CNL2ASP: Converting Controlled Natural Language Sentences into ASP

SIMONE CARUSO

DIBRIS, University of Genova, Genova, Italy
(*e-mail:* simone.caruso@edu.unige.it)

CARMINE DODARO

DeMaCS, University of Calabria, Rende, Italy
(*e-mail:* carmine.dodaro@unical.it)

MARCO MARATEA

DIBRIS, University of Genova, Genova, Italy
DeMaCS, University of Calabria, Rende, Italy
(*e-mail:* marco.maratea@unical.it)

MARCO MOCHI

DIBRIS, University of Genova, Genova, Italy
(*e-mail:* marco.mochi@edu.unige.it)

FRANCESCO RICCIO

Engineering Division, ALTEN Italia, Torino, Italy
(*e-mail:* riccio.francesco@outlook.it)

Abstract

Answer set programming (ASP) is a popular declarative programming language for solving hard combinatorial problems. Although ASP has gained widespread acceptance in academic and industrial contexts, there are certain user groups who may find it more advantageous to employ a higher-level language that closely resembles natural language when specifying ASP programs. In this paper, we propose a novel tool, called CNL2ASP, for translating English sentences expressed in a controlled natural language (CNL) form into ASP. In particular, we first provide a definition of the type of sentences allowed by our CNL and their translation as ASP rules and then exemplify the usage of the CNL for the specification of both synthetic and real-world combinatorial problems. Finally, we report the results of an experimental analysis conducted on the real-world problems to compare the performance of automatically generated encodings with the ones written by ASP practitioners, showing that our tool can obtain satisfactory performance on these benchmarks.

KEYWORDS: answer set programming, logic programming, controlled natural language

1 Introduction

Answer set programming (ASP) (Lifschitz 2019; Brewka *et al.* 2011; Gelfond and Lifschitz, 1988) is a well-known declarative programming paradigm proposed in the area of knowledge representation and reasoning (KRR) and geared toward solving hard

combinatorial problems. As a matter of fact, ASP has been widely used for solving problems in both academic and industrial contexts (see Erdem *et al.* 2016 for a complete survey on ASP applications). The success of ASP is mainly due to its simple syntax, its intuitive semantics, and the availability of efficient systems, like CLINGO (Gebser *et al.* 2016) or DLV (Alviano *et al.* 2017).

Nevertheless, despite the success of ASP, and in general of KR formalisms, it may be preferable for certain types of users to use a higher-level language that is closer to natural language for specifying ASP programs. For this reason, in the last decades, a number of attempts to convert English sentences expressed in a controlled natural language (CNL) into a KR formalism emerged (Fuchs 2005; Clark *et al.* 2005). In the context of ASP, a CNL has been used for solving logic puzzles (Baral and Dzifcak 2012) and for answering biomedical queries (Erdem and Yeniterzi 2009).

Arguably, using a CNL may offer several practical advantages:

1. CNL specifications are usually more readable.
2. Writing CNL specifications is expected to be easier and faster than encoding knowledge in a formal KR language, like ASP. The generated ASP encodings can be used as a starting point for further optimization made by ASP experts.
3. CNL specifications tend to be more adaptable to changes compared to ASP encodings, for example, adding a term in an ASP atom requires the substitution of all occurrences of the atoms, whereas in a CNL this should have almost no impact.
4. CNL specifications can be used as a basis for deploying richer language processing.

The contribution of this paper is in the aforementioned context. In fact, we propose a tool called CNL2ASP that automatically translates sentences expressed in a CNL language into ASP rules. The CNL supported by CNL2ASP is inspired by the Semantics of Business Vocabulary and Business Rules (SBVR) (Bajwa *et al.* 2011; The Business Rules Group 2000), which is a standard proposed by the Object Management Group to formally describe complex entities, for example, the ones related to a business, using natural language, and by PENG$^{\text{ASP}}$, a CNL defined by Schwitter (2018).

The development of the tool has been oriented toward four different types of use cases, that is, *(i)* enabling the possibility of specifying ASP programs also to users that have a limited experience on ASP; *(ii)* to help ASP experts to create a fast prototype of intuitive encodings which are subsequently subject to optimization; *(iii)* improving the readability of ASP programs since there is a one-to-one mapping between ASP rules and English specifications; and *(iv)* offering a modern tool that can be used as a basis for writing specifications in a natural language. In particular, to show the capabilities of our CNL, we reported several synthetic and real-world use cases showing how the CNL can be indeed used for solving (complex) combinatorial problems. Moreover, we performed an experimental analysis on the real-world use cases comparing the performance of the ASP encoding generated by CNL2ASP with the one created by ASP practitioners, showing that our tool can, in general, obtain good performances. Subsequently, we conducted a preliminary analysis to assess the usability and readability of the proposed CNL. Finally, we mention that the implementation of the tool presented in this paper is open source and publicly available at https://github.com/dodaro/cnl2asp.

Contributions. To summarize, the main contributions of this paper are the following:

1. We defined a CNL designed for solving complex combinatorial problems.

2. We implemented CNL2ASP, a tool that automatically translates English sentences expressed in such a CNL into a corresponding ASP encoding. CNL2ASP supports the main features of the ASP language, including disjunctive and choice rules, aggregates, and weak constraints (Section 3).
3. We provided several use cases on well-known, synthetic domains (Section 4), as well as on real-world problems described in the literature (Section 5). Concerning the latter, we also provided the results of an experimental analysis comparing the performance of the generated encodings with the ones written by human experts.
4. We performed a preliminary user validation to evaluate the usability and the readability of the CNL (Section 6).

2 Preliminaries

ASP (Brewka *et al.* 2011) is a programming paradigm developed in the area of KRR and logic programming. In this section, we overview the language of ASP (Calimeri *et al.* 2020).

Syntax. The syntax of ASP is similar to the one of Prolog. Variables are strings starting with an uppercase letter, and constants are integers or strings starting with lowercase letters. A *term* is either a variable or a constant. A *standard atom* is an expression $p(t_1,\ldots,t_n)$, where p is a *predicate* of arity n and $t_1,\ldots,t_n$ are terms. An atom $p(t_1,\ldots,t_n)$ is ground if $t_1,\ldots,t_n$ are constants. A *ground set* is a set of pairs of the form $\langle consts : conj \rangle$, where *consts* is a list of constants and *conj* is a conjunction of ground standard atoms. A *symbolic set* is a set specified syntactically as `{Terms`$_1$` :` `Conj`$_1$`; ...,` `Terms`$_t$` : Conj`$_t$`}` where $t > 0$, and for all $i \in [1, t]$, each `Terms`$_i$ is a nonempty list of terms, and each `Conj`$_i$ is a nonempty conjunction of standard atoms. A *set term* is either a symbolic set or a ground set. Intuitively, a set term `{X:a(X,c),p(X); Y:b(Y,m)}` stands for the union of two sets: the first one contains the `X`-values making the conjunction `a(X,c), p(X)` true, and the second one contains the `Y`-values making the conjunction `b(Y,m)` true. An *aggregate function* is of the form $f(S)$, where S is a set term, and f is an *aggregate function symbol*. Basically, aggregate functions map multisets of constants to a constant. The most common functions implemented in ASP systems are `#count`, for counting number of terms; `#sum`, for computing sum of integers, `#min`, for computing the minimum integer in a set, and `#max`, for computing the maximum integer in a set (Faber *et al.* 2011). An *aggregate atom* is of the form $f(S) \prec T$, where $f(S)$ is an aggregate function, $\prec \in \{<, <=, >, >=, !=, =\}$ is a comparison operator, and T is a term called guard. An aggregate atom $f(S) \prec T$ is ground if T is a constant and S is a ground set. An *atom* is either a standard atom or an aggregate atom. A *rule r* has the following form:

$$\texttt{a}_1 \texttt{ | } \ldots \texttt{ | } \texttt{a}_n \texttt{ :- } \texttt{b}_1, \texttt{ ..., } \texttt{b}_k, \texttt{ not } \texttt{b}_{k+1}, \texttt{ ..., } \texttt{ not } \texttt{b}_m.$$

where $\texttt{a}_1,\ldots,\texttt{a}_n$ are standard atoms (with $n \geq 0$), $\texttt{b}_1,\ldots,\texttt{b}_k$ are atoms, and $\texttt{b}_{k+1},\ldots,\texttt{b}_m$ are standard atoms (with $m \geq k \geq 0$). A literal is either a standard atom `a` or its negation `not a`. The disjunction $\texttt{a}_1 \texttt{ | } \ldots \texttt{ | } \texttt{a}_n$ is the *head* of r, while the conjunction $\texttt{b}_1, \ldots, \texttt{b}_k,$ `not` $\texttt{b}_{k+1}, \ldots,$ `not` $\texttt{b}_m$ is its *body*. Rules with empty body are called *facts*. Rules with empty head are called *constraints*. A variable that appears uniquely in set terms of a rule r is said to be *local* in r, otherwise it is a *global* variable of r. An ASP program is

a set of *safe* rules, where a rule r is *safe* if the following conditions hold: *(i)* for each global variable X of r there is a positive standard atom ℓ in the body of r such that X appears in ℓ and *(ii)* each local variable of r appearing in a symbolic set `{Terms : Conj}` also appears in a positive atom in `Conj`. A *weak constraint* ω (Buccafurri *et al.* 2000) is of the form:

$$:\sim \ \mathtt{b_1}, \ ..., \ \mathtt{b_k}, \ \mathtt{not} \ \mathtt{b_{k+1}}, \ ..., \ \mathtt{not} \ \mathtt{b_m}. \ [\mathtt{w@l}, \ \mathtt{t_1}, \ ..., \ \mathtt{t_z}]$$

where $\mathtt{t_1}, \ ..., \ \mathtt{t_z}$ are terms, $\mathtt{w}$ and $\mathtt{l}$ are the weight and level of ω, respectively. Intuitively, `[w@l]` is read "as weight $\mathtt{w}$ at level $\mathtt{l}$," where weight is the "cost" of violating the condition in the body of $\mathtt{w}$, whereas levels can be specified for defining a priority among preference criteria. An ASP program with weak constraints is $\Pi = \langle P, W \rangle$, where P is a program and W is a set of weak constraints. A standard atom, a literal, a rule, a program, or a weak constraint is *ground* if no variables appear in it.

