UNIVERSITY OF GENOVA

PHD PROGRAM ON SECURITY, RISK AND VULNERABILITY - CURRICULUM CYBERSECURITY AND RELIABLE AI

Logic-Based Approaches to Scheduling and Their Enhancement through Controlled Natural Languages

by

Simone Caruso

Thesis submitted for the degree of *Doctor of Philosophy* (38° cycle)

January 2026

Prof. Marco Maratea	Supervisor
Prof. Serena Cattari	Head of the PhD program
Prof. Luca Oneto	Head of the PhD curriculum

Thesis Jury:

Prof. Mauro Vallati, *University of Huddersfield*	External examiner
Prof. Jose Sousa, *Sano - Centre for Computational Personalised Medicine*	External examiner

Dibris

Department of Informatics, Bioengineering, Robotics and Systems Engineering

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university.

Simone Caruso
January 2026

Abstract

In the healthcare domain, the effective management of healthcare operations, such as patient scheduling, staff rostering, and resource allocation, remains a persistent challenge due to complex constraints and dynamic environments. Artificial Intelligence offers promising methods for optimizing these processes, particularly through Knowledge Representation and Reasoning (KRR) approaches that ensure explainable and logically grounded solutions. Among these, Answer Set Programming (ASP), also thanks to the presence of efficient solvers, has proven to be particularly suitable for modeling complex healthcare scheduling problems. Thus, in this thesis, we present solutions to two healthcare scheduling problems: the Pre-Operative Assessment Clinic scheduling problem, which involves assigning patients to days for examination and surgical preparation, and the scheduling of Periodic Treatments problem, where patients must follow predetermined treatment plans over a period of several weeks. The use of formal KRR languages like ASP, however, often requires specialized expertise, limiting accessibility for domain experts. To bridge this gap, Controlled Natural Languages (CNLs) have emerged as a means to express formal logic in a more intuitive and human-readable format. CNLs can enhance readability, reduce development time, and facilitate communication between technical and non-technical stakeholders. In this thesis, we propose a set of tools in this regard. First, we introduce CNL2ASP, which is able to convert sentences written in CNL into an ASP encoding. Then, the second tool is CNL2TEL. It extends CNL2ASP with temporal operators, and is able to convert CNL specifications into the TELINGO input language. Similarly, SBVR2ASP is a tool that translates specifications written in the Semantics of Business Vocabulary and Business Rules (SBVR) into ASP. A recent survey has highlighted several limitations in current approaches to SBVR conflict detection and analysis, and in our approach based on ASP, we address most of these limitations. Finally, the last tool we propose is CNLWizard, a framework that generates grammars for target representation languages, enabling the translation of problems stated in CNL into formal representations. This tool offers a flexible, high-level approach to defining desired grammars, significantly reducing the time and effort needed to create custom grammars.

Table of contents

List of figures

List of tables

Part I

Introduction

Chapter 1

Introduction

1.1 Context and Motivation

The continuous increase in demand for healthcare services, driven by factors such as an aging population and the rise in chronic diseases, has necessitated the adoption of new efficient solutions to improve the delivery and efficiency of the care provided. This includes a wide range of tools and innovations, including electronic health records, telemedicine, mobile health applications, wearable devices, health information systems, and data analytics platforms, with the overall goal of improving patient outcomes, supporting clinical decision-making, and enabling more efficient management of healthcare resources [Marynissen and Demeulemeester, 2019].

This is necessary because hospitals and clinics operate under significant resource constraints, including limited numbers of medical staff, equipment, and treatment facilities, while facing increasing demand for timely and high-quality care. Therefore, a poor scheduling or suboptimal allocation of resources can result in long patient waiting times, staff overload, underutilization of facilities, and overall inefficiencies that negatively impact both patient outcomes and organizational performance [Park et al., 2020, Yu et al., 2025, Abdalkareem et al., 2021].

In particular, these challenges are even more relevant in contexts involving the coordination of several activities such as scheduling pre-operative exams or scheduling periodic treatments where appointments are arranged in advance for an extended period. Optimizing patient scheduling in the face of such resource constraints is critical. For example, surgical patients often require multiple exams across different specialties to ensure they are adequately prepared for surgery. Efficiently organizing these tests can eliminate the need for patients to be admitted one or two days prior to surgery, allowing them to remain at home until

the day of the procedure. In addition, better scheduling can reduce waiting times between exams, increasing patient satisfaction [Harnett et al., 2010] and reducing the risk of surgery cancellations [Ferschl et al., 2005].

In this regard, Artificial Intelligence (AI) has emerged as a promising solution, offering advanced optimization and decision support techniques. In recent years, AI-based approaches have demonstrated that they can model and solve healthcare scheduling problems, resulting in better resource utilization, shorter waiting times, and improved overall service quality.

