A Tool For Reasoning Over CNL Sentences With Temporal Constructs

Pierangela Bruno[1], Simone Caruso[2], Carmine Dodaro[1] and Marco Maratea[1]

[1]DeMaCS, University of Calabria
[2]DIBRIS, University of Genova

Abstract

CNL2ASP is a recent tool designed to convert sentences in controlled natural language (CNL), based on English, into Answer Set Programming (ASP) rules. In this paper, we present an extension of CNL2ASP to support temporal constructs, called CNL2TEL, that facilitates the translation of sentences in this controlled language into rules in temporal equilibrium logic and processable by the TELINGO tool. We demonstrate the effectiveness of CNL2TEL by applying it to some domains of the TELINGO test suite, showcasing the readability and utility of our approach. Additionally, we compare the performance of our translation with TELINGO executed on the original specifications in its native language showing that our tool does not introduce significant overhead.

Keywords

Answer Set Programming, Controlled Natural Language, Temporal Equilibrium Logic, Telingo

1. Introduction

Answer Set Programming (ASP) [1, 2, 3] is a well-known declarative programming paradigm proposed in the area of knowledge representation and reasoning (KRR), and geared toward solving hard combinatorial problems. ASP has been widely used for solving problems in both academic and industrial contexts (see, e.g., [4] for a complete survey on ASP applications). The success of ASP is due to several factors, including its simple syntax and intuitive semantics, the availability of efficient systems [5, 6], and the availability of interesting extensions, e.g., the integration with Constraint Programming and SMT [7, 8, 9], and with machine learning (see, e.g., [10, 11]), and the specification of and reasoning about temporal concepts and constraints [12].

While ASP and other KRR formalisms have been successful in representing and reasoning about complex knowledge domains, they may not be suitable for all types of users and applications. Specifically, the need for formal and logic-based languages can be an entry barrier for non-experts or those without prior experience with KR techniques. Motivated by this observation, recent efforts have focused on developing higher-level languages that are closer to natural language, with automatic translations to ASP [13, 14, 15, 16]. These approaches aim to provide a more intuitive and accessible means of expressing knowledge and rules, thereby expanding the reach and applicability of KR formalisms. In this context, [17] introduced a controlled natural language (CNL) and a novel tool called CNL2ASP, which enables users to specify sentences in high-level English and automatically translate them into ASP rules that can be later on processed by standard ASP systems.

In this paper, we introduce an enhanced version of the CNL2ASP tool. This extended tool, called CNL2TEL, integrates temporal operators and the ability to represent temporal specifications. CNL2TEL is capable of translating these specifications into the language of TELINGO [18], which is the state-of-the-art tool for temporal reasoning within ASP. This allows for the specification of and reasoning about temporal operators, enabling the creation of rules that encompass both past and future references.

Datalog 2.0 2024: 5th International Workshop on the Resurgence of Datalog in Academia and Industry, October 11, 2024, Dallas, Texas, USA

pierangela.bruno@unical.it (P. Bruno); simone.caruso@edu.unige.it (S. Caruso); carmine.dodaro@unical.it (C. Dodaro); maratea@mat.unical.it (M. Maratea)

0000-0002-0832-0151 (P. Bruno); 0000-0002-2724-4342 (S. Caruso); 0000-0002-5617-5286 (C. Dodaro); 0000-0002-9034-2527 (M. Maratea)

To test the viability and effectiveness of our proposal, we apply our solution to some of the use cases within the TELINGO suite. We demonstrate how these problems can be expressed in our CNL and showcase the automatic translation into TELINGO specifications made by CNL2TEL. Additionally, we compare the performance of our translation with TELINGO executed on the original specifications in its native language showing that our tool does not introduce significant overhead.

2. Background

This section introduces needed preliminaries about the CNL2ASP tool, Temporal Equilibrium Logic, and TELINGO.

2.1. CNL2ASP

CNL2ASP is a tool for converting Controlled Natural Language (CNL) sentences into ASP rules. A CNL is a restricted language, in this case English, which is shaped to be processed from a computer although it is flexible enough to be natural to use. Thus, a CNL has a grammar with fixed strings (terminal symbols) that we will identify as quoted strings, e.g., `"prohibited"`, and some non-terminal symbols which are substituted with other rules. While the complete description of CNL2ASP is presented in [17], here we recall some concepts useful to understand the temporal extensions presented later. CNL2ASP takes as input a set of CNL propositions and converts them into a set of ASP rules. Propositions can be either `concept definitions`, i.e., sentences describing the entities of the domain problem, or `standard propositions` that are used for reasoning over the defined concepts and are translated into ASP rules by the tool. Standard propositions are, for example, `constraint propositions` and `guess propositions`.