Semantics. Let P be an ASP program. The *Herbrand universe* U_P and the *Herbrand base* B_P of P are defined as usual. The ground instantiation G_P of P is the set of all the ground instances of rules of P that can be obtained by substituting variables with constants from U_P. An *interpretation* I for P is a subset I of B_P. A ground standard atom $\mathtt{p}$ is true w.r.t. I if $\mathtt{p} \in I$, and false otherwise. A literal $\mathtt{not}$ $\mathtt{p}$ is true w.r.t. I if $\mathtt{p}$ is false w.r.t. I, and false otherwise. An aggregate atom is true w.r.t. I if the evaluation of its aggregate function (i.e., the result of the application of f on the multiset S) with respect to I satisfies the guard; otherwise, it is false. A ground rule r is *satisfied* by I if at least one atom in the head is true w.r.t. I whenever all conjuncts of the body of r are true w.r.t. I. A model is an interpretation that satisfies all rules of a program. Given a ground program G_P and an interpretation I, the *reduct* of G_P w.r.t. I is the subset G_P^I of G_P obtained by deleting from G_P the rules in which a body literal is false w.r.t. I (Faber *et al.* 2011). An interpretation I for P is an *answer set* (or stable model) for P if I is a minimal model (under subset inclusion) of G_P^I (i.e., I is a minimal model for G_P^I). Given a program with weak constraints $\Pi = \langle P, W \rangle$ and an interpretation I, the semantics of Π extends from the basic case defined above. Thus, let G_P and G_W be the instantiation of P and W, respectively. Then, let G_W^I be the set

$$\{(\mathtt{w@l,t_1,...,t_z}) \ | \ :\sim \ \mathtt{b_1},...,\mathtt{b_k},\mathtt{not} \ \mathtt{b_{k+1}},...,\mathtt{not} \ \mathtt{b_m}. \ [\mathtt{w@l,t_1,...,t_z}] \ \in G_W ,$$
$$\mathtt{b_1},...,\mathtt{b_k} \ \in I, \ and \ \mathtt{b_{k+1}},...,\mathtt{b_m} \ \notin I\}.$$

Moreover, for an integer l, $P_l^I = \sum_{(\mathtt{w@l,t_1,...,t_z}) \in G_W^I} w$ if there is at least one tuple in G_W^I whose level is equal to l, and 0 otherwise. Given a program with weak constraints $\Pi = \langle P, W \rangle$, an answer set M for P is said to be dominated by an answer set M' for P, if there exists an integer l such that $P_l^{M'} < P_l^M$ and $P_{l'}^{M'} = P_{l'}^M$ for all integers $l' > l$. An answer set M for P is said to be *optimal* or *optimum* for Π if there is no other answer set M' that dominates M (Calimeri *et al.* 2020).

Syntactic shortcuts. In the following, `p(1..n).` denotes the set of facts `p(1). ... p(n).`. Moreover, we use *choice rules* of the form `{X}`, where `X` is a set of atoms. Choice rules of this kind can be viewed as a syntactic shortcut for the rule `p | p'.` for each `p` $\in$ `X`, where `p'` is a fresh new atom not appearing elsewhere in the program, meaning that the atom p can be chosen as true. Choice rules can also have bounds, that is, `1 <= {X} <= 1`, and in this case can be seen as a shortcut for the choice rule `{X}` and the rule `:- #count{X} != 1`.
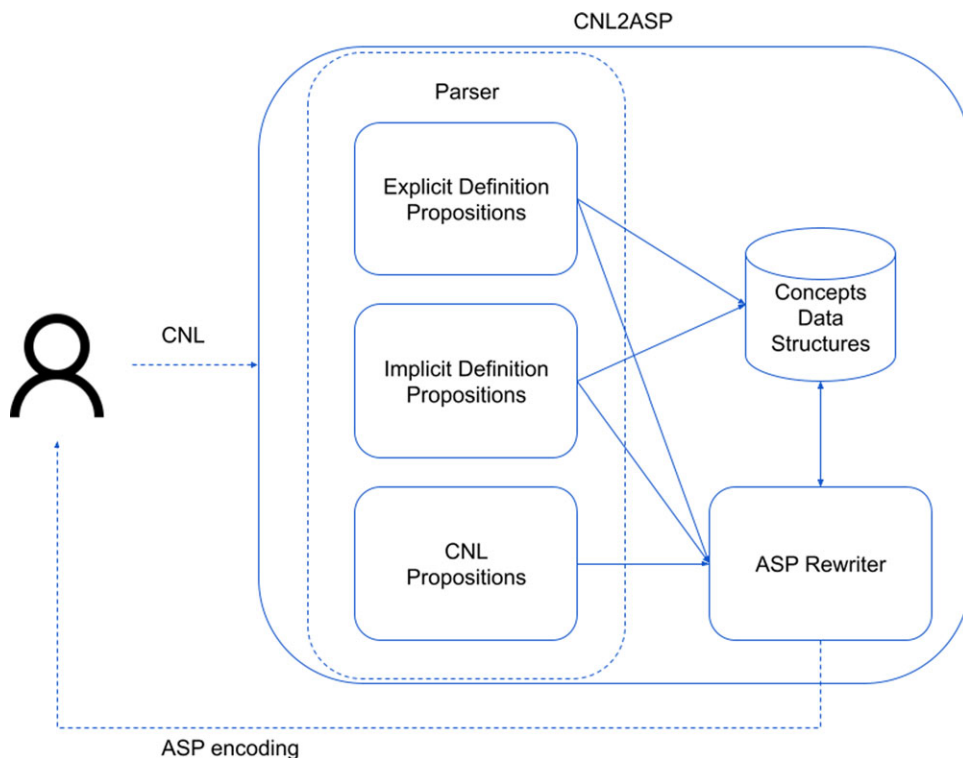
Fig. 1. Architecture of the tool CNL2ASP.

3 CNL2ASP

This section deals with the specification of CNL language and with the implementation of the tool CNL2ASP, whose architecture is depicted in Figure 1. The tool takes as input a file containing a list of statements written in a CNL and produces as output a file containing a set of ASP rules. 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. Concepts in a proposition define the application domain, that is, they describe entities that are used as subjects of other propositions. The combination of clauses that produces a proposition defines its type, that is used to understand what the proposition is supposed to mean and how that meaning can be translated into ASP rules and facts.

CNL2ASP is made of three main components, namely the *Parser*, the *Concepts Data Structures*, and the *ASP Rewriter*. In particular, each CNL proposition in the input file is processed by the Parser, whose role is *(i)* to create appropriate data structures for concepts to be stored in the concept data structures and *(ii)* to tokenize the CNL statements and send the result to the ASP Rewriter component. In more details, the parser interprets three subtypes of CNL propositions, namely explicit definition propositions, implicit definition propositions, and (standard) CNL propositions. In particular, the starting production rule is the following:

```
1  start ⟶ (explicit_definition_proposition | implicit_definition_proposition |
      standard_proposition)+
```

The first two types of propositions are used to define the concepts, where in our context a concept is a thing, a place, a person, or an object that is used to model entities of the application domain of the CNL. Standard CNL propositions are sentences describing the rules of the application domain. As an example, consider the application domain of describing the rights and the obligations of a customer of an online store. In this context, concepts can be the customer, the company, the product, and so on, whereas CNL propositions are sentences stating what actions customers and companies can/cannot do. It is important to highlight that, in our tool, concepts are exclusively defined by their names. Consequently, taking the earlier example into account, there exists only a single concept for customer, company, product, and so forth. After all the sentences have been processed by the Parser, they are sorted as follows: (explicit and implicit) definition propositions are processed before (standard) CNL propositions. Among the CNL propositions, the ASP Rewriter first processes the ones that are related to choice and disjunctive rules since they can also define new concepts in the data structures, and then processes strong and weak constraints. For each proposition, the ASP Rewriter first initializes the ASP atoms, then creates aggregates, arithmetic operations and comparisons, and further it merges all of them to create the head and the body of the ASP rules. Finally, after all ASP rules are created, they are stored in an output file that is returned to the user. In the following sections, we first describe the different propositions accepted by the Parser along with their grammar[1] and their translation as ASP rules (Sections 3.1, 3.2, and 3.3), and then we report a brief description of the usage of the tool (Section 3.4).

3.1 Explicit definition propositions

Explicit definition propositions are used to define the concepts occurring in the domain application, and they are used to create data structures which are later on used by the ASP Rewriter to produce ASP rules. In more details, the production rule of explicit definition propositions is the following:

```
1 explicit_definition_proposition ⟶ (domain_definition |
      temporal_concept_definition)CNL_END_OF_LINE
```

where each proposition is terminated by `CNL_END_OF_LINE` (in our case, a dot). In particular, an explicit definition proposition can be either a `domain_definition`, used to define all the entities of the problem and their structure; or a `temporal_concept_definition`, used to define only temporal elements, as days or timeslots.

3.1.1 Domain definition

Domain definitions start with a subject optionally followed by the sentence `"is identified by"` and the definition of the keys, that is, the parameters that uniquely represent the entity and then, also optionally, a sentence used to express the other parameters. The production rule is the following:

```
1 domain_definition: ("A " | "An ")? subject_name ("is identified by" atom_key)? ",
      and"? ("has" parameter ((","|", and") parameter)*)?
```

---

[1] For the sake of readability, we only provide basic elements of the grammar, we refer the reader to Caruso *et al.* (2022) for the full grammar.

Domain definitions are not directly translated as ASP rules, instead they are used to add elements in the data structures. All properties can be later on used in propositions to refer specific properties of a concept. By default, if no property is referred to by a sentence, then the identifier is used.

The following sentences are examples of domain definitions:

```
1 A movie is identified by an id, and has a title, a director, and a year.
2 A director is identified by a name.
3 A topMovie is identified by an id.
4 A scoreAssignment is identified by a movie, and by a value.
```

Note that `scoreAssignment` is identified by a movie, which is a concept that is created by the user. This has an impact on its translation into ASP, as shown in Section 3.3.1.

3.1.2 Temporal concept definition

Temporal concept definitions start with a subject followed by the sentence `"is a temporal concept expressed in,"` then by the temporal type that can be minutes, days, or steps. The preposition continues with a sentence used to express the temporal range and, finally, it can be closed with a sentence used to specify the length of each temporal step. The production rule is the following:

```
1 temporal_concept_definition: ("A " | "An ") subject_name "is a temporal concept
    expressed in" CNL_TEMPORAL_TYPE "ranging from" temporal_range_start "to"
    temporal_range_end ("with a length of" CNL_NUMBER ("minutes" | "days"))?
```

Temporal concepts enable the possibility to refer to them using special words like after, before, and so on.

An example of a temporal definition is the following sentence:

```
1 A timeslot is a temporal concept expressed in minutes ranging from 07:00 AM to
    09:00 AM with a length of 30 minutes.
```

Such concepts are conveniently translated as ASP facts by the ASP Rewriter as follows:

```
1 timeslot(1,"07:00").
2 timeslot(2,"07:30").
3 timeslot(3,"08:00").
4 timeslot(4,"08:30").
```

and the association between the used number and the corresponding time slot is stored into a dedicated data structure, so that when a user refers to a particular time slot (e.g., 07:30 AM), it is automatically encoded as the corresponding ASP atom (e.g., `timeslot(2, "07:30")`). The second term is a string representing the time slot, which is never used in the generated ASP encoding, but that can be useful when provided as output to the user.

