A tool set for converting Controlled Natural Languages into AI formalisms*

Simone Caruso[1]

[1]University of Genova, Italy

Abstract

Languages for Knowledge Representation and Reasoning, such as ASP, CP, and SMT, excel at solving complex combinatorial problems. However, their use typically requires expertise in formal logic and programming, which poses a barrier for domain experts and non-specialists. Controlled Natural Languages (CNLs) offer a promising solution by providing a restricted subset of natural language that is both human-readable and machine-interpretable. This paper presents a tool set designed to convert CNLs into formal knowledge representation languages, enabling broader accessibility and usability of these powerful reasoning systems. In particular, we introduce two tools that translate CNL into ASP and TELINGO input language, and a third tool that facilitates the definition of CNLs and their mappings into corresponding target formalisms. Together, these tools support the use of formal knowledge representation paradigms through structured natural language, lowering the barrier for non-experts.

Keywords

Controlled Natural Language, Answer Set Programming, Constraint Programming, Satisfiability Modulo Theories

1. Introduction

Answer Set Programming (ASP) [1], Constraint Programming (CP) [2], and Satisfiability Modulo Theories (SMT) [3] are three powerful and widely used approaches for solving complex combinatorial problems. Each offers a distinct approach to modeling and solving problems. ASP is a declarative programming rooted in logic programming and non-monotonic reasoning. ASP solvers (see, e.g., [4, 5, 6]) generate solutions, called answer sets, that satisfy a given logic program. It is particularly well-suited for problems requiring reasoning with incomplete or evolving information, such as knowledge representation, reasoning, and AI applications [7], recently particularly in the Healthcare sector [8, 9, 10, 11, 12, 13, 14, 15], possibly in combination with machine learning and LLM techniques (see, e.g., [16, 17, 18]). ASP also presents several extensions introduced in the past years, such as TELINGO, which supports temporal operators, CLINGO[DL] that provides a seamless way to integrate linear constraints, or CASP that integrates Answer Set Programming with constraint processing. CP focuses on defining problems in terms of variables, the domains they can take, and the constraints that must be satisfied. The goal is to find assignments to variables that satisfy all the specified constraints, making CP particularly effective for problems with intricate combinatorial structures, such as scheduling, planning, and resource allocation [19, 20]. SMT extends Boolean satisfiability (SAT) by incorporating more expressive theories like arithmetic, bit-vectors, and arrays [21]. SMT solvers (see, e.g., [22, 21**?**]) determine whether logical formulas are satisfiable within these theories, making them particularly effective in fields like formal verification, model checking, and software synthesis, where mathematical precision is crucial. Despite their unique approaches, CP, SMT, and ASP share a common purpose: solving complex combinatorial problems and they can be seen as complementary formalisms within the same broader framework of declarative problem-solving, where each formalism excels in different problem domains, yet they can be integrated or used in tandem to leverage their respective strengths.

Joint Proceedings of the Workshops and Doctoral Consortium of the 41st International Conference on Logic Programming, September 9–13, 2025, Rende, Italy

*Doctoral project supervised by Marco Maratea and Carmine Dodaro (University of Calabria).

simone.caruso@edu.unige.it (S. Caruso)

0000-0002-2724-4342 (S. Caruso)

Although they are able to solve complex combinatorial problems and their wide usage in several domains, their usage is still problematic for people without a mathematical or logical background. Thus, to make these technologies more accessible, Controlled Natural Languages (CNLs) have emerged as a promising solution. CNLs are simplified versions of natural language with restricted grammar and vocabulary, designed to reduce ambiguity and complexity. They offer a bridge between formal systems and human-readable specifications, allowing domain experts to describe complex problems in a more intuitive and accessible manner without requiring deep expertise in the underlying formalism.

