This can be used both to solve a puzzle, or as a step-wise 'help' function for people stuck while solving a puzzle.

1 Introduction

Automated problem solving is central to the research field of Artificial Intelligence. A common assumption here is that the problem is unambiguously specified as a problem specification (sometimes called *model*) in a formal language. However, formulating such a specification is non-trivial, and for human beings it is much easier to specify problems in natural language.

Natural language processing (NLP) (Manning, Manning, and Schütze 1999) is the subfield of AI concerned with processing and ultimately understanding natural language. Problem solving starting from natural language input has been studied for simple mathematical problems in the subfield of *word problem solving* or natural language mathematical problems (Mukherjee and Garain 2008).

We here aim to study natural language combinatorial problems, both from human-produced input (e.g. natural language) and for human-understandable output (in our case, visualisations), as per the Holy Grail 2019 challenge.[1] Generating human-understandable output to logic grid puzzles has initially been explored by Sqalli and Freuder (1996).

We present ZEBRATUTOR, an end-to-end solution for solving logic grid puzzles (also known as Zebra puzzles) and for explaining, in a human-understandable way, how this solution can be obtained from the clues.

ZEBRATUTOR[2] starts from a plain English language representation of the clues and a list of all the entities present in the puzzle. It then applies NLP techniques to build a puzzle-specific *lexicon*. This lexicon is fed into a type-aware variant of the semantical framework of Blackburn and Bos (2005; 2006), which translates the clues into discourse representation theory (Kamp 1988). This logic is further transformed to a specification in the IDP language (De Cat et al. 2016), an extension of first-order logic. The solver underlying IDP (called MINISAT(ID)) is a lazy-clause generation solver supporting among others recursive definitions (De Cat et al. 2013).

It then uses this formal representation of the clues both to solve the puzzle and to explain the solution. There are many different ways in which such a system could explain itself. For instance, after finding a solution, it can explain 1. *why that is a solution* or 2. *why there are no other solutions*; additionally, it can explain 3. *how the system found this solution*, and 4. *how a human could find this solution*.

Our system implements this last type of explanation. In contrast to how the system found the solution, it focuses on simplifying the explanation itself, rather than giving insights in the actual search algorithm used by IDP. As such, our explanations are not to be used for understanding the inner workings of the solver, but rather are to be used by people interested in solving logic puzzles.

In generating explanations and choosing the *order* in which the reasoning steps are explained, we chose to order by an estimate of mental effort required to follow the reasoning step. Each reasoning step is visualised as the clue(s) involved and the resulting changes on the grid.

When solving such puzzles, it can either be used for explaining how to obtain an entire solution, or for providing help to users who are stuck during the solving process. Indeed, our explanation method will, given a partial solution, find the easiest next derivation to make.

[1] https://freuder.wordpress.com/pthg-19-the-third-workshop-on-progress-towards-the-holy-grail/

[2] ZEBRATUTOR is an extension of software originally developed in the context of a Master's thesis (Claes 2017).

2 System Overview

The **input** to ZEBRATUTOR is a set of natural language sentences (from hereon referred to as "clues"), and the names of the *entities* that make up the puzzle, e.g. Englishman, red house, Zebra, etc.

In typical logic grid puzzles, the entity names are present in the grid that is supplied with the puzzle. For some puzzles, not all entities are named or required to know in advance; a prototypical example is Einstein's Zebra puzzle, which ends with the question "Who owns the zebra?", while the clues do not name the Zebra entity, and the puzzle can be solved without knowledge of the fact that there is a zebra in the first place.

Steps Our framework consists of the following steps, starting from the input:

A Part-Of-Speech tagging: with each word a part-of-speech tag is associated.

B Chunking and lexicon building: a problem-specific lexicon is developed.

C From chunked sentences to logic: using a custom grammar and semantics a logical representation of the clues is constructed.

D From logic to a complete IDP specification: the logical representation is translated into the IDP language and augmented with logic-grid-specific information.

E Explanation-generating search in IDP: we exploit the IDP representation of the clues to search for simple explanations as to how the puzzle can be solved.

F Visualisation of the explanation: the step-by-step explanation is visualized.

The first three of these steps are related to natural language processing and are discussed in detail in the next section. Step D is explained in Section 4; there we also briefly explain how the representation obtained at that point can be used to automatically solve the puzzle. The last two steps are related to explanations and are presented in Section 5.