3.2 Implicit definition propositions

Implicit definition propositions are used to define concepts that can, then, be used by other propositions. These definitions express the *signature* of the concept indicated by the subject of the proposition, carrying information regarding the concept in the definition that our tool can use later on in the specification whenever the same concept is

used. Differently from explicit definition propositions, users do not have to specify the properties of the concepts because they are inferred from the sentence. In more details, the production rule of implicit definition propositions is the following:

```
1 implicit_definition_proposition ⟶ (constant_definition_clause |
      compounded_clause | enumerative_definition_clause)CNL_END_OF_LINE
```

In particular, an implicit definition proposition can be a `constant_definition_clause`, used to specify constants; or a `compounded_clause`, used to define elements using lists and ranges; or a `enumerative_definition_clause`, used to define elements one at a time, optionally closing the proposition with a `when` clause, defining a condition in which the element is defined (e.g., `X is true when Y is true`), and with a `where` clause.

3.2.1 Constant definitions

Constant definitions are used to introduce constants to be used later on in the specification.

The following sentences are examples of constant definitions:

```
1 minKelvinTemperature is a constant equal to 0.
2 acceptableTemperature is a constant.
```

As we can see from the proposition at line 1, the constant 0 is introduced with a `equal to` clause, and it is bound to the subject of the proposition. Instead, in the proposition at line 2, we are defining a constant without assigning it a value, which can be later on assigned by the user (e.g., the ASP system CLINGO (Gebser *et al.* 2016) supports the option `--const` to specify constants). In the case of constant definitions, there are no translations to ASP available, because they are instead stored in the data structures and substituted in the resulting program when the subject of the definition is used.

3.2.2 Compound definitions

Compound definitions are used to introduce a set of related concepts all at once, by making use of either ranges of numbers or lists. The following sentences are examples of compound definitions:

```
1 A ColdTemperature goes from minKelvinTemperature to acceptableTemperature.
2 A day goes from 1 to 365.
3 A drink is one of alcoholic, nonalcoholic and has color that is equal to
      respectively blue, yellow.
```

Propositions at lines 1 and 2 are examples of definitions using a range, identified by the construct `goes from/to`. In particular, proposition at line 1 uses the constants defined in Section 3.2.1.

Proposition at line 3 is an example of a definition of a drink using lists with a `one of` clause, where one can also specify additional attributes for each element of the list in a positional way using a `respectively` clause, and a list with the same number of elements of the list enumerating all the possible values that the subject of the proposition can have.

The corresponding ASP code, in this case, is quite straightforward and is depicted below:

```
1 coldtemperature(0..acceptableTemperature).
2 day(1..365).
3 drink(1, "alcoholic", "blue").
4 drink(2, "nonalcoholic", "yellow").
```

First of all, note that constant `minKelvinTemperature` is directly replaced by its value (i.e., 0), whereas constant `acceptableTemperature` is left as is. List elements defined in proposition at line 3 carry on their position number with them, which turns out to be handy as a basic way to encode precedence relationships when the subject is not a number.

3.2.3 Enumerative definitions

Enumerative definitions are used to introduce a property for a single concept or a relationship among a set of concepts. The peculiarity of this kind of propositions lies in the different translations into ASP as the clauses used within them change.

The following sentences are examples of enumerative definitions:

```
1 John is a waiter.
2 1 is a pub.
3 Alice is a patron.
4 Waiter John works in pub 1.
5 Waiter John serves a drink alcoholic.
6 Pub 1 is close to pub 2 and pub X, where X is one of 3,4.
7 Waiter W is working when waiter W serves a drink.
```

Such propositions show the construction to define relationships or properties related to a particular subject. In particular, propositions from line 1 to line 3 are used to define the concepts of waiter, pub, and patron, respectively, whereas propositions at lines 4 and 5 define concepts related to work in and to serve, respectively. Proposition at line 6 illustrates another feature of our CNL, that is, `where` clauses, that are used in the example to define the values that the variable `X` can take. Proposition at line 7 is a *conditional* definition, identified by a `when` clause.

The translations in ASP of these examples are the following:

```
1 waiter("john").
2 pub(1).
3 patron("alice").
4 work_in("john", 1).
5 serve("john", "alcoholic").
6 close_to(1, 2, 3). close_to(1, 2, 4).
7 working(W) :- serve(W,_).
```

Propositions from line 1 to line 6 always hold true, therefore they are used by the ASP Rewriter to produce the corresponding ASP facts. In particular, proposition at line 6 is translated in a similar manner to compound definitions with lists. Instead, proposition at line 7 holds true only if the statement introduced by the `when` clause is true, hence it is translated into an ASP rule, where the body of the rule is the element inside the `when` clause.

Note that, in these examples, `W` is considered as a variable, whereas `John` and `Alice` are treated as ASP strings. This is because CNL2ASP assumes that every object starting with an upper case letter and containing only upper case letters, numbers or symbols is

considered as a variable, while other objects are strings or numbers (e.g., `MY_VARIABLE` is considered as a variable, whereas `My_String` is considered as a string).

3.3 CNL propositions

Explicit and implicit definition propositions are used to define the concepts of the domain application, whose specifications are instead described by (standard) CNL propositions. The production rule of standard CNL propositions is the following:

```
1 standard_proposition ⟶ (
2        whenever_then_clause_proposition |
3        fact_proposition |
4        quantified_choice_proposition |
5        negative_strong_constraint_proposition |
6        positive_strong_constraint_proposition |
7        weak_constraint_proposition |
8        )CNL_END_OF_LINE
```

Therefore, the CNL considers several types of propositions, which are described in the following.

3.3.1 Whenever/then clauses

`Whenever/then` clauses are used to describe actions occurring when a condition is fulfilled. In more details, the production rule is the following:

```
1 whenever_then_clause ⟶ (whenever_clause ","?)+ then_clause
```

They start with `whenever` clauses, that is, sentences specifying conditions, followed by a `then` clause, that is a sentence used to express the actions that must or can hold when the `whenever` clauses are fulfilled.

The following sentences are examples of `whenever/then` clauses:

```
1 Whenever there is a movie with director equal to spielberg, with id X then we
     must have a topmovie with id X.
2 Whenever there is a director with name X different from spielberg then we can
     have at most 1 topmovie with id I such that there is a movie with director X,
     and with id I.
3 Whenever there is a movie with id I, with director equal to nolan then we can
     have a scoreAssignment with movie I, and with value equal to 3 or a
     scoreAssignment with movie I, and with value equal to 2.
```

Such propositions are encoded in ASP as follows:

```
1 topmovie(X) :- movie(X,_,"spielberg",_).
2 0 <= {topmovie(I):movie(I,_,X,_)} <= 1 :- director(X), X != "spielberg".
3 scoreassignment(movie(I),3) | scoreassignment(movie(I),2) :-
     movie(I,_,"nolan",_).
```

In particular, the form `whenever/then` followed by the word `must` is translated as a normal rule by the ASP Rewriter, whereas if it is followed by the word `can` then it can be translated as a choice rule or as a disjunctive rule depending on whether the CNL sentence contains the keyword `or`. Here, we want also to emphasize the fact that the first term of `scoreAssignment` is of the form `movie(I)` since it is defined to be of the type movie.

3.3.2 Fact proposition

Fact propositions are used to define the facts of the problem. Differently from implicit definition propositions, here no new concepts are introduced, meaning that all concepts used here must be explicitly defined. An example of a fact proposition is the following sentence:

```
1 There is a movie with id equal to 1, with director equal to spielberg, with title
      equal to jurassicPark, with year equal to 1993.
```

This sentence is translated as:

```
1 movie(1,"jurassicPark","spielberg",1993).
```

It is worth mentioning that the order of the elements listed in the sentence has no impact on its translation, since the properties of the concepts are explicitly defined. Therefore, the specifications listed below all produce the same ASP output.

```
1 There is a movie with id equal to 1, with director equal to spielberg, with year
      equal to 1993, with title equal to jurassicPark.
2 There is a movie with director equal to spielberg, with year equal to 1993, with
      id equal to 1, with title equal to jurassicPark.
```

3.3.3 Quantified choice propositions

Quantified choice propositions are used to define relationships or properties that *can* be true for a given set of selected concepts following a choice. Also these propositions define a *signature* for the concept upon which the choice has to be made. Quantified propositions are always introduced by the `every` quantifier and, since they express possibilities, always contain a `can` clause. In more details, the production rule is the following:

```
1 quantified_choice_proposition ⟶ quantifier subject_clause "can" CNL_COPULA? (
      verb_name | verb_name_with_preposition) (quantified_exact_quantity_clause |
      quantified_range_clause)? (quantified_object_clause | disjunctive_clause)?
      foreach_clause?
```

Thus, they start with a quantifier and are always followed by a subject and a verb, optionally connected by a `CNL_COPULA` (e.g., `is, is a, is an, ...`) and then, also optionally, either by a sentence of type `quantified_exact_quantity_clause`, used to express the quantity in exact terms (e.g., `exactly 1`); or by a sentence of type `quantified_range_clause`, used to express it using a range (e.g., `between 1 and 2`). The proposition can be closed either with an object clause, that is, a sentence used to express an object for the proposition, in a `subject verb object` fashion, or with a disjunctive clause; and, finally, a `foreach` clause, that is, a sentence used to express additional objects for which any possible value can be tried.

The following sentences are examples of quantified choice propositions:

```
1 Every patron can drink in exactly 1 pub for each day.
2 Every waiter can serve a drink.
3 Every movie with id I can have a scoreAssignment with movie I, and with value
      equal to 1 or a scoreAssignment with movie I, and with value equal to 2, or a
      scoreAssignment with movie I, and with value equal to 3.
```

Proposition at line 1 shows how one can express an exact number of choices that can be made for the concept expressed by the subject, and also how other concepts can be used in tandem with the subject to create a sort of cartesian product of choices, using a `for each` clause. These last constructions are optional, as shown in proposition at line 2. Proposition at line 3, instead, shows an example of a disjunctive clause. Their full translations into ASP is shown below:

```
1 <= {drink_in(_X1,_X2,_X3):pub(_X3)} <= 1 :- patron(_X1), day(_X2).
{serve(_X1,_X2):drink(_,_,_X2,_)} :- waiter(_X1).
scoreassignment(movie(I),1) | scoreassignment(movie(I),2) |
    scoreassignment(movie(I),3) :- movie(I,_,_,_).
```

The first two translations use choice rules (possibly with bounds), that are the ASP constructs that make it possible to represent propositions of this type, whereas the third one uses a disjunctive rule. Note that the first two rules also employ generated variables (starting with symbol `_`) that are used wherever two atoms have to be bound and no variable to use has been found in the specification given in input. This feature enables the specification writer to avoid cluttering the document with unnecessary variables, as can be seen throughout the propositions, with the only limitation that anaphoras have to be expressed explicitly by providing the correct variable.

3.3.4 Negative and positive strong constraints

Negative and positive strong constraint propositions are used to define assertions that *must* be true for a given set of selected concepts. This kind of propositions does not introduce new *signatures* but, on the contrary, they consume other signatures that were previously defined, meaning that the concepts used inside such constraints have to be defined before they are used. A strong constraint can represent either a prohibition (sentences starting with `It is prohibited`) or a requirement (sentences starting with `It is required`). After specifying if the strong constraint is a prohibition or a requirement, then a user can add *simple* clauses, that are made of a subject, a verb, and related object clauses; *aggregate* clauses, either in active or passive form, that define an aggregation of the set of concepts that satisfy the statement in their body with the operator that was specified (`number`, `total`, `lowest`, `highest`); or other complex clauses as shown below.