A concept definition is of the form:

```
concept_defintion ⟶ "A" concept "is identified by" attributes (", and has"
    attributes)?
```

where `concept` and `attributes` are arbitrary names. The first list of attributes represents the definition of the keys, i.e., the attributes that uniquely identify the concept, while the second list defines other attributes and is optional. Then, constraints are of the form:

```
constraint_proposition ⟶ "It is" ("prohibited" | "required") "that"
   condition
```

where `condition` is a sequence of sentences containing aggregates, operations, or other statements. Guess rules and assignments instead are of the form:

```
guess_proposition ⟶ whenever_clauses ", then" entity ("can" | "must")
   predicate cardinality objects
```

where `whenever_clauses` is a list of conditions, `entity` is how the defined entities are accessed and initialized, `predicate` can be seen as the new relation between the entities and it will be the head of the rule. Finally, `objects` is a list of entities that will be translated into the condition in the head of the rule. For a complete description of the CNL, we refer the reader to [17].

2.2. Temporal Equilibrium Logic

Temporal Equilibrium Logic (TEL) is an extension of propositional logic with past and future temporal operators shown in the first column of Table 1. The initial and final operators, in the first row, exclusively hold in the initial and final state, respectively. Previous and next, instead, are unary operators that are used to check if a proposition (p) is true, in the previous or next state, respectively. They also have a corresponding weak, eventually and always version. The weak previous operator is defined as $\bullet p \vee \mathbf{I}$, while the weak next as $\circ p \vee \mathbb{F}$, i.e., p must be true in the previous (next) state or it is the initial (final) state. Then, the eventually before (after) and always before (after) operators have the intuitive meaning

Table 1
TEL operators with their corresponding TELINGO and CNL syntax. The operators referring to the past precede those referring to the future.

TEL	TELINGO	CNL
I (initial)	&initial	is the initial state
• (previous)	$< A$ $'A$	before A
$\widehat{\bullet}$ (weak previous)	$<: A$	before A or it is the initial states
⧫ (eventually before)	$<? A$	before A that eventually holds eventually A that holds since before
■ (always before)	$< * A$	before A that always holds always A that holds since before
S (since)	$A <? B$	A since B
T (trigger)	$A < * B$	A trigger(s) B
$\mathbb{F}$ (final)	&final	is the final state
○ (next)	$> A$ A'	after A
$\widehat{\circ}$ (weak next)	$>: A$	after A or it is the final state
◊ (eventually after)	$>? A$	after A that eventually holds eventually A that holds since after
□ (always after)	$> * A$	after A that always holds always A that holds since after
$\mathbb{U}$ (until)	$A >? B$	A until B
$\mathbb{R}$ (release)	$A > * B$	A release(s) B

that a proposition holds in at least one of the previous (next) states and in all the previous (next) states, respectively. Finally, the binary operators since, trigger, until, and release are defined as follows:

- a **S** b is true whenever there is a state in which b is satisfied and then, in the following state, a is satisfied;
- a **T** b is true whenever b becomes true from the state in which a became true;
- $a \mathbb{U} b$ is true whenever there is a sequence of states in which a is true and it becomes false in the state in which b becomes true;
- $a \mathbb{R} b$ is true whenever b holds until and including the state in which a becomes true. If a never becomes true, then b must be always true.

For a more detailed and formal description of the temporal operators, we refer the reader to [19]. As [18] shows, any temporal formula can be translated into a temporal logic program made of three types of rules:

- initial rules: $A \rightarrow B$
- dynamic rules: $\widehat{\circ}\Box(B \rightarrow A)$
- final rules: $\Box(\mathbb{F} \rightarrow (B \rightarrow A))$

where, given an alphabet $\mathcal{A}$, B and A are defined as follows: $B = b_1 \wedge \cdots \wedge b_n$ with $n \geq 0$, $A = a_1 \vee \cdots \vee a_m$ with $m \geq 0$, having for dynamic rules b_i and a_j as temporal literals $\{a, \neg a, \bullet a, \neg \bullet a, \mid a \in \mathcal{A}\}$, while for initial and final rules as regular literals $\{a, \neg a \mid a \in \mathcal{A}\}$. Naturally, initial and final rules allow to define the initial and final conditions, and dynamic rules define the state transitions. Finally, it is possible to convert any temporal logic program into a regular one, i.e., a program only made of initial rules, adorning literals with an explicit timestamp. The timestamp, in fact, allows to make rules applicable only in a certain time point. Moreover, also temporal operators can be timestamped, e.g., consider the previous operator ($\bullet$) applied to a proposition p at a time point k ($\bullet p_k$): it can be converted into p_{k-1}. Thus, by iterating this process it is possible to increase at each step the time horizon by 1.

2.3. TELINGO

TELINGO is a solver for temporal programs, based on TEL on finite traces. TELINGO makes usage of the CLINGO theory introducing &tel, &initial and &final atoms, temporal operators and Boolean operators. Thus, in TELINGO, a temporal formula is defined as &tel{φ}, where φ is made of temporal operators, whose syntax is shown in the second column of Table 1, and Boolean operators. Moreover, the unary operators next and previous can be represented with a single quote, suffixed (e.g., `predicate'(X)`) and prefixed (e.g., `'predicate(X)`) to the predicate to which they are referred, respectively. Initial, dynamic and final rules are achieved leveraging the CLINGO's $\#program$ directive that allows to split a program into subprograms. Thus, TELINGO defines the three corresponding programs: $initial$, $dynamic$, and $final$, and one more program, called $always$, that is a combination of $initial$ and $dynamic$, i.e., rules in this program apply both in $initial$ and $dynamic$. All the rules outside any program are considered part of $initial$. Finally, temporal programs are converted into regular programs and then solved by CLINGO, using its multi-shot solving capability, where a loop iteratively increments the time horizon until a stopping criterion is met. Such a criterion can be set by the three options controlling the loop: `-imin` and `-imax`, used to set the minimum and maximum solving steps, respectively, and `-istop` whose default value is `sat` but it can be also set to `unsat` or unknown.