3 Natural Language Processing

Step A. Part-Of-Speech tagging

The standard procedure in Natural Language Processing is to start by tagging each word with its estimated Part-Of-Speech tag (POS tag).

We use the standard English Penn Treebank II POS tagset (Marcus, Santorini, and Marcinkiewicz 1993). As POS tagger we use NLTK's built-in Perceptron tagger[3]. It uses a statistical inference mechanism, trained on a standard training set from the Wall Street Journal. Since any POS-tagger can make mistakes, we make sure that all of the puzzle's entities are tagged as *noun*.

[3]http://www.nltk.org

Step B. Chunking and lexicon building

To use the Blackburn & Bos framework (Blackburn and Bos 2005; 2006), a *lexicon* and a *grammar* have to be provided, where the lexicon assigns a role to different sets of words, and the grammar is a set of rules describing how words can be combined into sentences. The grammar is constructed for logic grid puzzles in general and not puzzle specific; the lexicon is partly problem agnostic and partly puzzle-specific.

We constructed a set of 12 lexical categories (Claes 2017). The 3 puzzle-specific categories are: *proper nouns*, namely the individual *entities* that are central to the puzzle, *other nouns* that refer to groups of entities (like house, animal) and *transitive verbs* that link two entities to each other.

The other categories are general and contain a built-in list of possible members. The categories are determiner, number, preposition, auxiliary verb, copular verb, comparative and *some**-words (somewhat, sometime, ...), and conjunction. Because of space constraints, we refer the reader to the full master thesis for the full details on these categories.

The goal of this second step is hence to group the POS-tagged words of the clues into *chunks* that are tagged with one of the above lexicon categories. This process is known in the NLP community as *chunking*. We use NLTK and a custom set of regular expressions for chunking the proper nouns and different types of transitive verbs.

The result is a lexicon, which is needed as input to the subsequent step in our framework. However, the POS tagging may be inaccurate and the chunking may also miss certain cases. Furthermore, logic puzzle authors are keen to use word play or seemingly ambiguous sentences to make the puzzle more interesting, but that require general world knowledge. For example, using 'in the morning' to refer to a timeslot at 11:00 when all other timeslots are after 13:00.

The automatically generated lexicon is hence tested as described in the next step, and if some clues can not be transformed into logic, the user is asked to update the lexicon or rewrite a clue. For example, to replace 'in the morning' by 'at 11:00' in the earlier example.

Step C. From chunked sentences to logic

The Blackburn and Bos framework requires a lexicon (discussed in the previous paragraph) and a grammar, each equipped with a suitable semantics. 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.

We constructed a grammar based on the first ten logic grid puzzles from Puzzle Baron's Logic Puzzles Volume 3 (Ryder 2016). We observe that the determiners in the 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 entities. 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. Sen-

tences like "No person drinks tea" do not occur in the puzzles we studied.

Since it would distract us from the core message of this paper, we do not detail the actual grammar rules here but refer the the Master thesis this paper builds on (Claes 2017) for more information.

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 the logigram booklet (Ryder 2016) 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 *alldifferent* constraint ("A, B, and C are three different persons") and a numerical comparison ("John scored 3 points higher than Mary").

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.

Compared to the original Blackburn and Bos framework (Blackburn and Bos 2005; 2006), we added type information. 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.

The output of the framework is a (typed variant of) discourse representation theory (Kamp 1988).

4 Solving Logic Grid Puzzles

Before detailing how we construct a complete specification, we give a very brief introduction to the IDP system. More information can be found in (De Cat et al. 2016).

Preliminaries: the IDP system

The IDP system (De Cat et al. 2016) is a knowledge base system (Denecker and Vennekens 2008) for a rich extension of first-order logic. In practice, when problem-solving with IDP, one typically makes use of the following components: vocabularies, theories, structures, inference methods, and procedures. A vocabulary is (as in standard first-order logic) a set of symbols. In IDP these symbols are furthermore *typed*. A (complete) structure over a vocabulary is an assignment of values (of the right type) to symbols (for instance, a set of tuples to a predicate symbol), this is typically called a variable assignment or candidate solution in CP. A *partial* structure may in addition contain partial information, such as $(1, 1)$ is an element of the interpretation of P and $(2, 1)$ is not, without fully specifying the interpretation of P; in a CP context, it can be seen as the current *domain* of each symbol. Partial structures are ordered according to *precision*. Intuitively, S_1 is more precise than S_2 (notation $S_1 \geq_p S_2$) if S_1 contains at least all information S_2 contains. A *theory* in IDP is an expression in an extension of first-order logic. It represents a piece of information (for instance, a clue) but does not represent a problem; it can be seen as a (sub)set of constraints in CP. In order to solve a problem using this knowledge, we make use of *inference methods* (solvers), which are generic algorithms that tackle a task given some of the aforementioned components. Some commonly used inference methods are:

- **modelexpand(T,S)** This method takes as input a theory and a partial structure S, it returns one or more (depending on options) structures that are more precise than S and that satisfy T.
- **optimalpropagate(T,S)** This inference method takes as input a theory T and a partial structure S, it returns the most precise partial structure S' that approximates all models of T that expand S (i.e., such that for each model M of T with $M \geq_p S$, it holds that $S' \leq_p M$). Thus, S' contains all consequences derivable from T starting from the structure S.
- **unsatstructure(T,S,V)** This inference method takes as input a theory T and a structure S and optionally a vocabulary V, whereby T has no models more precise than S, that is, the combination of T and S has no solution. It returns structure S' less precise than S, but equal to S outside V such that T still has no satisfying solution more precise than S'; furthermore the returned structured is minimally precise among such structures. Intuitively, this inference methods finds the reason for the inconsistency: it explains why there are no models of T expanding S by identifying a minimal set of facts in S that cause the lack of satisfying solutions. Internally, this is implemented using unsatisfiable core extraction (Lynce and Silva 2004).

Finally, IDP provides built-in support for the lua scripting language, where all above components are first-class citizens. Lua procedures are typically used to glue together different calls to different inference methods to solve an actual problem.

Step D. From logic to a complete IDP specification

In order to build a complete specification of the puzzle from the Discourse Representation Theory(DRT) returned by the typed Blackburn and Bos framework, we compute the interpretation of the different types. As mentioned before, the list of entities occurring in the puzzle needs to be given to build the lexicon. If additionally they are also partitioned in types (this information can e.g., be taken from a grid-representation), nothing else needs to be done here. If the partitioning of the entities in types is *not* given, we use **type inference** to compute an equivalence relation on the set of proper nouns occurring in the clues (two proper nouns are equivalent if they occur in the same position of a verb/preposition; for instance if "the Englishman smokes cigarettes" and "the person who owns a dog does not smoke cigars" we derive that cigars and cigarettes are nouns of the same type). It might happen that this does not yield enough information to completely determine the types for two reasons. First of all, not all proper nouns might occur in the

clues (for instance the zebra in Einsteins famous zebra puzzle). However, since the solution of a logic grid puzzle is always unique, there can at most be one such missing entity per type (otherwise by symmetry there would be multiple solutions) and hence, we can then simply add an anonymous element. Secondly, there might be a large variation in the verbs used to denote the same relation. In that case, without using knowledge on the partitioning of entities, we cannot decide which entities belong to the same type. Our system then queries the user to ask which verb are – for the purpose of the puzzle – synonyms.

Once the types are completed, we construct the IDP vocabulary containing: all the types and a *relation* for each transitive verb or preposition. For instance if the clues contain a sentence "John lives in the red house", then the vocabulary will contain a binary relation $livesIn(\cdot,\cdot)$ with the first argument of type $person$ and the second argument of type $house$. Additionally, we ensure that there is at least one relation between each two types, even if this relation does not occur in the clues. This is not important for solving the puzzle, but it plays an important role in explaining (more on that follows in the next section). The interpretation of the types is encoded in IDP by means of 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.

After the vocabulary, we construct IDP theories:

- we translate each clue into the IDP language, and
- we add implicit constraints present in logic grid puzzles.