In more details, the production rules of strong constraints are the following:

```
1 negative_strong_constraint_clause ⟶ "it is prohibited that" (simple_clause ("
    and also" simple_clause)* (where_clause)? ("," (whenever_clause ","?)+)? |
    aggregate_clause comparison_clause (where_clause)? ("," (whenever_clause ","
    ?)+)? | when_then_clause (where_clause)? | quantified_constraint (
    where_clause)? | condition_clause "," (whenever_clause ","?)+ |
    temporal_constraint "," (whenever_clause ","?)+)
2 positive_strong_constraint_proposition ⟶ "it is required that" (simple_clause "
    ," (whenever_clause ","?)+ | aggregate_clause comparison_clause (where_clause
    )? ("," (whenever_clause ","?)+)? | when_then_clause (where_clause)? |
    quantified_constraint (where_clause)? | condition_clause "," (whenever_clause
    ","?)+ | temporal_constraint "," (whenever_clause ","?)+)
```

It is possible to observe that they start with the sentence `it is prohibited that` or with the sentence `it is required that` and are always followed by a simple clause, that is, a sentence of the form `subject verb object`; by an aggregate clause, a sentence expressing

a form of aggregations (e.g., `the number of`); by a whenever clause, described in Section 3.3.1; by a quantified constraint, used to specify clauses with quantifiers as `every` or `any`; or by a temporal constraint, used to specify constraints on temporal concepts as `after 11:00 AM` or `before 11:00 AM`. After simple clauses, aggregate clauses, and quantified constraints, additional sentences can be added, which can be of the type `where_clause`, used to specify conditions; or of the type `comparison_clause`, used to specify comparison between elements (e.g., `X is equal to 1`).

The following sentences are examples of negative and positive strong constraints:

```
1 It is prohibited that waiter W1 work in pub P1 and also waiter W2 work in pub P1,
       where W1 is different from W2.
2 It is prohibited that X is equal to Y, whenever there is a movie with id X, and
       with year equal to 1964, whenever there is a topMovie with id Y.
3 It is prohibited that the lowest value of a scoreAssignment with movie X is equal
       to 1, whenever there is a topMovie with id X.
4 It is required that the total value of a scoreAssignment with movie X is equal to
       10, such that there is a topMovie with id X.
5 It is required that the number of pub where a waiter work in is less than 2.
6 It is required that when waiter X works in pub P1 then waiter X does not work in
       pub P2, where P1 is different from P2.
7 It is required that V is equal to 3, whenever there is a movie with id I, and
       with director equal to spielberg, whenever there is a scoreAssignment with
       movie I, and with value V.
8 It is required that every waiter is payed.
```

Proposition at line 1 shows a practical example of the combination of several simple clauses, and the feature enabled by `where` clauses, that makes it possible to express comparisons between variables. Proposition at line 2 shows an example of `whenever` clause. Propositions at line 3, at line 4, and at line 5 show examples of aggregation expressing conditions on the minimum value, on the sum of values, and on the number of occurrences, respectively. Proposition at line 6 shows a `when/then` clause. Proposition at line 7 shows an example of `whenever` clause in the context of positive strong constraints. Lastly, proposition at line 8 is an example of how to specify a requirement that must hold for all the elements present in a particular set of concepts. Such propositions are encoded as ASP rules as follows:

```
1 :- work_in(W1,P1), work_in(W2,P1), W1 != W2.
2 :- movie(X,_,_,1964), topmovie(Y), X = Y.
3 :- topmovie(X), #min{_X1: scoreassignment(movie(X),_X1)} = 1.
4 :- #sum{_X1: scoreassignment(movie(X),_X1), topmovie(X)} != 10.
5 :- waiter(_X1), #count{_X2: work_in(_X1,_X2)} >= 3.
6 :- work_in(X,P1), work_in(X,P2), P1 != P2.
7 :- movie(I,_,"spielberg",_), scoreassignment(movie(I),V), V != 3.
8 :- not payed(_X1), waiter(_X1).
```

Note that their translation is quite intuitive, and positive strong constraints are translated as ASP constraints by negating the condition expressed by the sentence.

3.3.5 Weak constraint propositions

Weak constraint propositions are used to define assertions that are *preferably* true for a given set of selected concepts. Also this type of propositions consumes *signatures* from

previously defined concepts. They are always introduced by `it is preferred` and need the specification of the optimization objective (either minimization or maximization), and the level of priority of the optimization (low, medium or high). The production rule is the following:

```
1 weak_constraint_proposition ⟶ "it is preferred that"
      CNL_WEAK_OPTIMIZATION_OPERATOR? ","? weak_priority_operator ","? "that" (
      condition_operation | aggregate_clause | subject_clause CNL_COPULA (verb_name
      | verb_name_with_preposition) object_clause "," whenever_clause)
      weak_optimization_operator? (where_clause)?
```

In particular, they start with the sentence `it is preferred ... that`, and can be followed by a sentence expressing the nature of the optimization (i.e., `as much as possible` or `as little as possible`), and are always followed by a priority operator, i.e., a sentence expressing the level of relevance of the constraint with respect to other weak constraints (e.g., `"with low priority"`) and either a clause followed by a whenever clause, an aggregate clause or a condition operation, that is, a sentence expressing operations between variables in the proposition (e.g., `the sum of X and Y`). The proposition is closed with an optimization operator, that is, a sentence expressing the nature of the optimization (i.e., `"is minimized"` or `"is maximized"`) and an optional where clause. Note that here we have two ways for expressing the object, either in the form of as much (little) as possible at the beginning of the sentence or using is maximized (minimized) at the end of the sentence. The two ways are equivalent, but we support both of them to make sentences more natural. Moreover, sentences containing both kind of specifications are well-formed, thus they are correctly parsed even if they are in contrast (e.g., a user can specify `as much as possible` and `"is minimized"` in the same sentence). However, CNL2ASP subsequently checks if this happens and, in case, it triggers an error so that only one of the form is used.

The following sentences are examples of weak constraint propositions:

```
1 It is preferred with low priority that the number of drinks that are serve is
      maximized.
2 It is preferred as little as possible, with high priority, that V is equal to 1,
      whenever there is a scoreAssignment with movie I, and with value V, whenever
      there is a topMovie with id I.
3 It is preferred, with medium priority, that whenever there is a topMovie with id
      I, whenever there is a scoreAssignment with movie I, and with value V, V is
      maximized.
4 It is preferred, with medium priority, that the total value of a scoreAssignment
      is maximized.
```

The sentence at line 1 shows an example of a maximization over the result of a `#count` aggregate. The sentence at line 2 instead is an example of minimization using the form `as little as possible`. Then, the sentence at line 3 shows a sentence where the subject of maximization is a variable defined in `scoreAssignment`. Finally, the sentence at line 4 is an example of a `#sum` aggregate, where the result of the aggregation is subject to maximization. Translation of the propositions above are shown below:

```
1 :∼ #count{_X1: serve(_,_X1)} = _X2. [-_X2@1]
2 :∼ scoreassignment(movie(I),V), topmovie(I), V = 1. [1@3, I,V]
3 :∼ topmovie(I), scoreassignment(movie(I),V). [-V@2, I]
4 :∼ #sum{_X1: scoreassignment(movie(_),_X1)} = _X2. [-_X2@2]
```

Also in this case the translation is quite intuitive; however, one should note that maximization constructs are translated using negative weights.

3.4 Usage

In this section, we provide a few technical details and report the usage of the tool. CNL2ASP has been implemented using the programming language Python, and the open-source library lark (https://github.com/lark-parser/lark) for creating the Parser, which is the only required dependence to run it. Moreover, the tool requires to use the version 3.10 (or higher) of Python. Concerning the distribution licence, CNL2ASP is released under the Apache 2.0 licence, a permissive open-source licence, which allows the user to use it also in industrial contexts. Its usage is quite intuitive since it can be used as a standalone tool by issuing the command

```
1   python3 src/main.py input_file [output_file]
```

or, as an alternative, it can be used as a library in other Python projects by simply importing it.

4 Synthetic use cases

In this section, we present some examples to demonstrate how the language can be used to define well-known combinatorial problems in a natural and easily understandable way. The corresponding translations into ASP are also provided.

4.1 Graph coloring

We begin by presenting an encoding of the graph coloring problem using our CNL. We recall that the graph coloring problem is the problem of finding an assignment of colors to nodes in a graph such that two adjacent nodes do not share the same color.

```
1 A node goes from 1 to 3.
2 A color is one of red, green, blue.
3 Node 1 is connected to node X, where X is one of 2, 3.
4 Node 2 is connected to node X, where X is one of 1, 3.
5 Node 3 is connected to node X, where X is one of 1, 2.
6 Every node can be assigned to exactly 1 color.
7 It is required that when node X is connected to node Y then node X is not
      assigned to color C and also node Y is not assigned to color C.
```

One can notice the presence of explicit definition propositions (lines 1–5), with a ranged definition proposition (line 1) and a list definition proposition (line 2), enumerative definition propositions with where clauses (lines 3–5), a quantified clause (line 6) and, lastly, a positive strong constraint (line 7).

The resulting ASP encoding is the following:

```
1 node(1..3).
2 color(1,"red"). color(2,"green"). color(3,"blue").
3 connected_to(1,2). connected_to(1,3).
4 connected_to(2,1). connected_to(2,3).
5 connected_to(3,1). connected_to(3,2).
```

```
6 {assigned_to(_X1,_X2): color(_,_X2)} = 1 :- node(_X1).
7 :- connected_to(X,Y), assigned_to(X,C), assigned_to(Y,C).
```

where each proposition at line i is translated as the rule(s) reported at line i (with $i = 1..7$).

4.2 Hamiltonian path

The second problem we consider here is the well-known Hamiltonian path problem, which is the problem of finding a path in a graph that visits each node exactly once, starting from a given node.

```
1  A node goes from 1 to 5.
2  Node 1 is connected to node X, where X is one of 2, 3.
3  Node 2 is connected to node X, where X is one of 1, 4.
4  Node 3 is connected to node X, where X is one of 1, 4.
5  Node 4 is connected to node X, where X is one of 3, 5.
6  Node 5 is connected to node X, where X is one of 3, 4.
7  Every node X can have a path to a node connected to node X.
8  It is required that the number of nodes where node X has a path to is equal to 1.
9  It is required that the number of nodes that have a path to node X is equal to 1.
10 Node Y is reachable when node X is reachable and also node X has a path to node Y
      .
11 It is required that every node is reachable.
12 start is a constant equal to 1.
13 Node start is reachable.
```

Line 1 defines the nodes and lines from 2–6 define the connections between nodes. Then, line 7 reports a quantified proposition with an object accompanied by a verb clause, lines 8 and 9 report strong constraint propositions with aggregates, line 10 reports a conditional definition clause, line 11 reports a constraint clause with the presence of a quantifier, and line 12 defines the constant `start`, which is subsequently used in line 13. The ASP encoding corresponding to the CNL statements is the following:

```
1  node(1..5).
2  connected_to(1,2). connected_to(1,3).
3  connected_to(2,1). connected_to(2,4).
4  connected_to(3,1). connected_to(3,4).
5  connected_to(4,3). connected_to(4,5).
6  connected_to(5,3). connected_to(5,4).
7  {path_to(X,_X1): connected_to(X,_X1)} :- node(X).
8  :- node(X), #count{_X2: path_to(X,_X2)} != 1.
9  :- node(X), #count{_X3: path_to(_X3,X)} != 1.
10 reachable(Y) :- reachable(X), path_to(X,Y).
11 :- not reachable(_X4), node(_X4).
12 reachable(1).
```

where a CNL statement at line i is represented by the rule(s) at line i with ($i = 1..11$), whereas CNL statements reported in lines 12 and 13 are encoded by the rule at line 12. As an alternative, one could also use the sentence `start is a constant`, and then use the solver options to change the starting node.