3. CNL grammar with temporal constructs

CNL2TEL extends CNL2ASP to support the new concepts introduced in TELINGO, summarized into three main elements: *(i)* well-defined program parts, whose rules apply in particular states; *(ii)* the possibility of being able to refer to an atom in the previous, subsequent and initial state; and *(iii)* the temporal formulas, detailed below.

Concerning *(i)*, TELINGO defines four program parts, namely $initial$, whose rules apply only to the first state, $always$, whose rules apply to all the states, $dynamic$, whose rules apply to all states except the initial state, and $final$, whose rules apply only to the last state. Therefore, we introduce `program` in the grammar:

`program` ⟶ `TEMPORAL_PART? (standard_proposition END_OF_LINE)+`

that is made of the old `standard_proposition` token, i.e., the CNL's constraints, guess and assignment rules shown in Section 2.1, adorned with the optional token TEMPORAL_PART mapped as follows:

```
TEMPORAL_PART ⟶ "The following propositions apply in the initial state:" |
    "The following propositions always apply:" | "The following
   propositions always apply except in the initial state:" |"The following
   propositions apply in the final state:"
```

Intuitively, the sentences correspond to the $initial$, $always$, $dynamic$ and $final$ programs, respectively, and, as TEMPORAL_PART is optional, when it is not declared the propositions are considered by TELINGO part of $initial$.

Concerning *(ii)*, to refer to the concepts in the previous, subsequent, and initial state, we extended the `entity` token as follows:

`temporal_entity` ⟶ `TELINGO_ENTITY_STATE entity`

where TELINGO_ENTITY_STATE is one of `"previously"`, `"subsequently"` or `"initially"`. The following examples should clarify their usage:

```
1 Whenever there is previously a gun unloaded, whenever there is not a gun
    loaded then we must have a gun with status equal to unloaded.
2 It is prohibited that there is a gun loading, whenever there is not
    subsequently a gun loaded.
```

```
3 It is required that there is a gun loading, whenever there is initially a
    gun unloaded.
```

Concerning *(iii)*, temporal formulas are concatenations of TELINGO operators:

```
TELINGO_TEMPORAL_OPERATOR ⟶ ","? ("always" | "eventually" | "before" | "
    since before" | "after" | "since after") ("this state" | "now" | "here")
    ? ","?
hold_condition ⟶ ("that" VERB_NEGATION? TELINGO_TEMPORAL_OPERATOR "hold")
    | ("that" VERB_NEGATION? "hold" TELINGO_TEMPORAL_OPERATOR)
telingo_formula ⟶ "there is" VERB_NEGATION? TELINGO_TEMPORAL_OPERATOR?
    telingo_operand hold_condition? (TELINGO_BINARY_OPERATOR telingo_formula
    )?
```

where the combination of TELINGO_TEMPORAL_OPERATOR and hold_condition allows to specify the temporal operators. The full list of temporal operators supported in TEL with the corresponding TELINGO and CNL syntax is shown in Table 1.

Instead, the optional elements TELINGO_BINARY_OPERATOR and telingo_operation are used to concatenate temporal formulas where TELINGO_BINARY_OPERATOR includes all the temporal and Boolean operators that accept two operands:

```
TELINGO_BINARY_OPERATOR ⟶ "and" | "or" | "implies" | "imply" | "equivalent
    " | "trigger" | "since" | "precede" | "release" | "until" | "follow"
```

Finally, telingo_operand is defined as:

```
telingo_operand ⟶ entity (TELINGO_BINARY_OPERATOR telingo_operand)?
| TELINGO_CONSTANT (TELINGO_BINARY_OPERATOR telingo_operand)?
TELINGO_CONSTANT ⟶ "it is the initial state" | "it is the final state"
                 | "the true constant" | "the false constant"
```

thus, a telingo_operand can be an entity or a TELINGO_CONSTANT (initial and final constants in Table 1), while the optional pair TELINGO_BINARY_OPERATOR telingo_operand is used for concatenation. The telingo_formula can be used in whenever_clauses and constraint_proposition, presented before. The following sentences are some possible examples:

```
1 Whenever there is a gun shooting, whenever, before now, there is a gun
    unloaded that always holds and there is eventually a gun shooting that
    holds since before, then we must have a gun with status equal to broken.
2 It is prohibited that, after now, there is a gun loaded and a gun shooting
    that does not always hold.
```

The first sentence shows the usage of the keywords before used with always holds, and eventually with holds since before which defines the TELINGO operator always before (<*) and eventually before (<?), respectively. The second sentence shows the keyword after with not always hold, and defines the operator always after (>*) followed by the Boolean negation. Moreover, the keyword and represents a Boolean conjunction.