However, the task of defining a CNL and its mapping into a target formalism, from a technical perspective, is a time-consuming and challenging process. Indeed, creating a custom CNL involves designing a grammar, creating and processing a corresponding Abstract Syntax Tree (AST), and finally, generating code that translates the natural language input into the specific formalism. The first step requires defining a set of syntactic rules capable of capturing the nuances of the language while maintaining precision and clarity. The grammar must be comprehensive enough to cover the wide variety of constructs that users may wish to express, yet restricted enough to avoid ambiguity. Once the CNL input is parsed according to the grammar, it must be converted into an AST. This stage is complex because it involves resolving ambiguities, managing scope, and ensuring compliance with all semantic rules of the CNL. The final step involves translating the AST into code that conforms to the specific formalism, whether it be ASP, CP, or SMT. This translation process is intricate, as it must carefully map the high-level CNL constructs to their corresponding representations in the target language. To address these challenges, we developed a tool set designed to bridge the gap between formal methods and human-readable specifications. As a starting point, we introduced a new CNL along with a tool called CNL2ASP, which translates CNL input into Answer Set Programming. We then extended CNL2ASP to support the TELINGO input language, thereby enabling the use of temporal operators. Finally, we created CNLWizard, a novel framework aimed at simplifying the development of CNLs for various knowledge representation formalisms.

2. Background

Several CNLs have been proposed and used in several domains. For instance, [23] introduced a CNL which can be translated into various formalisms: linear time logic (LTL), computational tree logic, graphical interval logic, metric temporal logic, timed computational tree logic, and real-time graphical interval logic. This CNL has been applied in different practical applications and domains. [24] tested its applicability in the automotive domain, while [25] employed it to verify industrial system requirements via SMT-based translations. Similarly, [26] and [27] converted CNL constraints into LTL formulae, while [28] used a CNL to specify embedded systems specifications for the automotive industry in natural language, subsequently translating them into Boolean expressions and checking their consistency using Z3.

In the field of logic programming, one of the first approaches was proposed by [29] and [30], resulting in Attempto CNL [31], a CNL designed to translate natural language sentences into Prolog clauses. [32] presented a computer-processable language designed to be more accessible for computers than for human users. [33] proposed BIOQUERYCNL, a CNL for expressing biomedical queries, along with an algorithm designed to automatically convert this biomedical query expressed in this language into an ASP program. BIOQUERYCNL is a subset of Attempto CNL that can represent queries, and it was also used as a basis to generate explanations of complex queries [34]. [35] proposed a CNL specific for solving logic puzzles, whose idea is to have two sets of sentences, namely *Puzzle Domain data* and *Puzzle clues*, which are then converted into ASP rules. More recently, [36] defined the language PENG^{ASP}, which is similar to our CNL2ASP tool. However, PENG^{ASP} is specifically designed to express combinatorial problems in a way that feels natural yet remains unambiguous and reliably translatable into ASP. To achieve this, it uses words that stand out during reading and whose meanings can be easily inferred from context. This design choice enhances the predictability in the translation but comes at the cost of the naturalness of the language. Although CNLs are widely used, they are rarely

made publicly available. This limits their practical use and makes it challenging to conduct meaningful comparisons. Motivated by this, we have made our tools open source and publicly available to support broader adoption and facilitate reproducible research.

CNLWizard is a grammar-based system with a scope similar to the traditional compiler-compiler frameworks such as the Cornell Program Synthesizer [37]. However, while these frameworks generate parsers and compilers for formal programming languages through syntax-directed grammars and semantic rules, CNLWizard focuses on translating CNLs into KR formalisms. Moreover, unlike the interactive, template-based editing of the Synthesizer that guides users in building correct programs, CNLWizard generates a grammar using commands provided in a YAML-based format and processes a complete CNL specification. Though both are grammar-driven, CNLWizard serves domain experts modeling knowledge, not programmers building executable systems.