4.3 Maximal clique

The third problem is the maximal clique problem, which is the problem of finding a clique, that is, a subset of the nodes of a given graph where all nodes in the clique are adjacent to each other, and the cardinality of the clique is maximal.

```
1 A node goes from 1 to 5.
2 Node 1 is connected to node X, where X is one of 2, 3.
3 Node 2 is connected to node X, where X is one of 1, 3, 4, 5.
4 Node 3 is connected to node X, where X is one of 1, 2, 4, 5.
5 Node 4 is connected to node X, where X is one of 2, 3, 5.
6 Node 5 is connected to node X, where X is one of 2, 3, 4.
7 Every node can be chosen.
8 It is required that when node X is not connected to node Y then node X is not
      chosen and also node Y is not chosen, where X is different from Y.
9 It is preferred with high priority that the number of nodes that are chosen is
      maximized.
```

where statements from line 1 to line 6 define the input graph. Then, line 7 reports a quantified proposition with no object, line 8 contains a strong constraint proposition with a comparison on the variables used inside it, and line 9 reports a weak constraint expressing a maximization preference on the highest priority level. The resulting ASP encoding is reported in the following:

```
1 node(1..5).
2 connected_to(1,2). connected_to(1,3).
3 connected_to(2,1). connected_to(2,3). connected_to(2,4). connected_to(2,5).
4 connected_to(3,1). connected_to(3,2). connected_to(3,4). connected_to(3,5).
5 connected_to(4,2). connected_to(4,3). connected_to(4,5).
6 connected_to(5,2). connected_to(5,3). connected_to(5,4).
7 {chosen(_X1)} :- node(_X1).
8 :- not connected_to(X,Y), chosen(X), chosen(Y), X != Y.
9 :~ #count{_X1: chosen(_X1)} = _X2. [-_X2@1]
```

where each CNL proposition at line i is translated as the rule(s) reported at line i (with $i = 1..9$).

5 Real-world use cases

In this section, we show the usage of the CNL specifications to encode three real-world problems which we previously addressed using plain ASP encodings, namely the Nurse Scheduling Problem (NSP) (Section 5.1; Dodaro and Maratea 2017), the Manipulation of Articulated Objects Using Dual-Arm Robots (Section 5.2; Bertolucci *et al.* 2021), and the Chemotherapy Treatment Scheduling (CTS) Problem (Section 5.3; Dodaro *et al.* 2021). Moreover, for each of the reported problem, we also show an empirical analysis comparing the performance of the encoding generated in an automatic way by CNL2ASP and the encoding written by human experts. The encodings were compared using the same solver, that is, CLINGO version 5.6.1 configured with the same options used in the original papers where the problems were presented. The experiments were executed on a AMD Ryzen 5 2600 with 3.4 GHz, with time and memory limits set to 1200 s and 8 GB, respectively. For the sake of the readability, we do not report in this section the generated encodings, which are, however, available in Supplementary material.

5.1 Nurse scheduling problem (NSP)

The NSP is the problem of computing an assignment of nurses to shifts (morning, afternoon, night, or rest) in a given period of time such that the assignment satisfies a set of requirements. In particular, the NSP described in this section was originally defined by Dodaro and Maratea (2017), where authors classified the requirements as follows: *(i)* Hospital requirements, which impose the length of the shifts and that each shift is associated with a minimum and a maximum number of nurses that must be present in the hospital; *(ii)* nurses requirements, which impose that nurses have a limit on the minimum and maximum number of working hours during the considered period of time, and that each nurse has an adequate rest period; *(iii)* balance requirements, which impose that the number of times a nurse can be assigned to morning, afternoon and night shifts is fixed.

The first part of our CNL specifications concerns the definition of the domain and of the input facts of the NSP, and it is reported in the following:

```
1  numberOfNurses is a constant.
2  A nurse goes from 1 to numberOfNurses.
3  A day goes from 1 to 365 and is made of shifts that are made of hours.
4  A shift is one of morning, afternoon, night, specrest, rest, vacation and has
       hours that are equal to respectively 7, 7, 10, 0, 0, 0.
5  maxNurseMorning is a constant.
6  maxNurseAfternoon is a constant.
7  maxNurseNight is a constant.
8  minNurseMorning is a constant.
9  minNurseAfternoon is a constant.
10 minNurseNight is a constant.
11 maxHours is a constant equal to 1692.
12 minHours is a constant equal to 1687.
13 maxDay is a constant equal to 82.
14 maxNight is a constant equal to 61.
15 minDay is a constant equal to 74.
16 minNight is a constant equal to 58.
17 balanceNurseDay is a constant equal to 78.
18 balanceNurseAfternoon is a constant equal to 78.
19 balanceNurseNight is a constant equal to 60.
```

In the definition above, we used implicit definition propositions that therefore also create the input facts of the problem. Note that the number of nurses is a constant that is specified by the user, some constants like `maxNurseMorning`, `maxNurseAfternoon`, etc. depend on the number of nurses, therefore they are also left to the user, whereas all other constants are specific to the NSP considered, therefore they are stated.

Then, the second part of our CNL specifications is used for solving the problem:

```
1  Every nurse can work in exactly 1 shift for each day.
2  It is required that the number of nurses that work in shift S for each day is at
       most M, where S is one of morning, afternoon, night and M is one of
       respectively maxNurseMorning, maxNurseAfternoon, maxNurseNight.
3  It is prohibited that the number of nurses that work in shift S for each day is
       less than M, where S is one of morning, afternoon, night and M is one of
       respectively minNurseMorning, minNurseAfternoon, minNurseNight.
4  It is prohibited that the total of hours in a day where a nurse works in is more
       than maxHours.
```

5 It is prohibited that the total of hours in a day where a nurse works in is less than minHours.

6 It is prohibited that the number of days with shift vacation where a nurse works in is different from 30.

7 It is prohibited that a nurse works in shift S in a day and also the next day works in a shift before S, where S is between morning and night.

8 It is required that the number of occurrences between each 14 days with shift rest where a nurse works in is at least 2.

9 It is required that when a nurse works in shift night for 2 consecutive days then the next day works in shift specrest.

10 It is prohibited that a nurse works in a day in shift specrest and also the previous 2 consecutive days does not work in shift night.

11 It is prohibited that the number of days with shift S where a nurse works in is more than M, where S is one of morning, afternoon, night and M is one of respectively maxDay, maxDay, maxNight.

12 It is prohibited that the number of days with shift S where a nurse works in is less than M, where S is one of morning, afternoon, night and M is one of respectively minDay, minDay, minNight.

13 It is preferred, with high priority, that the difference in absolute value between B, and the number of days with shift S where a nurse works in ranging between minDay and maxDay is minimized, where B is one of balanceNurseDay, balanceNurseAfternoon and S is one of morning, afternoon.

14 It is preferred, with high priority, that the difference in absolute value between balanceNurseNight, and the number of days with shift night where a nurse works in ranging between minNight and maxNight is minimized.

Here, it is interesting to observe that the specifications first define that a nurse can work in exactly one shift for each day leaving a free choice about the shift to assign to each nurse, and then they impose some requirements on the assigned shift. Moreover, note that in general we used negative constraints (i.e., sentences starting with It is prohibited), with the exception of the ones at lines 8 and 9 since we found that this formulation is more natural.

Comparison of the performances. The encoding generated by the CNL specifications described before has been compared to the original one proposed by Dodaro and Maratea (2017) (referred to as *Original*) and with an optimized version proposed by Alviano *et al.* (2018) (referred to as *Optimized*). The experiment consists of five instances of the NSP with increasing number of nurses. Results are shown in Figure 2, where it is possible to observe that the optimized encoding outperforms both the original and the CNL one. This result is not surprising since the optimized encoding takes advantage of specific properties of the NSP to reduce the search space for the solver. Concerning the performance of the CNL encoding, it is possible to observe that it is approximately between 1.5 and 2 times slower than the original one. The main difference in terms of performance is due to the fact that CNL encoding generates aggregates for constraints at lines 9 and 10, which are less efficient in this context than the normal rules used in the original encoding. In this respect, tools for the automatic rewriting of aggregates, such as the one proposed by Dingess and Truszczynski (2020), can be helpful also in our context. However, it is worth mentioning that, even on the hardest instance, the generated encoding is able to terminate in approximately ten minutes.
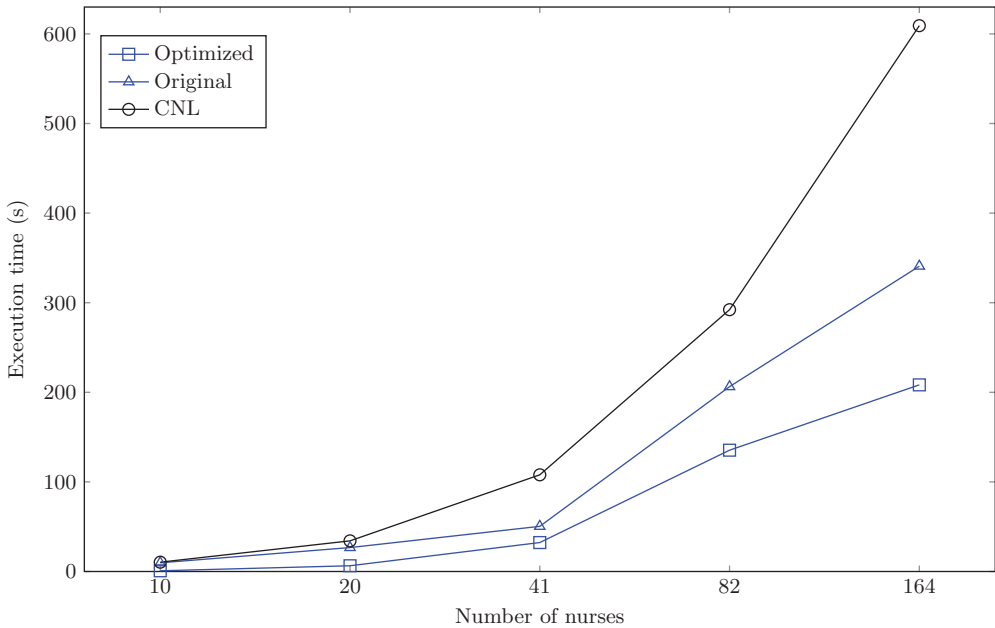
Fig. 2. Time comparison of the performance of the original, the optimized, and the CNL encodings to solve instances of the NSP.