4. Use cases

In this section, we present the CNL specifications for a selection of the domain examples taken from the TELINGO repository (https://github.com/potassco/telingo), namely Gun Problem, Tower of Hanoi, and Logistic problem, containing the most significant constructs introduced. Moreover, in this section, we report the results of an experimental analysis conducted on the aforementioned domains, where we compare TELINGO executed on the original program and on the program generated by CNL2TEL. The CNL2TEL encodings of the analyzed domains and the generated instances are available at https://github.com/simocaruso/datalog24cnl2tel.

Gun Problem. In the gun problem, there is a gun that can either shoot, wait, or load. Whenever the gun shoots two times without loading, then it breaks. In the following, we present the CNL specification of the problem with the corresponding translation.

First, we define the concepts of the problem (lines 1–2) and then the *initial* state:

```
1 A gun is identified by a status.
2 A shooter is identified by an id.

3 The following propositions apply in the initial state:
4 There is a gun with status equal to unloaded.
```

as concept definitions (lines 1–2) do not have a corresponding TELINGO representation, this block is translated into the two following rules:

```
1 #program initial.
2 gun("unloaded").
```

Then, it is defined an *always* program:

```
5 The following propositions always apply:
6 There is a shooter with id 1.
```

which is translated into:

```
3 #program always.
4 shooter(1).
```

The following set of CNL propositions, instead, constitute a *dynamic* program.

```
7 The following propositions always apply except in the initial state:
8 Whenever there is a shooter X, then we must have a gun with status equal to
    shooting, or a gun with status equal to loading, or a gun with status
    equal to waiting.
9 Whenever there is a gun loading then we must have a gun with status equal to
     loaded.
10 Whenever there is not a gun unloaded, whenever there is previously a gun
    loaded then we must have a gun with status equal to loaded.
11 Whenever there is a gun shooting, whenever there is previously a gun loaded,
     whenever there is not a gun broken, then we must have a gun with status
     equal to unloaded.
12 Whenever there is previously a gun unloaded, whenever there is not a gun
    loaded then we must have a gun with status equal to unloaded.
13 It is prohibited that there is a gun loading, whenever there is previously a
     gun loaded.
14 Whenever there is a gun shooting, whenever, before now, there is a gun
    unloaded that always holds and there is eventually a gun shooting that
    holds since before, then we must have a gun with status equal to broken.
15 Whenever there is previously a gun broken, then we must have a gun with
    status equal to broken.
```

It is first defined which action can be selected, then how that gun updates its status based on the selected action, and finally the condition that if a gun shoots two times without loading, then it breaks. The corresponding TELINGO encoding is presented below:

```
5 #program dynamic.
6 gun("shooting") | gun("loading") | gun("waiting") :- shooter(X).
7 gun("loaded") :- gun("loading").
8 gun("loaded") :- not gun("unloaded"), 'gun("loaded").
```

```
9 gun("unloaded") :- gun("shooting"), 'gun("loaded"), not gun("broken").
10 gun("unloaded") :- 'gun("unloaded"), not gun("loaded").
11 :- gun("loading"), 'gun("loaded").
12 gun("broken") :- gun("shooting"), not not &tel {(<* gun("unloaded")) & (<
    <? gun("shooting"))}.
13 gun("broken") :- 'gun("broken").
```

Notice how the `previously` operator has been converted into 'gun("loaded") in the line 8 of the encoding and how the temporal formula of the CNL in line 14 (`whenever there is before a gun unloaded that always holds and there is eventually a gun shooting that holds since before`) has been converted into the complex TELINGO formula in line 12 (`not not &tel {(<* gun("unloaded"))& (< <? gun("shooting"))}`). Finally, there is the goal of the problem, in which we ensure that the gun shoots at least once, as defined in the following:

```
16 The following propositions apply in the final state:
17 It is prohibited that, before here, there are not a gun loaded and a gun
    shooting that eventually hold.
```

Again, the CNL presents a temporal formula that is converted into:

```
14 #program final.
15 :- not &tel {<? (gun("loaded") & gun("shooting"))}.
```

Tower of Hanoi. The following are the specifications for the Tower of Hanoi problem:

```
1 A disk is identified by an id.
2 A peg is identified by an id.
3 A goal is identified by a disk, and by a peg.

4 The following propositions always apply:
5 A disk ranges from 0 to 3.
6 A peg ranges from 1 to 3.

7 There is a goal with disk id 3, with peg 3.
8 There is a goal with disk id 2, with peg 3.
9 There is a goal with disk id 1, with peg 3.

10 The following propositions apply in the initial state:
11 Every disk X must be on peg 1, where X is greater than 0 and X is less than
    4.

12 The following propositions always apply except in the initial state:
13 Whenever there is a disk D, then D can be moved to a peg.
14 It is required that the number of disks that are moved to a peg is equal to
    1.
15 A disk D is on a peg P when disk D is moved to peg P.
16 A disk D is moved when disk D is moved to a peg P.
17 A disk D is on a peg P when disk D is previously on peg P and also disk D is
     not moved.
18 A disk X is blocked in peg P when disk D is previously on peg P, where X is
    equal to D-1.
19 A disk X is blocked in peg P when disk D is blocked in peg P, where X is
    equal to D-1.
```

```
20 It is prohibited that a disk D is moved to a peg P, when a disk X is blocked
       in peg P, where X is equal to D-1.
21 It is prohibited that a disk D is moved to a peg P1, when disk D is
     previously on peg P2 and also disk D is blocked in peg P2.

22 The following propositions apply in the final state:
23 It is prohibited that disk D is not on peg P, whenever there is a goal with
     disk id D, with peg id P.
```