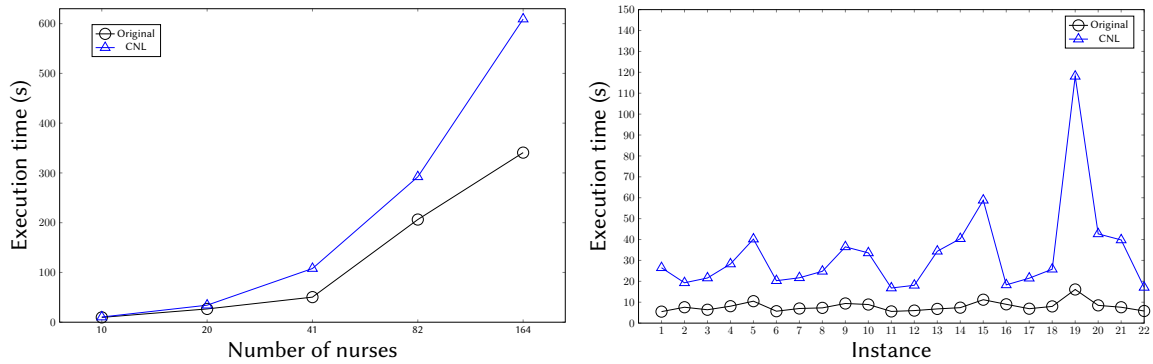
Another line of related work concerns LLMs, even if they differ fundamentally from CNLs. Indeed, while LLMs show generally good performance, they remain unreliable for (consistently) generating correct logic programs due to their probabilistic nature. As a result, they lack the precision and reliability required in knowledge representation tasks. This is crucial for logic programming languages, which are highly sensitive to both syntax and semantics, where even small mistakes, common in LLM-generated output, can result in programs that are either unusable or, worse, incorrect. In contrast, CNLs ensure precision through grammar-based determinism, making them more suitable for such tasks. For instance, [38] suggested that LLMs such as GPT-3 can function as few-shot semantic parsers, transforming natural language into logical forms for ASP. However, as noted by the authors, some results remain unpredictable, and the LLM does not always behave as intended. Other works focus on specific tasks, as [39], which translates NL sentences into ASP facts, or [40], which supports some simple patterns. Thus, despite their promise, LLMs are not yet capable of reliably producing arbitrary KR programs. While CNLs continue to offer a more consistent alternative, another promising direction is the approach proposed by [18], where natural language sentences are translated into ASP using our CNL2ASP system.

3. Goal of the Research

This research explores the integration of CNLs with formal reasoning paradigms such as ASP and its extensions, CP, and SMT. These paradigms are well-established for their expressiveness and effectiveness in solving complex combinatorial and logical problems. However, their usage outside expert communities remains limited, primarily due to the steep learning curve associated with formal syntax and semantics. On the other hand, CNLs offer a bridge between human-readable specifications and formal methods. They enable domain experts to formalize complex problems in an intuitive and accessible manner, without requiring in-depth knowledge of the underlying computational formalisms. The integration of CNLs with formal reasoning systems offers several benefits, as highlighted in earlier work by Clark et al. [32] and more recently by Caruso et al. [41]. These include:

- enhanced accessibility, as CNLs lower the entry barrier for non-experts, allowing them to participate in problem formulation, as they can express constraints, rules, and requirements in a language close to the natural one;
- reduced errors by minimizing ambiguity, since CNLs help to prevent misinterpretations and errors in problem specifications, leading to more accurate solutions;
- improved collaboration, as CNLs facilitate the communication between interdisciplinary teams, including domain experts and developers;
- fast prototyping, as experts can intuitively formulate problem encodings in natural language, which can then be further optimized;
- improved performance of natural language processing tools, as recently shown by [18].

Thus, the goal of this research is to develop a tool set able to integrate knowledge representation formalisms with CNLs in order to make these systems more accessible and usable to a wider audience, including those without expertise in logic or formal methods.



(a) NSP (b) CTS

Figure 1: Time comparison of the performance of the original and the CNL encodings.