The implicit constraints are stemming from the following: First of all, our translation might generate multiple relations between two types. For instance if there are clues "The tea drinker is from France" and "The person who owns a dog lives in England", then the translation will create two relations $from$ and $livesIn$ between persons and countries. This happens regularly since logigram designers tend to vary their vocabulary to keep the puzzles interesting. However, we know that there is only one relation between two types, hence we add a theory containing *synonymy* axioms; for this case concretely:

$$\forall x \forall y : livesIn(x,y) \Leftrightarrow from(x,y).$$

Similarly, if two relations have an inverse signature, they represent the inverse functions (for instance $isOwnedBy$ and $likes$) in the clues "The Englishman likes cats" and "The dog is owned by the Belgian"). In this case we add constraints of the form

$$\forall x \forall y : likes(x,y) \Leftrightarrow isOwnedBy(y,x).$$

Next, we add axioms that state that each relation between two types is actually a *bijection*, e.g.

$$\forall x : \#\{y \mid from(x,y)\} = 1.$$
$$\forall y : \#\{x \mid from(x,y)\} = 1.$$

Finally, we add *transitivity* axioms that link the different relations. For instance if the dog is kept in the red house and the Englishman lives in the red house, then the Englishman keeps a dog. This kind of axioms is expressed as:

$$\forall x \forall y \forall z : keptIn(x,y) \wedge livesIn(z,y) \Rightarrow keeps(z,x).$$

Solving the puzzle The conjunction of all the logical theories created in the previous paragraph completely characterizes the constraints underlying a logic grid puzzle. In order to solve the puzzle, we use IDP's built-in *model expansion* inference, which searches for a solution in a given finite domain. Under the hood, IDP uses MINISAT(ID) (De Cat et al. 2013), a solver using SAT (Marques Silva, Lynce, and Malik 2009) and CP (Apt 2003) technology, in particular lazy clause generation (Stuckey 2010) and conflict-driven clause learning (Marques-Silva and Sakallah 1999). In our experience, the solving part is often quite trivial for logic grid puzzles, since they are usually crafted to be solvable using the grid and single clues.

5 Explaining Logic Grid Puzzles

Step E. Explanation-generating search in IDP

A part of the holy grail challenge was not to just solve the puzzle from the natural language specification, but also to *explain* how the solution was obtained, or rather, to explain how a human could obtain this solution as well.

The idea of our explanation approach is that we will gradually fill the logigram grid with more and more information. The explanation is hence a series of reasoning steps, and at each step, information is added that can be obtained by reasoning on the clues and by using the partial solution obtained so far.

Simple Explanations In order to make the explanations as simple as possible we order the reasoning steps (derivations) in the following way:

- Derivations that can be made without using any clues are always prioritized. These derivations are made solely using logigram axioms, such as the fact that all involved predicates are bijections as explained above.
- Next, derivations that require one or more clues are executed, where fewer clues are preferred.
- Within both of the above classes, we always prioritize derivations that can be made by as little as possible information, that is, using as few as possible of the fields in the grid that have been filled already (see below for details on how to compute this).

It deserves to be noted that in the examples we encountered, it sufficed to only consider derivations that use at most one clue. This is probably due to puzzle designers making their puzzles easy enough. However, we can craft artificial examples in which that does not work, as illustrated next.

Example 5.1. Consider the following two clues of a logic puzzle.

> "Either the Englishman lives in the red house with the fish, or he lives in the green house with the dog. "
> "If the Russian is a tea drinker, then the Englishman keeps a dog in the red house."

From these two clues, it follows that the Russian is not a tea drinker, but this can not be obtained by isolated reasoning over the two clues separately and at each step only deriving information that is present in a logic puzzle grid. ▲

Implementation In order to implement this, we ensured that our automatic translation contains a separate logical theory for each clue, so that we can reason over the clues separately. For this, we made use of IDP's procedural interface (based on the lua scripting language) in which theories and structures are first-order citizens, and of the built-in inference methods described above.

These methods are used as follows. Our procedure maintains a partial structure S representing the current state of the grid.[4] At each point in time, for all sets of clues of a given size n (starting with $n = 0$), all consequences (value assignments) of the conjunction of this set of clues are computed with **optimalpropagate**. If there are no consequences, the same is repeated for a greater n. For each of the consequences, a minimal set of facts from the partial structure S that entail this consequence is computed using **unsatstructure** (this is done by making the consequence false in S and using V to disallow changing this consequence in the outcome of the unsatstructure-call). The result is a set of pairs $(S', clues, fact)$ where S' is a substructure of S such that from the set of clues $clues$ and S' the fact $fact$ followed; for all computed facts. Among those, a cardinality-minimal set S' is selected and its propagation is executed and added to the list of derivations that make up the explanation. Afterwards, the procedure starts over with $n = 0$.