Similarly to the previous problem, first we have the declaration of the concepts of the problem (lines 1–3), then we have a definition of the *always* program (line 4) that is made of a series of propositions that declare the facts, i.e. the disks, pegs, and goal of the problem (lines 5–9). Then, we have a program part that applies in the initial state in which all disks are put on peg 1 (lines 10 and 11) and a definition of a *dynamic* program (lines 12–21), where we state that one disk at the time can be moved, the constraints to respect and how the disk position is updated accordingly. Finally, it is defined the goal of the problem (lines 22 and 23), that is to have all the disks on the desired peg. As for the gun problem, it can be noticed the usage of the TEMPORAL_ENTITY_STATE `previously` to refer to a previous state. The resulting TELINGO encoding is:

```
1 #program always.
2 disk(0..3).
3 peg(1..3).
4 goal(3,3).
5 goal(2,3).
6 goal(1,3).

7 #program initial.
8 on(X,1): peg(1) :- X > 0, X < 4, disk(X).

9 #program dynamic.
10 {moved_to(D,PG_D): peg(PG_D)} :- disk(D).
11 :- #count{D: moved_to(D,MVD_T_D), peg(MVD_T_D)} != 1.
12 on(D,P) :- moved_to(D,P), disk(D), peg(P).
13 moved(D) :- moved_to(D,P), peg(P), disk(D).
14 on(D,P) :- 'on(D,P), not moved(D), disk(D), peg(P).
15 blocked_in(X,P) :- disk(D), 'on(D,P), X = D-1, disk(X), peg(P).
16 blocked_in(X,P) :- disk(D), blocked_in(D,P), X = D-1, disk(X), peg(P).
17 :- disk(D), moved_to(D,P), disk(X), blocked_in(X,P), peg(P), X = D-1.
18 :- moved_to(D,P1), peg(P1), 'on(D,P2), disk(D), blocked_in(D,P2), peg(P2).

19 #program final.
20 :- disk(D), not on(D,P), peg(P), goal(D,P).
```

Logistic problem. The Logistic problem consists of delivering packages to a specific location in a specific city. Packages can be carried by truck or airplanes, but trucks can only move to locations inside the same city, and airplanes, obviously, can only move between airports. The following is the CNL specification of the problem:

```
1 An object is identified by an id.
2 A vehicle is identified by a object.
3 A truck is identified by a object.
4 An airplane is identified by a object.
5 A package is identified by an id.
```

```
6 A location is identified by an id.
7 A city is identified by a location, and by a name.
8 An airport is identified by a location.
9 A goal is identified by a package, and by a location.

10 The following propositions always apply:
11 A truck T is a vehicle.
12 An airplane A is a vehicle.
13 Whenever there is a city with location L, then we must have a location with
       id L.

14 The following propositions always apply except in the initial state:
15 Every vehicle V can load at most 1 package P, when package with id P is
       previously deposited in location L and also vehicle V is previously at
       location L and also package with id P is not previously loaded.
16 Every vehicle V can unload a package P, when a package with id P is
       previously loaded in vehicle V.
17 A package P is loaded in vehicle V, when package P is previously loaded in
       vehicle V and also vehicle V does not unload package P.
18 A package P is loaded, when package P is loaded in a vehicle V.
19 A package P is loaded in vehicle V, when a vehicle V loads package P.
20 It is prohibited that a vehicle V1 loads a package P and also vehicle V2
       loads package P, where V1 is different from V2.
21 A vehicle V has a task when vehicle V load package P.
22 A vehicle V has a task when vehicle V unload package P.
23 Every truck T can move to at most 1 city with location L different from M,
       with name C, when truck T is previously at location M, whenever there is
        a city with location M, with name C.
24 Every airplane A can move to at most 1 airport with location L different
       from M, when airplane A is previously at location M.
25 It is prohibited that a vehicle V moves to a location L whenever there is a
       task with vehicle V.
26 A vehicle V is moving, when vehicle V moves to a location.
27 A package P is deposited in location L, when package P is loaded in vehicle
       V and also vehicle V is at location L.
28 A vehicle V is at location L, when vehicle V moves to location L.
29 A truck T is at location L when truck T is previously at location L and also
        truck T is not moving.
30 An airplane A is at location L when airplane A is previously at location L
       and also airplane A is not moving.
31 A package P is deposited in location L when package P is previously
       deposited in location L and also package P is not loaded.
32 It is prohibited that a vehicle with object id V is moving, whenever there
       is not after a task V.

33 The following propositions apply in the final state:
34 It is prohibited that package P is not deposited in location L, whenever
       there is a goal with package P, and with location L.
35 It is prohibited that package P is loaded, whenever there is a goal with
       package P.
```