4. Current Status

Following the goal directions mentioned above, we have developed three tools: CNL2ASP, CNL2TEL, and CNLWizard. CNL2ASP was developed to convert controlled natural language sentences into ASP. To this scope, we first defined a specification language intending to be general enough to represent problems from several (possibly all) domains in a natural way. Then, we developed a tool called CNL2ASP to convert such language into ASP as a first target formalism.

The tool takes as input a file containing a list of statements written in CNL and produces as output the translated ASP program. A specification written in this CNL is made of propositions, the structure of which is defined by clauses, linked by connectives, that are used to express concepts, to query them for information, or to express conditions on them. The combination of clauses that produces a proposition defines its type, which is used to understand what the proposition is supposed to mean and how that meaning can be translated into ASP rules and facts. More in detail, CNL2ASP is made of four main components, namely the *Parser*, the *Compiler*, the *Intermediate Data Structure*, and the *ASP Rewriter*. Each CNL proposition in the input file is processed by the Parser, whose role is to tokenize the CNL statements and construct a parsing tree, then the *Compiler* visits this tree bottom-up and initializes an *Intermediate Data Structure*. This is an abstract representation of the problem concepts and rules. This structure decouples the CNL from the target formalism, making it easy to extend the CNL itself and to support multiple target languages. Finally, the *ASP Rewriter* processes the *Intermediate Data Structure* and returns the corresponding ASP encoding.

To demonstrate the effectiveness of the specification language of CNL2ASP, we specified well-known combinatorial problems and performed an empirical analysis comparing the performance of the encodings automatically generated by CNL2ASP and the encoding written by human experts. While for simplicity we do not report the CNL specification of the problems, which is available in the tool repository[1], here, we report the results obtained by two well-known scheduling problems: the Nurse Scheduling Problem (NSP) [42] and the Chemotherapy Treatment Scheduling (CTS) Problem [43]. The NSP is the problem of assigning nurses to shifts (morning, afternoon, night, or rest) in a given period of time such that the assignment satisfies a set of requirements. The CTS problem, instead, consists of assigning a starting time to the chemotherapy treatments of the patients and to the phases necessary before each treatment.

Overall, as expected, the original human-written encodings perform in general better than those automatically generated by CNL2ASP as shown in Figure 1; concerning the NSP, it is approximately between 1.5 to 2 times slower than the original one, while the CTS on average requires 32 seconds to compute a solution, with a peak of 2 minutes on the hardest instance against an average of 8 seconds for the human-written encodings. While this overhead might be acceptable in some cases, it is not comparable to the optimized encodings that experts could manually produce. However, optimization is

[1]https://github.com/dodaro/cnl2asp/tree/main/examples

	CTS	GC	MAO	NSP	All
CNLWizard	266	130	287	415	514
CNL2ASP	2620	2162	2422	2566	2936

Table 1
Comparison of the lines of code needed to define CNLs using CNLWizard and CNL2ASP.

not the current focus of our tool. That said, existing tools such as NGO [2], which aim to automatically optimize ASP encodings, could improve the performance of the encodings automatically generated by CNL2ASP.

Subsequently, we extended CNL2ASP to support temporal operators, resulting in CNL2TEL. Thus, the specification language of CNL2ASP and the tool were extended to support temporal specifications and convert such language into the TELINGO input language, respectively. Then, a similar to CNL2ASP experimental analysis was conducted for CNL2TEL, demonstrating that the automatically generated temporal encodings do not introduce significant performance overhead. CNL2ASP and its extension with temporal operators are open source and publicly available [3].

CNLWizard, instead, was developed to facilitate the development of such CNLs and their conversion into a target logic formalism. This framework enables the user to specify a grammar for multiple target languages abstractly, automatically suggests a possible default implementation for some elements involved in the pipeline, and provides auxiliary data structures that make the construction of the CNL language more flexible and guided. CNLWizard is a two-phase system, in which in the first phase processes an input described in a simple YAML-based language and automatically generates the grammar for the CNL, along with a set of pre-implemented imperative functions that minimize boilerplate code. This allows developers to focus primarily on writing the specific code required to convert sentences into the target formalism, streamlining the process and making custom CNL development more accessible and efficient. Then, in the second phase, it expects as input the generated grammar and the implemented imperative functions, together with the CNL input text, and produces the corresponding KR program. As for CNL2ASP and CNL2TEL, CNLWizard is open source and publicly available [4].