A promising approach to address these complex scheduling challenges lies in the use of Knowledge Representation and Reasoning (KRR) languages. While data-driven Machine Learning approaches are invaluable for predictive tasks, such as image analysis, estimating procedure durations, or risk of disease, they often function as "black-boxes" lacking explicit logical justifications for their outputs. In contrast, KRR approaches are inherently explainable, as they rely on explicit logical rules to perform reasoning. This is because many KRR systems use tree-based search mechanisms that allow reconstruction of the reasoning paths leading to specific decisions. The combination of explainability and structured reasoning makes KRR-based approaches particularly suitable for critical domains such as healthcare, where trust and transparency in automated decision-making are essential.

Such symbolic approaches have already proven effective; for instance, planners (such as PDDL) are used to solve personalized medication problems [Alaboud and Coles, 2019, Alon et al., 2024]. Among KRR formalisms, Answer Set Programming (ASP) has been recently applied in scheduling problems in the Healthcare domain is Answer Set Programming (ASP) [Dodaro et al., 2018, 2019a,b, Alviano et al., 2020, Cardellini et al., 2021, Cappanera et al., 2023a,b, Caruso et al., 2023] . ASP is a declarative programming rooted in logic programming and non-monotonic reasoning. ASP solvers (e.g., [Gebser et al., 2012, Alviano et al., 2019a, Dodaro et al., 2024b]) generate solutions, called answer sets, that satisfy a given logic program. It is particularly well-suited for problems requiring reasoning with incomplete or evolving information, such as knowledge representation, reasoning, and AI applications [Erdem et al., 2016]. Recently, it has been also applied in neuro-symbolic approaches, in combination to neural networks [Riley and Sridharan, 2019]. A notable example is [Bruno et al., 2025], where the authors proposed a robust solution to the *Operating Room Scheduling problem* [Dodaro et al., 2018] by using ML techniques to estimate the surgery durations and ASP for the scheduling part.

Nevertheless, despite the success of KRR formalisms, their usage is not that trivial for people without a background in this field and 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 KRR formalism emerged [Fuchs, 2005, Clark et al., 2005]. 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 KRR 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, e.g., 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.

A notable example is the Semantics of Business Vocabulary and Business Rules (SBVR) standard [The Object Management Group, 2019], established by the Object Management Group (OMG). SBVR provides a structured yet human-readable format for expressing business rules and policies, allowing domain experts to formally define constraints without programming expertise. This approach promotes consistency, traceability, and verifiability across systems [Ceci et al., 2016, Abi-Lahoud et al., 2013].

More recently, Large Language Models (LLMs) [Kaplan et al., 2020, Min et al., 2024], such as ChatGPT [OpenAI, 2023], have shown to be a promising approach for formalizing and explaining KRR programs in a more accessible manner to users. Indeed, their applicability covers several areas of use. By translating natural language specification into complex logical constructs, they can significantly make formalisms accessible also to non-experts. They can also be integrated into debugging and explainability systems, offering further support to users. In domains with strict compliance and regulatory requirements, they can simplify the verification process, ensuring that KRR systems conform to the necessary rules and guidelines by translating their logic into a format that is easier to review or they can be integrated. However, a notable limitation of LLMs is their tendency to *hallucinate*, or generate information that is inaccurate or not present in the input data. This characteristic poses a challenge for reliability, especially in applications where precision and factual accuracy are crucial.

For this reason, CNLs continue to be valuable: their formally defined grammars enable a deterministic and error-free translation into the desired KRR target formalism, ensuring the reliability required in critical domains such as healthcare.

Another advantage of using CNLs is that they can also serve as intermediate representations that bridge informal descriptions of problems written in natural language and formal logical encodings. This has been the subject of study by more recent approaches such as [Borroto Santana et al., 2024], which aim to generate specifications from unconstrained natural language through an intermediate CNL representation.

However, implementing custom CNLs is still a time-consuming and challenging process. Indeed, from a technical perspective, creating a custom CNL involves writing the grammar, processing the Abstract Syntax Tree (AST), and generating code that translates the natural language input into the specific formalism. The first step requires defining a set of syntactic rules capable of capturing the nuances of the language while maintaining precision and clarity. The grammar must be comprehensive enough to cover the wide variety of constructs that users may wish to express, yet restricted enough to avoid ambiguity. Once the CNL input is parsed according to the grammar, it must be converted into an AST. This stage is complex because it involves resolving ambiguities, managing scope, and ensuring compliance with all semantic rules of the CNL. The final step involves translating the AST into code that conforms to the specific formalism. Thus, the construction of the CNL and the translation are an intricate process that could be mitigated by an higher-level framework.