After the domain definitions in lines 1-9, the concepts of vehicles and locations are introduced. The

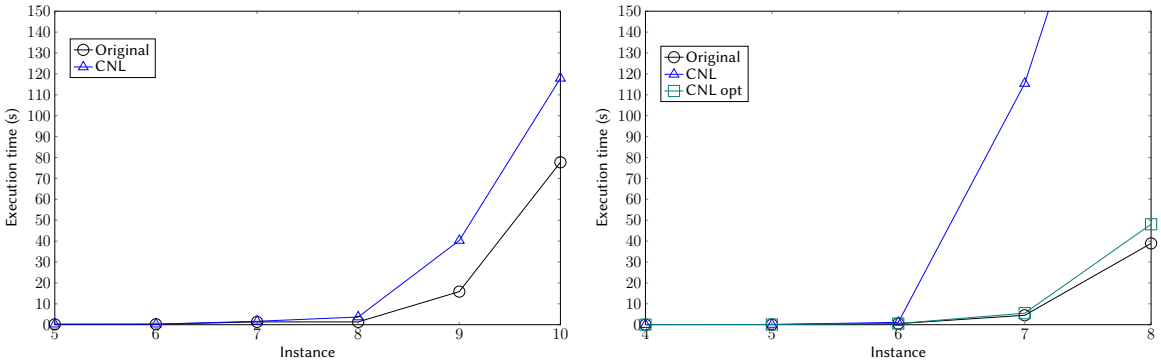
former is a general name for trucks and aircraft while the latter is a part of city. Then, from line 14, the *dynamic* rules are defined. First, it is specified when a vehicle can load and unload a package (lines 15-16) and then how this consequentially updates the state of the package (lines 17-20). Here, the `previously` operator is used to check whether a package can be loaded or unloaded and if its state changes. Lines 21 and 22 define the concept of task, while lines 23 to 31 define the action of moving a vehicle. Again, `previously` operator is used, which ensures that the vehicle moves from its location to a different location. Line 32 ensures that a vehicle moves only to have task, i.e. to load or to unload a package, which is guaranteed by the `after` operator. Finally, the last 3 lines define the objective of the problem, which is having all the packages deposited in the goal location. Below, the corresponding encoding:

```
1  #program always.
2  vehicle(T) :- truck(T).
3  vehicle(A) :- airplane(A).
4  location(L) :- city(L,_).

5  #program dynamic.
6  {load(V,P)} <= 1 :- 'deposited_in(P,L), 'at(V,L), location(L), package(P),
       not 'loaded(P), vehicle(V).
7  {unload(V,P)} :- package(P), 'loaded_in(P,V), vehicle(V).
8  loaded_in(P,V) :- 'loaded_in(P,V), not unload(V,P), package(P), vehicle(V).
9  loaded(P) :- loaded_in(P,V), vehicle(V), package(P).
10 loaded_in(P,V) :- load(V,P), package(P), vehicle(V).
11 :- vehicle(V1), load(V1,P), vehicle(V2), load(V2,P), package(P), V1 != V2.
12 task(V) :- load(V,P), package(P), vehicle(V).
13 task(V) :- unload(V,P), package(P), vehicle(V).
14 {move_to(T,L): city(L,C), L != M} <= 1 :- 'at(T,M), location(M), city(M,C),
       truck(T).
15 {move_to(A,L): airport(L), L != M} <= 1 :- 'at(A,M), location(M),
       airplane(A).
16 :- vehicle(V), move_to(V,L), location(L), task(V).
17 moving(V) :- move_to(V,LCTN_D), location(LCTN_D), vehicle(V).
18 deposited_in(P,L) :- loaded_in(P,V), vehicle(V), at(V,L), package(P),
       location(L).
19 at(V,L) :- move_to(V,L), vehicle(V), location(L).
20 at(T,L) :- 'at(T,L), not moving(T), truck(T), location(L).
21 at(A,L) :- 'at(A,L), not moving(A), airplane(A), location(L).
22 deposited_in(P,L) :- 'deposited_in(P,L), not loaded(P), package(P),
       location(L).
23 :- vehicle(V), moving(V), not &tel {> task(V)}.

24 #program final.
25 :- package(P), not deposited_in(P,L), location(L), goal(P,L).
26 :- package(P), loaded(P), goal(P,_).
```

Performance comparison. We conducted an analysis where we compare the performance of CNL2TEL, which we remind runs TELINGO, with those of TELINGO executed on the encodings contained in the repository. Although, for the Gun problem, the size is fixed with negligible solving times, we performed more detailed experiments by generating larger input instances for the Logistic problem and Tower of Hanoi domains. For the Logistic problem, we considered instances with an increasing number of packages from 5 to 10, while for the Tower of Hanoi problem, we tested instances with 4 up to 8 disks. Results are shown in Figure 1. Overall, as expected, the original human-written encodings



(a) Logistic (b) Hanoi

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

perform better than those automatically generated by CNL2TEL. Nevertheless, concerning Tower of Hanoi, the original encoding has a rule that is not necessary for the correct solution of the problem but it improves its performance. Most likely, a non-expert user would not include it; however, if such a rule is included in the CNL, denoted as CNL opt, the CNL2TEL performance matches the original one.