Propagation Strength One important thing to note here is that when we apply propagation, we do not do it on a theory that *only* contains the clues in question. Instead, that theory always also contains all logigram axioms (bijections, transitivity, etcetera). The reason is that clues by themselves rarely propagate. To illustrate this, let us consider some examples.

Example 5.2. Consider the clue "The patient who was prescribed enalapril is not Heather". If one were to manually translate this clue into first-order logic over the given vocabulary, one would probably come up with:

$$\neg prescribed(heather, enalapril). \quad (1)$$

However, our automatic parsing method makes an explicit mention of "the person who" in the form of a logic variable and instead produces

$$\exists p : prescribed(p, enalapril) \wedge p \neq heather. \quad (2)$$

If the system were given the clue "Heather was not prescribed enalapril", the system would find (1). Equations (1) and (2) are not equivalent and in fact from (2) it does not follow that heather is not prescribed enalapril. That is... unless the fact that there is exactly one person who is prescribed enalapril is taken into account. In conjunction with the theory stating that all predicates are bijections, these two equations are equivalent. ▲

Example 5.3. Consider the clue "The owner of the lime house was prescribed enalapril", which is translated into first-order logic as:

$$\exists o : lives\_in(o, lime) \wedge prescribed(o, enalapril). \quad (3)$$

This clue actually gives information about the relation between houses and medication. However, it is clear that from (3) alone we cannot propagate such information: it does not even mention the predicate that links houses and medications. However, in combination with the transitivity and bijection axioms, we can propagate that

$$\neg used\_in(enalapril, lime). \quad ▲$$

F. Visualisation of the explanation

We explored two different possibilities to present this explanation to humans. The first was generating natural language sentences of the form "From the clue(s) ⟨clue⟩ and the fact that ⟨assumptions⟩, it follows that ⟨conclusions⟩." However, in our experience, as soon as there are a couple of assumptions involved, this kind of sentence easily becomes hard to read and understand. Furthermore, it is not always easy to create these sentences: if the input does not mention a name for the relation between medication and houses, how can we express that enalapril is *not used in* the lime house? There are two possibilities to do that, the first is using a generic verb such as "associated with" (rendering boring sentences); the second is avoiding the relation and for instance writing "the user of enalapril does not live in the lime house".

Overall, we were not satisfied with the outcome of this first approach, which is why we implemented a different way to present the explanation process to the user, namely by means of a visualisation. To represent our derivations, we make use of the standard grid in logic puzzles. In each step, we indicate which clue(s) are used, highlight all cells used for the propagation in blue and all conclusions in green. The user can then navigate through the reasoning process by means of "next" and "previous" buttons.

Figure 1 contains a screenshot of this explanation process. It displays a partially filled grid, in which checkmarks represent that something is derived to be true and minus signs that it is false. In this specific frame, we can see that the clue "Roxanne is 2 years younger than the Kansas native" is used, together with the previously derived knowledge (highlighted in orange) that the Kansas native is not 111 years old, we can derive (highlighted in blue) that Roxanne is not 109 years old.

6 Demonstration

The working of our system is demonstrated on http://bartbog.github.io/zebra. This webpage contains for some puzzles:

- All the the clues, and which (minor) modifications to the natural language formulation we implemented.

[4] Since such a structure interprets all the predicates, which are binary relations between two types, a partial structure indeed corresponds nicely to a partially filled grid.

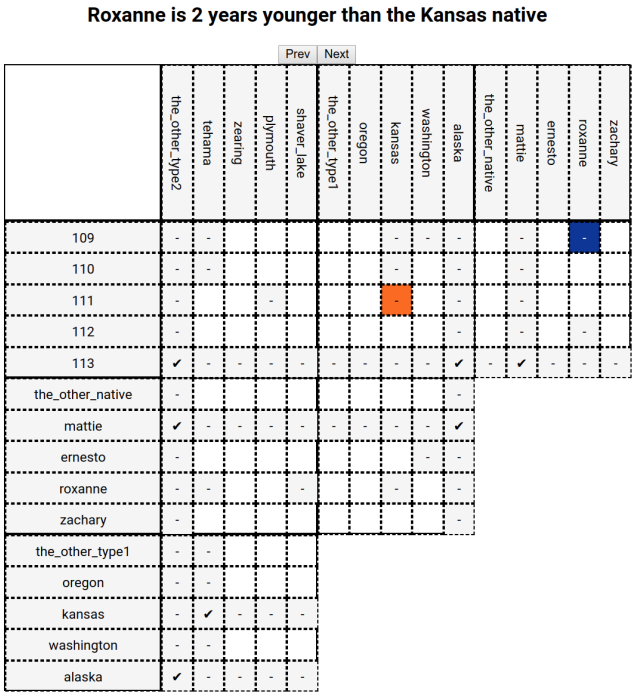

Figure 1: Screenshot of the visualization.