For the evaluation of the performance, as CNLWizard is a tool to define novel CNLs, we evaluated how it can help developers to reduce their development time. This is often measured in Software Engineering as the number of lines of code needed to solve a problem (see, e.g. [44] for a discussion about pros and cons). In our case, the problem is the generation of a grammar for a CNL and its translation to a KR formalism. Specifically, we compared the lines of code (both grammar and Python code) required to implement the translations from a CNL into ASP using CNLWizard with the ones required by CNL2ASP. Concerning the problem specifications, we used the domains we previously defined for CNL2ASP, namely Chemotherapy Treatment Scheduling (CTS), Graph Coloring (GC), Manipulation of Articulated Objects (MAO), and Nurse Scheduling (NSP). However, being CNL2ASP a versatile tool with many constructs and functions, e.g., it supports temporal operators, and with a total of about 6 thousand lines of code, for a fair evaluation of the lines of code, for CNL2ASP, we only considered the lines needed for parsing and translating the specific problem. The results are presented in Table 1, where the column labeled “All" indicates the total lines of code required to support the CNL sentences across all the considered domains. The main advantages of CNLWizard are the compact way of describing the grammar of the CNLs, a quick and easy way to import parts of the grammar, and import/generate Python functions, and internal management of features such as the concatenation of grammar rules. As a result, CNLWizard consistently generated a corresponding CNL with significantly fewer lines of code, up to 10 times fewer than CNL2ASP.

[2] https://github.com/potassco/ngo
[3] https://github.com/dodaro/cnl2asp
[4] https://github.com/dodaro/CNLWizard

5. Preliminary Results and Open Issues

CNL2ASP has been published in the *Theory and Practice of Logic Programming* journal, while CNL2TEL has been presented in the *5th International Workshop on the Resurgence of Datalog in Academia and Industry (Datalog 2.0)* and its extended version is currently submitted to *Theory and Practice of Logic Programming* journal. In addition, CNL2ASP is currently being used as a middle layer between LLMs and ASP in the work of Borroto et al. [18]. They introduced a tool, NL2ASP, for generating ASP programs from natural language specifications. The tool accepts in input specifications written in natural language and translates them into CNL statements recognizable by CNL2ASP, which subsequently converts those into ASP code. Their experimental analysis showed that this is an effective solution and most often provides correct ASP code.

CNLWizard, instead, has been accepted to the *34th International Joint Conference on Artificial Intelligence*. For future work, we plan to support a conversion from ASP into CNL to enhance the readability of models already encoded in ASP. Another planned direction is the support for additional languages, both on the CNL and the target formalism sides. For instance, the Semantics of Business Vocabulary and Business Rules (SBVR) can be a possible CNL target. SBVR, standardized by the Object Management Group, enables the structured and human-readable specification of business rules. However, due to the inherent complexity of business rule formalization, often involving intricate constraints and policies, rule conflicts (i.e., logical inconsistencies within a rule set) frequently arise. These conflicts can undermine system reliability, introduce ambiguity, and hinder automation. Translating SBVR into a KR language could enable automated conflict detection, a key step in the validation and verification of business rules.

We are also exploring support for translation into CLINGO[DL], which extends CLINGO with difference and linear constraints. Concerning CNLWizard, it could be extended to naively support additional languages, currently limited to ASP, CP, and SMT, and enable users to share their custom CNLs with the broader community. Moreover, it would be interesting to extend the implementation of CNLWizard to avoid misspelled CNL text, e.g., by adding an optional syntax check that users can enable.