1.2 State of the Art

Scheduling problems in Healthcare. Traditional approaches to addressing complex planning problems in the healthcare domain have been based primarily on mathematical programming (MP) and metaheuristic optimization methods. Among these, mixed-integer programming (MIP) has been widely used to model and solve various healthcare planning tasks, such as scheduling patient appointments and optimizing the use of hospital resources [Heshmat and Eltawil, 2021, Huggins et al., 2014, Pérez et al., 2011, Zhang and Xie, 2015]. While MIP-based formulations provide a rigorous mathematical framework capable of producing optimal or near-optimal solutions under well-defined constraints, other approaches use metaheuristic algorithms such as Simulated Annealing [Edward et al., 2008b] and Genetic Algorithms [Xiao et al., 2018], that do not guarantee that a globally optimal solution is found but they employ stochastic and population-based search mechanisms that enable exploration of large and complex solution spaces within reasonable computational time.

Other research efforts have focused on Stochastic Dynamic Programming (SDP) [Akhavizadegan et al., 2017, Landa et al., 2016, Zhang et al., 2017], which provides a principled framework for sequential decision-making under uncertainty. SDP is particularly relevant

for healthcare scenarios involving stochastic patient arrivals, uncertain treatment durations, or variable resource availability. Nevertheless, the method's computational complexity increases exponentially with the number of state variables, a phenomenon often referred to as the *curse of dimensionality*, which significantly restricts its scalability to large-scale or high-dimensional problems. Similarly, Constraint Programming (CP) approaches [Alade and Amusat, 2019, Gaspero and Urli, 2014, Öztürkoğlu, 2020] have been applied to healthcare scheduling problems by emphasizing constraint satisfaction.

In contrast to these traditional optimization and constraint-based paradigms, KRR systems offer an alternative for modeling and solving scheduling problems in the healthcare doamin. KRR frameworks combine logical reasoning with declarative modeling, enabling the explicit representation of knowledge. One of the main advantages of KRR approaches lies in the availability of efficient open-source solvers, which often have similar performances to the industrial tools, and their capability of multi-level reasoning, making them suitable for scheduling problems in healthcare where different elements (e.g., patients' preferences, resources, operators) are optimized with different criteria and priorities.

Controlled Natural Languages. A Controlled Natural Language is a deliberately restricted subset of a full natural language whose syntax, vocabulary and/or semantics are constrained to reduce ambiguity and complexity. As highlighted by [Kuhn, 2014] and [Schwitter, 2002], CNLs can be broadly categorized into three main types, according to their primary purpose:

1. to improve human-to-human communication, particularly among speakers with diverse linguistic backgrounds;

2. to enhance the quality and efficiency of translation processes, whether manual, computer-aided, semi-automatic, or automatic; and

3. to serve as an accessible, natural representation layer for formal notations and KRR systems.

Furthermore, over the past few decades, CNLs have evolved from communication aids specific to certain fields to a mature interdisciplinary research area that intersects computational linguistics, formal logic, and artificial intelligence. Early examples, such as Simplified Technical English (STE) [1], were designed to standardize and simplify technical documentation, thereby minimizing misinterpretations and translation errors. On the other hand, Attempto

[1] https://www.asd-ste100.org/

Controlled English [Schwitter et al., 1995] is an example of a formal, logic-oriented CNL that maps syntactic structures directly onto first-order logical representations.

Today, CNLs remain a valuable technology from a practical standpoint and well-founded from a theoretical standpoint, particularly in safety-critical sectors such as aviation, healthcare, and knowledge engineering, where low ambiguity and formal interpretability are essential.

Additionally, the advent of Large Language Models (LLMs), such as GPT and related architectures, has profoundly reshaped the field of natural language understanding and generation. These models demonstrate good performance in generating executable code directly from natural language descriptions. However, they remain inherently probabilistic and therefore unreliable when producing formally precise code, e.g., that used in KRR languages such as ASP. Their stochastic nature and absence of explicit formal grounding often result in syntactic errors or semantic inconsistencies. In contrast, CNLs ensure grammatical determinism and semantic rigor, making them particularly well-suited for applications where correctness and consistency are essential. Recent research explores strategies for bridging LLMs and KRR formalisms but they are still limited to simple and specific tasks. For example, [Alviano and Grillo, 2024] employs LLMs to generate ASP facts from textual descriptions, while [Coppolillo et al., 2024] focuses on pattern-based conversion of ASP code snippets, though their approach struggles with more complex encodings. Similarly, [Wang et al., 2024] and [Yang et al., 2024] investigate the inverse direction, using KRR structures to enhance LLM reasoning capabilities, but their methods currently face challenges in generalizing to multi-step reasoning and maintaining logical consistency.