5. Related Work

In this section, we present an overview of the CNLs proposed in the field of logic programming, with focus on the usage of temporal constructs. For a complete review of CNLs, we refer the reader to the interesting survey in [20].

Although several CNLs have been defined with the aim of specifying logic programs in a natural language, such as Attempto CNL [13] or PENG^{ASP} [21], few of them provide an explicit way to represent temporal concepts and also their grammar is mainly domain-specific. [22] defined a controlled natural language and a tool for converting such language in a temporal logic called Computation Tree Logic, for the specification and verification of hardware designs. As a domain-specific CNL, its grammar is less varied than CNL2TEL and it is mainly built toward signals. [23] proposed a language with support for temporal relations, defined by 15 different templates, for functional testing of control software for passenger vehicles. CNL2TEL, instead, extends CNL2ASP [17] introducing the temporal concepts, thus, its grammar is not domain-specific and tries to cover different application domains. Similar extensions could be applied, e.g., to enable natural language specifications over ASP with temporal constructs following the approach by [24]. PENG Light [25] extends PENG^{ASP} to support the input language of the Simplified Event Calculus [26]. The user can describe events, and define states and conditional statements, then, PENG Light generates a model with the knowledge about the events and in which time point they are effective, and it is able to answer queries. The main difference with CNL2TEL, apart from the grammar, is that PENG Light supports Event Calculus while CNL2TEL is based on TEL.

6. Conclusion

In this paper, we have presented a tool, CNL2TEL, which converts sentences in controlled natural language containing temporal constructs in the language of TELINGO, by extending the specification and reasoning capabilities of CNL2ASP. The tool is applied and evaluated on domains of the TELINGO suite with satisfying results. As future work, we plan to analyze all domains of the TELINGO suite, and to employ CNL2TEL in practical applications.

Acknowledgments

Carmine Dodaro and Marco Maratea were supported by the European Union - NextGenerationEU and by Italian Ministry of Research (MUR) under PNRR project FAIR "Future AI Research", CUP H23C22000860006 and by the European Union - NextGenerationEU and by the Ministry of University and Research (MUR), National Recovery and Resilience Plan (NRRP), Mission 4, Component 2, Investment 1.5, project "RAISE - Robotics and AI for Socio-economic Empowerment" (ECS00000035) under the project "Gestione e Ottimizzazione di Risorse Ospedaliere attraverso Analisi Dati, Logic Programming e Digital Twin (GOLD)", CUP H53C24000400006. Carmine Dodaro was supported by the European Union - NextGenerationEU and by Italian Ministry of Research (MUR) under PNRR project Tech4You "Technologies for climate change adaptation and quality of life improvement", CUP H23C22000370006; and by GNCS-INdAM.

References

[1] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Commun. ACM 54 (2011) 92–103. URL: https://doi.org/10.1145/2043174.2043195. doi:10.1145/2043174.2043195.

[2] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: Proc. of Logic Programming, MIT Press, 1988, pp. 1070–1080.

[3] I. Niemelä, Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm, Ann. Math. Artif. Intell. 25 (1999) 241–273.

[4] E. Erdem, M. Gelfond, N. Leone, Applications of answer set programming, AI Mag. 37 (2016) 53–68. URL: https://doi.org/10.1609/aimag.v37i3.2678. doi:10.1609/AIMAG.V37I3.2678.

[5] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, P. Wanko, Theory solving made easy with clingo 5, in: Proc. of ICLP Technical Communications, volume 52 of *OASIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, pp. 2:1–2:15. URL: https://doi.org/10.4230/OASIcs.ICLP.2016.2. doi:10.4230/OASICS.ICLP.2016.2.

[6] M. Alviano, G. Amendola, C. Dodaro, N. Leone, M. Maratea, F. Ricca, Evaluation of disjunctive programs in WASP, in: M. Balduccini, Y. Lierler, S. Woltran (Eds.), Proc. of LPNMR, volume 11481 of *LNCS*, Springer, 2019, pp. 241–255.

[7] V. S. Mellarkod, M. Gelfond, Y. Zhang, Integrating answer set programming and constraint logic programming, Ann. Math. Artif. Intell. 53 (2008) 251–287. URL: https://doi.org/10.1007/s10472-009-9116-y. doi:10.1007/S10472-009-9116-Y.

[8] D. Shen, Y. Lierler, Smt-based constraint answer set solver EZSMT+ for non-tight programs, in: M. Thielscher, F. Toni, F. Wolter (Eds.), Proc. of KR, AAAI Press, 2018, pp. 67–71.