5.2 Manipulation of articulated objects using dual-arm robots

The manipulation of articulated objects is an important task in real-world robot scenarios. Bertolucci *et al.* (2021) presented an ASP framework for handling the manipulation of an articulated object in a 2D workspace with the possibility of performing actions like rotating one of its link with respect to another one around their joint. The framework was composed by five modules, namely Knowledge Base, Goal Checker, Consistency Checking, Action Planner, and Motion Planner. All the modules but Motion Planner were implemented using ASP. In this section, we focus on the Action Planner as described by the encoding reported in the Figure 6 of the paper by Bertolucci *et al.* (2021), since it involves a number of interesting constructs, such as temporal concepts as well as the concept of angle.

The first part of our CNL specifications concerns the definition of the domain of the problem, and it is reported in the following:

```
1 A time is a temporal concept expressed in steps ranging from 1 to 10.
2 A joint is identified by an id.
3 An angle is identified by a value.
4 A position is identified by a joint, by an angle, and by a time.
5 A link is identified by a first joint, and by a second joint.
6 A rotation is identified by a first joint, by a second joint, by a desired angle,
       by a current angle, and by a time.
7 A goal is identified by a joint, and by an angle.
8 granularity is a constant.
9 timemax is a constant.
```

It is possible to observe that we have the concept of time that is marked as temporal. As described in Section 3.1.2, this enables the possibility to use constructs like after,

before, and so forth (as shown at line 2 of the second part of the CNL specifications). Moreover, our CNL implicitly handles the concept of angle, for example, by ensuring that sum operations always create angles whose values are between 0 and 359 degrees.

The second part of the CNL specifications is instead used for solving the problem, and it is reported in the following:

1 Whenever there is a link with a first joint J1, and with a second joint J2, then
     we must have a link with a first joint J2, and with a second joint J1.
2 Whenever there is a time T that is after 0, then we can have at most 1 rotation
     with a first joint J1, with a second joint J2, with a desired angle A, with a
      current angle AI, and with time T such that there is a joint J1, a joint J2,
      an angle A, a link with first joint J1, and with second joint J2, a position
      with joint J1, with angle AI, and with time T.
3 It is required that T is less than timemax, whenever there is a rotation with
     time T.
4 It is required that the first joint J1 of the rotation R is greater than the
     second joint J2 of the rotation R, whenever there is a rotation R with first
     joint J1, with second joint J2.
5 It is required that the desired angle A of the rotation R is different from the
     current angle AI of the rotation R, whenever there is a rotation R with
     desired angle A, and with current angle AI.
6 It is required that the sum between the desired angle A of the rotation R and
     granularity is equal to the current angle AI of the rotation R, whenever
     there is a rotation R with desired angle A greater than 0, with current angle
      AI greater than A.
7 It is required that the sum between the current angle AI of the rotation R and
     granularity is equal to the desired angle A of the rotation R, whenever there
      is a rotation R with current angle AI greater than 0, with desired angle A
     greater than AI.
8 It is required that the difference between 360 and granularity is equal to the
     desired angle A of the rotation R, whenever there is a rotation R with
     desired angle A, and with current angle equal to 0.
9 It is required that the difference between 360 and granularity is equal to the
     current angle AI of the rotation R, whenever there is a rotation R with
     desired angle A equal to 0, and with current angle AI.
10 Whenever there is a joint J, whenever there is a time T, then we can have a
     position with joint J, with angle A, and with time T to exactly 1 angle A.
11 It is required that the angle A1 of the position P1 is equal to the angle A2 of
     the position P2, whenever there is a position P1 with joint J, with angle A1,
      and with time T, whenever there is a position P2 the next step with joint J,
      and with angle A2, whenever there is not a rotation with time T less than or
      equal to timemax.
12 It is required that the angle A1 of the position P is equal to the desired angle
     A2 of the rotation R, whenever there is a position P with joint J1, with time
      T, with angle A1, whenever there is a rotation R the previous step with
     first joint J1, and with desired angle A2.
13 It is required that the angle AN of the position P is equal to |AC+(A-AP)+360|,
     whenever there is a time T, whenever there is a position P the next step with
      joint J1, and with angle AN, whenever there is a rotation with first joint
     J2, with desired angle A, with current angle AP, and with time T, whenever
     there is a position P2 with joint J1 greater than J2, with angle AC, and with
      time T.
14 It is required that the angle A1 of the position P1 is equal to the angle A2 of
     the position P2, whenever there is a position P1 with joint J1, with angle A1
     , and with time T, whenever there is a position P2 with joint J1, and with

Table 1. *Comparison of time (in seconds) employed by the original encoding and by the CNL encoding to compute a solution within 10 steps or to prove that there is no solution*

Instance	12 joints, 180 granularity		12 joints, 90 granularity		14 joints, 180 granularity	
	Original	CNL	Original	CNL	Original	CNL
1	0.01	0.01	49.7	8.4	0.03	0.06
2	0.01	0.01	0.1	0.3	0.04	0.09
3	0.01	0.01	51.3	2.8	0.03	0.03
4	0.01	0.03	24.8	4.3	0.01	0.03
5	0.01	0.01	20.7	3.8	0.01	0.01
6	0.01	0.01	27.7	5.8	0.01	0.04
7	0.01	0.01	0.1	0.3	0.01	0.01
8	0.01	0.01	14.7	1.8	0.01	0.05
9	0.01	0.01	0.06	0.1	0.01	0.05
10	0.01	0.01	0.1	0.2	0.01	0.03

```
     angle A2, and with the next step respect to T, whenever there is a rotation
     with first joint J2 greater than J1, and with time T not after timemax.
15 It is required that the angle A1 of the goal G is equal to the angle A2 of the
     position P, whenever there is a goal G with joint J, with angle A1, whenever
     there is a position P with joint J, with angle A2, and with time equal to
     timemax.
```

Here, due to the structure of the problem, we found more natural to use positive constraints.

Comparison of the performances. The encoding generated by the CNL specifications described before has been compared to the original one proposed by Bertolucci *et al.* (2021), referred to as *Original*. In particular, we considered all the instances with 12 and 14 joints, with granularity equal to 180 and 90, and we set the number of maximum steps equal to 10. Such instances represent the biggest ones in terms of number of joints and granularity. Results are shown in Table 1, where we report, for both the original and the generated encodings, the time (expressed in seconds) for computing a solution within the maximum number of time steps, or to prove that there is no a solution within such a limit. It is possible to observe that there is no overhead introduced by the CNL encoding, which is actually faster than the original one on some instances. In particular, we observed that the generated encoding is faster on instances where there is no solution within 10 time steps (i.e., unsatisfiable instances). This difference seems to be related to the structure of the encodings, since the original encoding uses some direct rules to compute the position of joint angles which are not modified in a given time step, whereas the same task is performed by the generated encoding by using a choice rule and some constraints. This structure seems to be heuristically preferred by the solver.

5.3 Chemotherapy treatment scheduling (CTS) problem

The CTS problem is a complex problem taking into account different constraints and resources. In this section, we consider a simplified version of the problem described by Dodaro *et al.* (2021) that presented a case study based on the requirements of an Italian

hospital. The idea here is to focus on the main constraints and optimization statements that are useful to show the capabilities of our tool, without considering all the variants described by Dodaro *et al.* (2021). In particular, the CTS problem consists of assigning a starting hour to the treatment of all the patients, and to the phases before the treatment, where the phases are *(i)* the admission to the hospital, *(ii)* the blood collection, and *(iii)* the medical check. Moreover, during the treatment, each patient must be assigned either to a bed or a chair. A proper solution to the CTS problem requires the satisfaction of a number of constraints, for example, the starting time of the admission to the hospital must be after the opening time of the hospital, patients with long therapy must be assigned after 11:20 AM, and each bed or chair must be assigned to just one patient at a time. Finally, every patient has a preference between chairs and beds and the solution should try to maximize the number of patients assigned to the preferred resource.

The first part of our CNL specifications concerns the definition of the domain of the problem, and it is reported in the following:

```
1 A timeslot is a temporal concept expressed in minutes ranging from 07:30 AM to
     01:30 PM with a length of 10 min.
2 A day is a temporal concept expressed in days ranging from 01/01/2022 to
     07/01/2022.
3 A patient is identified by an id and has a preference.
4 A registration is identified by a patient, and by an order, and has a number of
     waiting days, a duration of the first phase, a duration of the second phase,
     a duration of the third phase, and a duration of the fourth phase.
5 A seat is identified by an id, and has a type.
6 An assignment is identified by a registration, by a day, and by a timeslot.
7 A position in is identified by a patient, by an id, by a timeslot, and by a day.
```

The second part of the CNL defines the CTS problem, and it is reported in the following:

```
1 Whenever there is a registration R with an order equal to 0, then R can have an
     assignment to exactly 1 day, and timeslot.
2 Whenever there is a registration R with patient P, with order OR, and with a
     number of waiting days W, whenever there is an assignment with registration
     patient P, with registration order OR-1, and with day D, whenever there is a
     day with day D+W, then we can have an assignment with registration R, and
     with day D+W to exactly 1 timeslot.
3 It is required that the sum between the duration of the first phase of the
     registration R, the duration of the second phase of the registration R, and
     the duration of the third phase of the registration R is greater than the
     timeslot of the assignment A, whenever there is a registration R, whenever
     there is an assignment A with registration R, with timeslot T.
4 Whenever there is a patient P, whenever there is an assignment with registration
     patient P, with timeslot T, and with day D, whenever there is a registration
     R with patient P, and with a duration of the fourth phase PH4 greater than 0,
     then P can have a position with id S, with timeslot T, with day D in exactly
     1 seat S for PH4 timeslots.
5 It is required that the number of patient that have position in id S, day D,
     timeslot TS is less than 2, whenever there is a day D, whenever there is a
     timeslot TS, whenever there is a seat with id S.
6 It is required that the assignment A is after 11:20 AM, whenever there is a
     registration R with a duration of the fourth phase greater than 50 timeslots,
     whenever there is an assignment A with registration R.
```
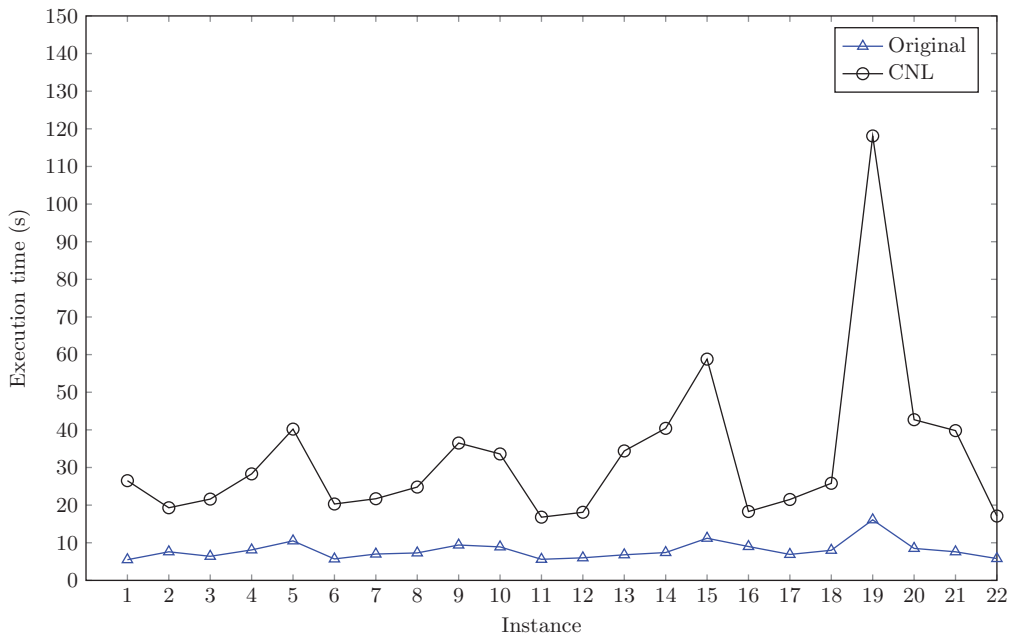
Fig. 3. Time comparison of the performance of the original and CNL encodings to solve instances of the CTS problem.

```
7 It is preferred as much as possible, with high priority, that a patient P with
    preference T has a position in a seat S, whenever there is a seat S with type
    T.
```