Finally, a valuable application could be providing high-level explainability. This capability is crucial for the success of any system that needs to justify its decisions. Being able to explain why one solution is preferred over another, or why a certain element is included or excluded, is essential. By taking advantage of the readability offered by a CNL, we can present these explanations in a way that is easier to understand, while, currently, solver outputs are often difficult to interpret, especially for users who lack experience with AI and formal logic. Improving this aspect could significantly enhance the usability and accessibility of logic programs.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT in order to: Grammar and spelling check. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] V. Lifschitz, Answer Set Programming, Springer, 2019. URL: https://doi.org/10.1007/978-3-030-24658-7. doi:10.1007/978-3-030-24658-7.

[2] F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, volume 2 of *Foundations of Artificial Intelligence*, Elsevier, 2006. URL: https://www.sciencedirect.com/science/bookseries/15746526/2.

[3] C. W. Barrett, R. Sebastiani, S. A. Seshia, C. Tinelli, Satisfiability modulo theories, in: Handbook of Satisfiability - Second Edition, volume 336 of *FAIA*, IOS Press, 2021, pp. 1267–1329. URL: https://doi.org/10.3233/FAIA201017. doi:10.3233/FAIA201017.

[4] M. Gebser, B. Kaufmann, T. Schaub, Conflict-driven answer set solving: From theory to practice, Artificial Intelligence 187 (2012) 52–89.
[5] M. Alviano, G. Amendola, C. Dodaro, N. Leone, M. Maratea, F. Ricca, Evaluation of disjunctive programs in WASP, in: M. Balduccini, Y. Lierler, S. Woltran (Eds.), LPNMR, volume 11481 of *LNCS*, Springer, 2019, pp. 241–255.
[6] C. Dodaro, G. Mazzotta, F. Ricca, Blending grounding and compilation for efficient ASP solving, in: KR, 2024.
[7] E. Erdem, M. Gelfond, N. Leone, Applications of answer set programming, AI Mag. 37 (2016) 53–68. URL: https://doi.org/10.1609/aimag.v37i3.2678. doi:10.1609/AIMAG.V37I3.2678.
[8] C. Dodaro, G. Galatà, M. Maratea, I. Porro, Operating room scheduling via answer set programming, in: AI*IA, volume 11298 of *LNCS*, Springer, 2018, pp. 445–459.
[9] C. Dodaro, G. Galatà, M. K. Khan, M. Maratea, I. Porro, An ASP-based solution for operating room scheduling with beds management, in: P. Fodor, M. Montali, D. Calvanese, D. Roman (Eds.), Proceedings of the Third International Joint Conference on Rules and Reasoning (RuleML+RR 2019), volume 11784 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 67–81.
[10] C. Dodaro, G. Galatà, M. Maratea, I. Porro, An ASP-based framework for operating room scheduling, Intelligenza Artificiale 13 (2019) 63–77.