- The lexicon that is required to parse the puzzles (semi-automatically generated).
- The resulting idp theory associated to each of the clues.
- Runnable IDP files to either solve the puzzle, or generate the explanations.
- The visualization of the explanation by derivation steps.

The website is still under construction and will be updated with more puzzles in the near future.

7 Conclusion

In this paper, we presented ZEBRATUTOR, a tool focused on explaining the solution process of logic grid puzzles. For the explanation part, we developed a method that prioritizes derivation steps according to the mental effort required to understand them. This tool has the potential to work end-to-end, but preliminary experiments show that due to inaccuracies or ambiguities in the NLP parts, manual intervention in the lexicon construction or clue formulation may be needed.

Future work includes combining the NLP interpretation with the solving, so that different interpretations are tried if the corresponding logic formulation does not yield a solution. Another interesting avenue is to expand the work on explanations to other satisfaction problems, where the abstraction in terms of high-level *clues* may be less obvious.

References

Apt, K. R. 2003. *Principles of Constraint Programming*. Cambridge University Press.

Blackburn, P., and Bos, J. 2005. Representation and inference for natural language.

Blackburn, P., and Bos, J. 2006. Working with discourse representation theory. *An Advanced Course in Computational Semantics*.

Claes, J. 2017. Automatic translation of logic grid puzzles into a typed logic. Master's thesis, KU Leuven, Leuven, Belgium.

De Cat, B.; Bogaerts, B.; Devriendt, J.; and Denecker, M. 2013. Model expansion in the presence of function symbols using constraint programming. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, November 4-6, 2013*, 1068–1075. IEEE Computer Society.

De Cat, B.; Bogaerts, B.; Bruynooghe, M.; Janssens, G.; and Denecker, M. 2016. Predicate logic as a modelling language: The IDP system. *CoRR* abs/1401.6312v2.

Denecker, M., and Vennekens, J. 2008. Building a knowledge base system for an integration of logic programming and classical logic. In García de la Banda, M., and Pontelli, E., eds., *ICLP*, volume 5366 of *LNCS*, 71–76. Springer.

Kamp, H. 1988. Discourse representation theory: What it is and where it ought to go. In Blaser, A., ed., *Natural Language at the Computer, Scientific Symposium on Syntax and Semantics for Text Processing and Man-Machine-Communication, Heidelberg, FRG, February 25, 1988, Proceedings*, volume 320 of *Lecture Notes in Computer Science*, 84–111. Springer.

Lynce, I., and Silva, J. P. M. 2004. On computing minimum unsatisfiable cores. In *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings*.

Manning, C. D.; Manning, C. D.; and Schütze, H. 1999. *Foundations of statistical natural language processing*. MIT press.

Marcus, M. P.; Santorini, B.; and Marcinkiewicz, M. A. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics* 19(2):313–330.

Marques-Silva, J. P., and Sakallah, K. A. 1999. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers* 48(5):506–521.

Marques Silva, J. P.; Lynce, I.; and Malik, S. 2009. Conflict-driven clause learning SAT solvers. In Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press. 131–153.

Mukherjee, A., and Garain, U. 2008. A review of methods for automatic understanding of natural language mathematical problems. *Artificial Intelligence Review* 29(2):93–122.

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

Sqalli, M. H., and Freuder, E. C. 1996. Inference-based constraint satisfaction supports explanation. In *AAAI/IAAI, Vol. 1*, 318–325.

Stuckey, P. J. 2010. Lazy clause generation: Combining the power of SAT and CP (and mip?) solving. In Lodi, A.; Milano, M.; and Toth, P., eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 7th International Conference, CPAIOR 2010, Bologna, Italy, June 14-18, 2010. Proceedings*, volume 6140 of *Lecture Notes in Computer Science*, 5–9. Springer.