Here, we want to emphasize the simplicity of using specific constructs for temporal concepts like the time slot, as done in the sentence at line 6, where we state that an assignment is after 11:20 AM.

Comparison of the performances. The encoding generated by the CNL specifications described before has been compared to the original one proposed by Dodaro *et al.* (2021), referred to as *Original*. The results are presented in Figure 3. As expected, the original encoding is in general faster than the generated encoding. Nevertheless, the performance of generated encoding is still satisfactory, since on average it requires 32 s to compute a solution, with a peak of 2 min on the hardest instance.

6 Preliminary user validation

In this section, we present an analysis conducted to assess the usability and readability of the proposed CNL. The test was conducted on August 1, 2023, and involved 10 individuals among doctoral students and researchers from the Department of Mathematics and Computer Science at the University of Calabria. It is worth noting that 5 participants work with ASP daily and can be considered experts, while the other 5 work on different research topics. Additionally, 7 participants had attended at least one course on ASP during their studies, whereas the others attended only short seminars about ASP. The tool was not introduced beforehand, and the content of the experiment was not

Table 2. *Results on the usability test. Each row represents the results of an individual participant. A value of "1" indicates that the provided ASP rule/CNL specification was correct, while "0" indicates that it was incorrect*

Language	Main research area	Attended ASP course?	c_1	c_2	c_3	c_4
ASP	ASP Systems and Tools	Y	1	1	1	1
ASP	ASP Systems and Tools	Y	1	1	1	1
ASP	ASP Semantics and Theory	Y	1	1	1	0
ASP	Deep Learning	Y	0	0	0	0
ASP	Deep Learning	Y	0	0	0	0
CNL	ASP Systems and Tools	Y	1	0	1	0
CNL	ASP Systems and Tools	Y	0	0	0	0
CNL	Theoretical Computer Science	N	0	1	0	0
CNL	Deep Learning	N	0	0	0	0
CNL	Deep Learning	N	0	0	0	0

announced in advance. Moreover, we ensured that: *(i)* participants had no prior experience with CNL2ASP; *(ii)* the set of participants did not exclusively consist of individuals interested in tools or those with specific biases toward using programming environments; *(iii)* the set of participants included a mix of both proficient and less proficient ASP programmers, which is the expected target of users. Indeed, we believe that a limited experience on ASP or at least on declarative languages for solving combinatorial problems might be needed to proficiently use the tool.

Finally, we mention that this analysis should be considered preliminary due to the limited number of participants, and none of them had received prior training on CNL2ASP.

6.1 Usability

We designed a test in which participants were asked to solve the following problem: Given a set of n persons and m teams (assuming $n > m$), the goal is to assign persons to teams while satisfying the following conditions:

- Each person must be assigned to exactly one team (c_1).
- Each team can have a maximum of 4 persons (c_2).
- Two persons who are incompatible cannot be on the same team (c_3).
- If possible, two friends should be placed in the same team (c_4).

We divided the participants into two groups. The first group was expected to use ASP to solve the problem, while the second group was instructed to use our CNL. The test began with a brief description of the task and some basic instructions on the CNL syntax for the second group. To ensure a fair comparison, individuals who had never attended an ASP course were included in the second group.

The results are presented in Table 2. As expected, participants familiar with ASP were able to create an ASP program that successfully addressed the given problem. In contrast, individuals who had taken an ASP course during their studies but were not actively using ASP were unable to solve the problem.

Regarding the second group, the best performance came from a researcher who also had experience with ASP, achieving partial success in solving the problem. Interestingly,

Table 3. *Results on the readability test. Each row represents the results of an individual participant. A "1" indicates that the participant correctly identified the truth or falsity of the corresponding statement, while "0" denotes an incorrect or an empty response*

ASP	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	Number of correct answers
ASP	0	1	1	0	1	0	0	0	1	0	4
ASP	0	1	1	0	1	0	0	1	1	0	5
ASP	0	1	1	1	0	0	0	0	0	0	3
ASP	1	1	1	1	1	1	0	1	1	0	8
ASP	1	1	1	1	1	1	0	1	1	0	8
CNL	1	1	1	0	1	0	0	1	1	0	6
CNL	1	1	1	1	1	1	1	1	1	1	10
CNL	1	0	1	1	1	1	1	1	0	1	8
CNL	1	1	1	1	1	0	0	1	1	0	7
CNL	0	0	0	1	1	1	0	0	0	0	3

one of the researchers who did not work with ASP managed to correctly specify condition c_2. Consequently, even without prior training, 2 out of 5 participants in this group were able to specify some of the problem's conditions accurately.

6.2 Readability

We designed a test in which participants were required to determine the truth or falsity of the following statements:

1. After two consecutive nights there is a special rest day.
2. Each nurse has at least 2 rest days every two weeks.
3. Each nurse has exactly 30 days of holidays.
4. A nurse can work at most two consecutive nights.
5. Each nurse has at most 30 days of holidays.
6. A nurse can work at most three consecutive nights.
7. A special rest day must be provided when a nurse is in vacation.
8. Each nurse can be assigned to at most "maxNight" nights shift during the whole year.
9. Each nurse can be assigned to at least "minNight" nights shift during the whole year.
10. Each nurse should be assigned to exactly "balanceNurseNight" nights shift during the whole year.

Subsequently, we grouped the participants in the same manner as in the usability experiment. The first group was provided with the ASP encoding for the NSP as described by Dodaro and Maratea (2017). The second group received the CNL specifications described in Section 5.1. To alleviate social pressure, we requested that participants remain anonymous during this test. The fifth statement (s_5) was contested as ambiguous, as it can be interpreted as both true and false. Therefore, we assigned a score of 1 for both true and false responses and 0 if the answer was left blank. Results are reported in Table 3. On average, participants in the CNL group obtained a score of 6.8 with a peak of 10, whereas participants in ASP group obtained a score of 5.6 with a peak of 8.

7 Related work

In this section, we overview the main CNLs proposed in the area of logic programming; for a complete review of CNL, we refer the reader to the interesting survey by Kuhn (2014).

One of the first attempts of designing encoding expressed in a CNL as logic programs was presented by Fuchs and Schwitter (1995) and by Schwitter *et al.* (1995), where Attempto CNL (Fuchs 2005) was proposed, whose idea was to convert sentences expressed in a CNL as Prolog clauses. Clark *et al.* (2005) presented a Computer-Processable Language, whose key principle was to be easier for computers rather than a language easier for users. Moreover, they presented also an interpreter and a reasoner for this language and discussed the strengths and weaknesses of natural languages to be used as a the basis for knowledge acquisition and representation.

Concerning ASP, Erdem and Yeniterzi (2009) proposed BIOQUERYCNL, a CNL for biomedical queries, and developed an algorithm designed to automatically encode a biomedical query expressed in this language as an ASP program. BIOQUERYCNL is a subset of Attempto CNL, and it can represent queries of the form *Which symptoms are alleviated by the drug Epinephrine?* (we refer the reader to Chapter 3 of Erdem and Yeniterzi 2009 for more queries). Later on, BIOQUERYCNL was also used as a basis to generate explanation of complex queries (Öztok and Erdem 2011). The main difference with our approach is that CNL2ASP does not cover query answering and is not specialized on one particular application context.

Baral and Dzifcak (2012) proposed a CNL specific for solving logic puzzles. The CNL was split into two sets of sentences, namely *Puzzle Domain data* and *Puzzle clues*. The former plays a similar role of our explicit domain definitions (see Section 3.1), whereas Puzzle clues can be seen as the logic rules to solve the puzzle. As in our case, the CNL was then automatically converted into ASP rules.

Lifschitz (2022) showed the process of translating the English sentence "A prime is a natural number greater than 1 that is not a product of two smaller natural numbers." into executable ASP code.

Schwitter (2018) defined the language PENG^{ASP}, a CNL that is automatically converted into ASP. Albeit some aspects of PENG^{ASP}'s grammar rules are present in the grammar of our CNL, the latter is geared more toward the formal definition of combinatorial problems in a natural feeling and unambiguous way that is also reliably predictable in its translation to ASP, choosing words that would stand out easily during reading and with an easily deducible meaning from the given context; this meant sacrificing some of the naturalness of PENG^{ASP}. In addition, the grammar of PENG^{ASP} is designed for allowing a conversion from the CNL to ASP and then back in the other direction, whereas in CNL2ASP this possibility is not yet available, although the language has been designed in such a way that it should be possible to make it viable. Another feature that is available in PENG^{ASP} is the possibility to express queries, which is not possible in our CNL. However, our CNL presents some features that, to best of our knowledge, are not available in PENG^{ASP}, such as explicit definitions, and positive strong constraints, that we found to be useful for specifying real-world problems in a natural way. Moreover, it should be noted that the implementation of PENG^{ASP}, as well as a binary executable, is not yet public, whereas the implementation of CNL2ASP is open source and

publicly available. As an example of the differences with our CNL, we report a comparison with the CNL for specifying the graph coloring problem used by PENG^ASP (Figure 5 of Schwitter 2018[2]).

```
1  The node 1 is connected to the nodes 2 and 3.
2  The node 2 is connected to the nodes 1 and 3.
3  The node 3 is connected to the nodes 1 and 2.
4  Red is a color.      Green is a color.      Blue is a color.
5  Every node is assigned to exactly one color.
6  It is not the case that a node X is assigned to a color and a node Y is assigned
       to the color and the node X is connected to the node Y.
```

There are two main differences with our CNL presented in Section 4.1. The first one is that our CNL must use variables (i.e., X in our example) also to specify the connections, whereas the one of PENG^ASP does not need it. In our case, sentence at line 1 would create the atom `connected_to(1,2,3)`. Second, the last sentence is expressed in a negative form in case of PENG^ASP, which is similar to the concept of constraint in ASP, whereas our CNL uses a positive sentence which is similar to the concept of clause in propositional logic. Moreover, the PENG^ASP and the CNL2ASP methodologies differ in the way the sentences are processed before being unified with the grammar rules. First of all, the grammar rules for PENG^ASP are specified with a Definite Clause Grammar (DCG), while in our solution the grammar is defined in Extended Backus-Naur Form. Moreover, our tool builds a sort of syntax tree for handling the internal structure of the sentences before rewriting them into ASP. While in PENG^ASP, after the collection in the DCG, a chart parser is used to extract the information needed for the translation and this information can be parsed and passed to the users to help with completing the sentence.