[9] A. Armando, C. Castellini, E. Giunchiglia, M. Idini, M. Maratea, TSAT++: an open platform for satisfiability modulo theories, Electronic Notes in Theoretical Computer Science 125 (2005) 25–36.

[10] P. Bruno, F. Calimeri, C. Marte, M. Manna, Combining deep learning and asp-based models for the semantic segmentation of medical images, in: S. Moschoyiannis, R. Peñaloza, J. Vanthienen, A. Soylu, D. Roman (Eds.), Proc. of RuleML+RR, volume 12851 of *LNCS*, Springer, 2021, pp. 95–110.

[11] P. Bruno, F. Calimeri, C. Marte, Dedudeep: An extensible framework for combining deep learning and asp-based models, in: G. Gottlob, D. Inclezan, M. Maratea (Eds.), Proc. of LPNMR, volume 13416 of *LNCS*, Springer, 2022, pp. 505–510.

[12] F. Aguado, P. Cabalar, M. Diéguez, G. Pérez, T. Schaub, A. Schuhmann, C. Vidal, Linear-time temporal answer set programming, TPLP 23 (2023) 2–56. URL: https://doi.org/10.1017/S1471068421000557. doi:10.1017/S1471068421000557.

[13] N. E. Fuchs, Knowledge representation and reasoning in (controlled) natural language, in: Proc. of ICCS, volume 3596 of *LNCS*, Springer, 2005, pp. 51–51. URL: https://doi.org/10.1007/11524564_3. doi:10.1007/11524564_3.

[14] P. Clark, P. Harrison, T. Jenkins, J. A. Thompson, R. H. Wojcik, Acquiring and using world

knowledge using a restricted subset of english, in: Proc. of FLAIRS, AAAI Press, 2005, pp. 506–511. URL: http://www.aaai.org/Library/FLAIRS/2005/flairs05-083.php.

[15] C. Baral, J. Dzifcak, Solving puzzles described in english by automated translation to answer set programming and learning how to do that translation, in: Proc. of KR, AAAI Press, 2012. URL: http://www.aaai.org/ocs/index.php/KR/KR12/paper/view/4562.

[16] E. Erdem, R. Yeniterzi, Transforming controlled natural language biomedical queries into answer set programs, in: Proc. of BioNLP@HLT-NAACL, ACL, 2009, pp. 117–124. URL: https://aclanthology.org/W09-1315/.

[17] S. Caruso, C. Dodaro, M. Maratea, M. Mochi, F. Riccio, CNL2ASP: converting controlled natural language sentences into ASP, TPLP 24 (2024) 196–226. URL: https://doi.org/10.1017/s1471068423000388. doi:10.1017/S1471068423000388.

[18] P. Cabalar, R. Kaminski, P. Morkisch, T. Schaub, telingo = ASP + time, in: Proc. of LPNMR, volume 11481 of *LNCS*, Springer, 2019, pp. 256–269. URL: https://doi.org/10.1007/978-3-030-20528-7_19. doi:10.1007/978-3-030-20528-7_19.

[19] P. Cabalar, R. Kaminski, T. Schaub, A. Schuhmann, Temporal answer set programming on finite traces, TPLP 18 (2018) 406–420. URL: https://doi.org/10.1017/S1471068418000297. doi:10.1017/S1471068418000297.

[20] T. Kuhn, A survey and classification of controlled natural languages, Comput. Linguistics 40 (2014) 121–170. URL: https://doi.org/10.1162/COLI_a_00168. doi:10.1162/COLI_A_00168.

[21] R. Schwitter, Specifying and verbalising answer set programs in controlled natural language, TPLP 18 (2018) 691–705. URL: https://doi.org/10.1017/S1471068418000327. doi:10.1017/S1471068418000327.

[22] A. Holt, E. Klein, A semantically-derived subset of english for hardware verification, in: Proc. of ACL, ACL, 1999, pp. 451–456. URL: https://aclanthology.org/P99-1058/. doi:10.3115/1034678.1034747.

[23] M. Esser, P. Struss, Obtaining models for test generation from natural-language-like functional specifications, in: Proc. of DX, 2007, pp. 75–82.

[24] M. Borroto, I. Kareem, F. Ricca, Towards automatic composition of ASP programs from natural language specifications, Accepted at IJCAI https://doi.org/10.48550/arXiv.2403.04541 (2024). URL: https://doi.org/10.48550/arXiv.2403.04541. doi:10.48550/ARXIV.2403.04541. arXiv:2403.04541.

[25] R. Schwitter, Working for two: A bidirectional grammar for a controlled natural language, in: Proc. of AJCAI, volume 5360 of *LNCS*, Springer, 2008, pp. 168–179. URL: https://doi.org/10.1007/978-3-540-89378-3_17. doi:10.1007/978-3-540-89378-3_17.

[26] R. Schwitter, Specifying events and their effects in controlled natural language, Procedia - Social and Behavioral Sciences 27 (2011) 12–21. URL: https://www.sciencedirect.com/science/article/pii/S1877042811024050. doi:https://doi.org/10.1016/j.sbspro.2011.10.578, computational Linguistics and Related Fields.