[11] M. Alviano, R. Bertolucci, M. Cardellini, C. Dodaro, G. Galatà, M. K. Khan, M. Maratea, M. Mochi, V. Morozan, I. Porro, M. Schouten, Answer set programming in healthcare: Extended overview, in: IPS and RCRA 2020, volume 2745 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020.
[12] M. Cardellini, P. D. Nardi, C. Dodaro, G. Galatà, A. Giardini, M. Maratea, I. Porro, A two-phase ASP encoding for solving rehabilitation scheduling, in: S. Moschoyiannis, R. Peñaloza, J. Vanthienen, A. Soylu, D. Roman (Eds.), Proceedings of the 5th International Joint Conference on Rules and Reasoning (RuleML+RR 2021), volume 12851 of *LNCS*, Springer, 2021, pp. 111–125.
[13] P. Cappanera, M. Gavanelli, M. Nonato, M. Roma, Logic-based Benders decomposition in answer set programming for chronic outpatients scheduling, Theory and Practice of Logic Programming 23 (2023) 848–864.
[14] P. Cappanera, M. Gavanelli, M. Nonato, M. Roma, Decomposition approaches for scheduling chronic outpatients' clinical pathways in answer set programming, J. Log. Comput. 33 (2023) 1851–1871.
[15] S. Caruso, G. Galatà, M. Maratea, M. Mochi, I. Porro, Scheduling pre-operative assessment clinic with answer set programming, Journal of Logic and Computation 34 (2023) 465–493.
[16] P. Bruno, F. Calimeri, C. Marte, M. Manna, Combining deep learning and ASP-based models for the semantic segmentation of medical images, in: S. Moschoyiannis, R. Peñaloza, J. Vanthienen, A. Soylu, D. Roman (Eds.), Proceedings of the 5th International Joint Conference on Rules and Reasoning (RuleML+RR 2021), volume 12851 of *LNCS*, Springer, 2021, pp. 95–110.
[17] P. Bruno, F. Calimeri, C. Marte, Dedudeep: An extensible framework for combining deep learning and asp-based models, in: G. Gottlob, D. Inclezan, M. Maratea (Eds.), Proceedings of the 16th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2022), volume 13416 of *LNCS*, Springer, 2022, pp. 505–510.
[18] M. A. Borroto Santana, I. Kareem, F. Ricca, Towards automatic composition of ASP programs from natural language specifications, in: Proc. of IJCAI, ijcai.org, 2024, pp. 6198–6206. URL: https://www.ijcai.org/proceedings/2024/685.
[19] M. Wallace, Practical applications of constraint programming, Constraints An Int. J. 1 (1996) 139–168. URL: https://doi.org/10.1007/BF00143881. doi:10.1007/BF00143881.
[20] J. N. Hooker, W. J. van Hoeve, Constraint programming and operations research, Constraints An Int. J. 23 (2018) 172–195. URL: https://doi.org/10.1007/s10601-017-9280-3. doi:10.1007/S10601-017-9280-3.
[21] L. M. de Moura, N. S. Bjørner, Satisfiability modulo theories: introduction and applications, Commun. ACM 54 (2011) 69–77.
[22] A. Armando, C. Castellini, E. Giunchiglia, M. Idini, M. Maratea, TSAT++: an open platform for satisfiability modulo theories, Electronic Notes in Theoretical Computer Science 125 (2005) 25–36.