We also mention that some of the sentences used in the CNL presented in this paper are inspired by the SBVR (Bajwa *et al.* 2011; The Business Rules Group 2000), which is a standard proposed by the Object Management Group to formally describe complex entities, for example, the ones related to a business, using natural language.

In Table 4, we present a comparison of the features of the different CNLs translating to ASP. In particular, we want to highlight the constructs that are covered by the CNLs in order to ease the usage of the tool and to be more adherent to natural language. We considered the same constructs considered in Schwitter *et al.* (1995) plus temporal sentences and new constructs specifically adopted for ASP: cardinality constraints, aggregates, and preferences.

Comparison to previous work. This paper represents an extended version of the paper (Dodaro *et al.* 2022) presented at the 4th International Workshop on the Resurgence of Datalog in Academia and Industry (DATALOG 2.0 2022). In this paper, the following additional contributions are provided:

1. We extended the CNL to support some of the ASP constructs missing in the previous paper, such as disjunctive rules and `#min` and `#max` aggregates.
2. We extended the CNL to include explicit definitions and temporal constructs, which we found useful in practice to have a natural description of complex real-world problems.

---

[2] Since PENG^ASP is not publicly available, we could not compare the two languages on the other problems used in our paper.

Table 4. *Comparison of the linguistic features of CNL2ASP, λ-based (Baral and Dzifcak 2012), BIOQUERYCNL (Erdem and Yeniterzi 2009), and PENG^{ASP} (Schwitter 2018). Yes (Y) means the construct is supported, No (N) means that the construct is not supported, Unknown (U) means that there is no evidence that the construct is supported nor unsupported*

Characteristic	CNL2ASP	λ-Based	BIOQUERYCNL	PENG^{ASP}
Simple sentences	Y	Y	Y	Y
Modifying clauses	Y	Y	Y	Y
Comparative clauses	Y	Y	Y	Y
Conjunction/disjunction clauses	Y	N	Y	Y
Conditional sentences	Y	N	N	Y
Negated sentences	Y	Y	N	Y
Cardinality constraints	Y	N	Y	Y
Aggregates	Y	N	N	U
Temporal sentences	Y	Y	N	Y
Preferences	Y	N	N	Y
Queries	N	Y	Y	Y

3. We extended the CNL with novel constructs such as `whenever/then` clauses that should also make more intuitive the CNL for ASP users.
4. We included as use cases three additional real-world problems and, for each of them, we executed an experimental analysis comparing the performance of the generated encoding with the one produced by ASP practitioners.
5. We performed a preliminary user validation, conducted to assess the usability, and the readability of the CNL.
6. We extended the presentation by providing more details about the architecture of CNL2ASP and also by discussing related work.

8 Conclusion

In this paper, we proposed a tool for automatically converting English sentences expressed using a controlled language into ASP rules. Moreover, we provided several examples of combinatorial problems that can be specified using our CNL, and their translations as ASP rules. Concerning future work, the CNL supported by CNL2ASP might be extended to cover additional constructs and language extensions, for example, temporal operators (Cabalar *et al.* 2020) or Constraint ASP (CASP) (Balduccini 2011; Banbara *et al.* 2017), and the tool can be extended to implement a bidirectional conversion, where ASP rules are translated into CNL statements. Another interesting future work can be the integration of novel constructs, similar to temporal ones, and use them for enabling additional features, such as input validation (Alviano *et al.* 2022). It also worth mentioning that the ASP encoding produced by CNL2ASP can be subject of optimization by using tools like the ones presented by Dingess and Truszczynski (2020), and by Liu *et al.* (2022), which automatically rewrite some of the constructs to improve the performance. Finally, we recall that the tool presented in this paper as well as all examples and encodings used in this paper are available at https://github.com/dodaro/cnl2asp.

Supplementary material

Acknowledgments

Conflicts of interest

The authors declare none.

References

ALVIANO, M., CALIMERI, F., DODARO, C., FUSCÀ, D., LEONE, N., PERRI, S., RICCA, F., VELTRI, P. AND ZANGARI, J. The ASP system DLV2. In *Proceedings of LPNMR* 2017, vol. 10377. Springer, 215–221.

ALVIANO, M., DODARO, C. AND MARATEA, M. 2018. Nurse (re)scheduling via answer set programming. *Intelligenza Artificiale 12*, 2, 109–124.

ALVIANO, M., DODARO, C. AND ZAMAYLA, A. 2022. Valasp: A tool for data validation in answer set programming. *Theory and Practice of Logic Programming, accepted to appear* (doi:10.1017/S1471068422000084).

BAJWA, I. S., LEE, M. G. AND BORDBAR, B. 2011. SBVR business rules generation from natural language specification. In *AI for Business Agility*. AAAI.

BALDUCCINI, M. 2011. Industrial-size scheduling with ASP+CP. In *Proceedings of LPNMR*, LNCS, vol. 6645. Springer, 284–296.

BANBARA, M., KAUFMANN, B., OSTROWSKI, M. AND SCHAUB, T. 2017. Clingcon: The next generation. *Theory and Practice of Logic Programming 17*, 4, 408–461.

BARAL, C. AND DZIFCAK, J. 2012. Solving puzzles described in english by automated translation to answer set programming and learning how to do that translation. In *Proceedings of KR*. AAAI Press.

BERTOLUCCI, R., CAPITANELLI, A., DODARO, C., LEONE, N., MARATEA, M., MASTROGIOVANNI, F. AND VALLATI, M. 2021. Manipulation of articulated objects using dual-arm robots via answer set programming. *Theory and Practice of Logic Programming 21*, 3, 372–401.

BREWKA, G., EITER, T. AND TRUSZCZYNSKI, M. 2011. Answer set programming at a glance. *Communications of the ACM 54*, 12, 92–103.

BUCCAFURRI, F., LEONE, N. AND RULLO, P. 2000. Enhancing disjunctive datalog by constraints. *IEEE Transactions on Knowledge and Data Engineering 12*, 5, 845–860.

CABALAR, P., DIÉGUEZ, M., SCHAUB, T. AND SCHUHMANN, A. 2020. Towards metric temporal answer set programming. *Theory and Practice of Logic Programming 20*, 5, 783–798.

CALIMERI, F., FABER, W., GEBSER, M., IANNI, G., KAMINSKI, R., KRENNWALLNER, T., LEONE, N., MARATEA, M., RICCA, F. AND SCHAUB, T. 2020. Asp-core-2 input language format. *Theory and Practice of Logic Programming 20*, 2, 294–309.

CARUSO, S., DODARO, C., MARATEA, M., MOCHI, M. AND RICCIO, F. 2022. Grammar of the CNL. URL: https://github.com/dodaro/cnl2asp/blob/main/src/cnl/grammar/cnl_grammar.lark.

CLARK, P., HARRISON, P., JENKINS, T., THOMPSON, J. A. AND WOJCIK, R. H. 2005. Acquiring and using world knowledge using a restricted subset of english. In *Proceedings of FLAIRS*. AAAI Press, 506–511.

DINGESS, M. AND TRUSZCZYNSKI, M. 2020. Automated aggregator - rewriting with the counting aggregate. In *Proceedings of ICLP Technical Communications*, EPTCS, vol. 325, 96–109.

DODARO, C., GALATÀ, G., GRIONI, A., MARATEA, M., MOCHI, M. AND PORRO, I. 2021. An asp-based solution to the chemotherapy treatment scheduling problem. *Theory and Practice of Logic Programming 21*, 6, 835–851.

DODARO, C. AND MARATEA, M. 2017. Nurse scheduling via answer set programming. In *Proceedings of LPNMR*, LNCS, vol. 10377. Springer, 301–307.

DODARO, C., MARATEA, M. AND RICCIO, F. A tool for encoding controlled natural language specifications as ASP rules. In *Datalog* 2022, CEUR Workshop Proceedings, vol. 3203. CEUR-WS.org, 188–201.

ERDEM, E., GELFOND, M. AND LEONE, N. 2016. Applications of answer set programming. *AI Magazine 37*, 3, 53–68.

ERDEM, E. AND YENITERZI, R. 2009. Transforming controlled natural language biomedical queries into answer set programs. In *Proceedings of the BioNLP Workshop*. Association for Computational Linguistics, 117–124.

FABER, W., PFEIFER, G. AND LEONE, N. 2011. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence 175*, 1, 278–298.

FUCHS, N. E. 2005. Knowledge representation and reasoning in (controlled) natural language. In *Proceedings of ICCS*, LNCS, vol. 3596. Springer, 51–51.

FUCHS, N. E. AND SCHWITTER, R. 1995. Specifying logic programs in controlled natural language. CoRR, abs/cmp-lg/9507009.

GEBSER, M., KAMINSKI, R., KAUFMANN, B., OSTROWSKI, M., SCHAUB, T. AND WANKO, P. 2016. Theory solving made easy with clingo 5. In *Proceedings of ICLP Technical Communications*, OASIcs, vol. 52. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2:1–2:15.

GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming*. MIT Press, 1070–1080.

KUHN, T. 2014. A survey and classification of controlled natural languages. *Computational Linguistics 40*, 1, 121–170.

LIFSCHITZ, V. 2019. *Answer Set Programming*. Springer.

LIFSCHITZ, V. 2022. Translating definitions into the language of logic programming: A case study. In *Proceedings of the ICLP Workshops*, CEUR Workshop Proceedings, vol. 3193. CEUR-WS.org.

LIU, L., TRUSZCZYNSKI, M. AND LIERLER, Y. 2022. A machine learning system to improve the performance of ASP solving based on encoding selection. In *Proceedings of LPNMR*, LNCS, vol. 13416. Springer, 415–428.

ÖZTOK, U. AND ERDEM, E. 2011. Generating explanations for complex biomedical queries. In *Proceedings of AAAI*. AAAI Press.

SCHWITTER, R. 2018. Specifying and verbalising answer set programs in controlled natural language. *Theory and Practice of Logic Programming 18*, 3–4, 691–705.

SCHWITTER, R., HAMBURGER, B. AND FUCHS, N. E. 1995. Attempto: Specifications in controlled natural language. In *Proceedings of the Workshop on Logische Programmierung*, 151–160.

THE BUSINESS RULES GROUP 2000. Defining business rules ∼ what are they really?