[23] S. Konrad, B. H. C. Cheng, Real-time specification patterns, in: Proc. of (ICSE), ACM, 2005, pp. 372–381. URL: https://doi.org/10.1145/1062455.1062526. doi:10.1145/1062455.1062526.
[24] A. Post, I. Menzel, A. Podelski, Applying restricted english grammar on automotive requirements - does it work? A case study, in: Proc. of REFSQ, volume 6606 of *LNCS*, Springer, 2011, pp. 166–180. URL: https://doi.org/10.1007/978-3-642-19858-8_17. doi:10.1007/978-3-642-19858-8_17.
[25] P. Filipovikj, G. Rodríguez-Navas, M. Nyberg, C. Seceleanu, SMT-based consistency analysis of industrial systems requirements, in: Proc. of SAC, ACM, 2017, pp. 1272–1279.
[26] S. Vuotto, M. Narizzano, L. Pulina, A. Tacchella, Poster: Automatic consistency checking of requirements with reqv, in: Proc. of IEEE ICST, IEEE, 2019, pp. 363–366.
[27] M. Narizzano, L. Pulina, A. Tacchella, S. Vuotto, Consistency of property specification patterns with boolean and constrained numerical signals, in: Proc. of NFM, volume 10811 of *LNCS*, Springer, 2018, pp. 383–398. URL: https://doi.org/10.1007/978-3-319-77935-5_26. doi:10.1007/978-3-319-77935-5_26.
[28] N. Mahmud, C. Seceleanu, O. Ljungkrantz, Resa tool: Structured requirements specification and sat-based consistency-checking, in: Proc. of FedCSIS, volume 8 of *Annals of Computer Science and Information Systems*, IEEE, 2016, pp. 1737–1746. URL: https://doi.org/10.15439/2016F404. doi:10.15439/2016F404.
[29] N. E. Fuchs, R. Schwitter, Specifying logic programs in controlled natural language, CoRR abs/cmp-lg/9507009 (1995). URL: http://arxiv.org/abs/cmp-lg/9507009. arXiv:cmp-lg/9507009.
[30] R. Schwitter, B. Hamburger, N. E. Fuchs, Attempto: Specifications in controlled natural language, in: Proc. of the Workshop on Logische Programmierung, 1995, pp. 151–160.
[31] N. E. Fuchs, Knowledge representation and reasoning in (controlled) natural language, in: Proc. of ICCS, volume 3596 of *LNCS*, Springer, 2005, pp. 51–51. URL: https://doi.org/10.1007/11524564_3. doi:10.1007/11524564_3.
[32] P. Clark, P. Harrison, T. Jenkins, J. A. Thompson, R. H. Wojcik, Acquiring and using world knowledge using a restricted subset of english, in: Proc. of FLAIRS, AAAI Press, 2005, pp. 506–511. URL: http://www.aaai.org/Library/FLAIRS/2005/flairs05-083.php.
[33] E. Erdem, R. Yeniterzi, Transforming controlled natural language biomedical queries into answer set programs, in: Proc. of the BioNLP Workshop, Association for Computational Linguistics, 2009, pp. 117–124.
[34] U. Öztok, E. Erdem, Generating explanations for complex biomedical queries, in: Proc. of AAAI, AAAI Press, 2011. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3519.
[35] C. Baral, J. Dzifcak, Solving puzzles described in english by automated translation to answer set programming and learning how to do that translation, in: Proc. of KR, AAAI Press, 2012.
[36] R. Schwitter, Specifying and verbalising answer set programs in controlled natural language, Theory Pract. Log. Program. 18 (2018) 691–705. doi:10.1017/S1471068418000327.
[37] T. Teitelbaum, T. Reps, The cornell program synthesizer: a syntax-directed programming environment, Commun. ACM 24 (1981) 563–573. URL: https://doi.org/10.1145/358746.358755. doi:10.1145/358746.358755.
[38] Z. Yang, A. Ishay, J. Lee, Coupling large language models with logic programming for robust and general reasoning from text, in: ACL, Association for Computational Linguistics, 2023, pp. 5186–5219.
[39] M. Alviano, L. Grillo, Answer set programming and large language models interaction with YAML: preliminary report, in: Proc. of CILC, volume 3733 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3733/short2.pdf.
[40] E. Coppolillo, F. Calimeri, G. Manco, S. Perri, F. Ricca, LLASP: fine-tuning large language models for answer set programming, in: Proc. of KR, 2024. URL: https://doi.org/10.24963/kr.2024/78. doi:10.24963/KR.2024/78.
[41] S. Caruso, C. Dodaro, M. Maratea, M. Mochi, F. Riccio, CNL2ASP: converting controlled natural language sentences into ASP, Theory Pract. Log. Program. 24 (2024) 196–226. URL: https://doi.org/10.1017/s1471068423000388. doi:10.1017/S1471068423000388.
[42] C. Dodaro, M. Maratea, Nurse scheduling via answer set programming, in: Proceedings of LPNMR,

volume 10377 of *LNCS*, Springer, 2017, pp. 301–307.
[43] C. Dodaro, G. Galatà, A. Grioni, M. Maratea, M. Mochi, I. Porro, An ASP-based solution to the chemotherapy treatment scheduling problem, Theory Pract. Log. Program. 21 (2021) 835–851.
[44] V. Nguyen, S. Deeds-Rubin, T. Tan, B. Boehm, A sloc counting standard, in: Cocomo ii forum, volume 2007, Citeseer, 2007, pp. 1–16.