In summary, while LLMs have dramatically expanded the expressive potential of natural language interfaces, CNLs continue to offer indispensable advantages in formal precision, reproducibility, and interpretability. The convergence of these two paradigms, through hybrid architectures with a CNL-guided generation, or using CNLs as an intermediate layer to map NL to ASP (e.g. [Borroto Santana et al., 2024]), represents a promising direction for advancing trustworthy, explainable, and verifiable language understanding systems.

1.3 Contributions

The contributions of this thesis regard two different but interconnected areas: AI solutions to scheduling problems in the healthcare domain and the improvement of the usage of AI techniques by the CNL integration. The contributions presented in the thesis are listed below:

- A solution to the Pre-Operative Assessment Clinic scheduling problem. This problem is divided into two sub-problems [Edward et al., 2008a]. In the first sub-problem,

patients are assigned to a day taking into account a default list of exams, and the solution has to schedule patients before their due date and prioritize the assignments to patients with higher priority. In the second sub-problem, the scheduler assigns a starting time to each exam needed by the patients, considering the available operators and the duration of the exams. In this thesis, we first present a mathematical formulation of both sub-problems: in the solution of the first sub-problem, the scheduler minimizes the number of unassigned patients, while in the second sub-problem, using as input the result of the first sub-problem, the time each patient stays at the hospital is minimized. We then apply ASP to model and solve the PAC scheduling problem by presenting two ASP encodings for both sub-problems. We also propose a solution to the rescheduling problem. A solution to this problem is necessary when the scheduling solution can not be applied fully, due to, e.g., unavailable patients, or an operator who is suddenly missing. We, then, run an experimental analysis on synthetic PAC benchmarks with realistic sizes and parameters inspired by data seen in the literature, varying the number of scheduled patients and the available operators. Overall, results using the state-of-the-art ASP solver CLINGO [Gebser et al., 2012] show that ASP is a suitable solving methodology also for the PAC scheduling and rescheduling problems. We finally present the implementation and show the usage of a web application that allows to insert the main parameters of the problem, run our scheduling solution and analyze the results graphically without the need to install any ASP software locally.

- A solution to the scheduling of Periodic Treatments problem. This problem consists of planning a care path over a period of several weeks, in which patients have to perform different treatments respecting a certain periodicity. Treatments must be assigned to a day taking into account patients' preferences, operators' availability, the possibility of needing instruments/machinery, and of having different facilities in which operators can work and move around during the working time. The solution to this problem was developed with a certain degree of flexibility and to be suitable for different contexts where periodicity is required, e.g., rehabilitation or planning a cycle of drug therapies. As our contribution, we propose an ASP encoding to compute the scheduling of periodic treatments over a fixed period of time. We then present the results of a preliminary experimental analysis, demonstrating that the ASP system CLINGO [Gebser et al., 2016], when executed on the presented encoding, achieves good performance in solving randomly-generated instances.

- We developed CNL2ASP, a tool that automatically translates English sentences written in a CNL into corresponding ASP encodings. CNL2ASP supports the main features of ASP, including disjunctive rules, choice rules, aggregates, and weak constraints. We demonstrate its functionality through several use cases, both on well-known synthetic domains and on real-world problems described in the literature. For the latter, we also conducted an experimental evaluation comparing the performance of the automatically generated encodings with those written by human experts. In addition, we carried out a preliminary user validation to assess the usability and readability of the proposed CNL.

- 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 [Cabalar et al., 2019], 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. 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.

- We integrated ASP in the Semantics of Business Vocabulary and Rules (SBVR) language to be able to do integrity verification. SBVR provides a formal framework for representing business rules in a structured and human-readable manner and they have an important role in aligning business logic with enterprise requirements. However, a recent survey has highlighted several limitations in current approaches to SBVR conflict detection and analysis. We propose a novel approach based on ASP that addresses most of such limitations as it offers a simple and declarative way of representing SBVR, a set of robust tools for conflict explanation, and high-performance solvers.

- We propose a novel framework that facilitates the creation of custom CNLs. Specifically, this framework (i) enables the user to specify a grammar for multiple target languages abstractly, (ii) automatically suggests a possible default implementation for some elements involved in the pipeline, and (iii) provides auxiliary data structures that make the construction of the CNL language more flexible and guided. We implemented

the framework, obtaining a system called CNLWizard. CNLWizard processes inputs described in a simple YAML-based language and automatically generates the grammar for the CNL, along with a set of pre-implemented imperative functions that minimize boilerplate code. This allows developers to focus primarily on writing the specific code required to convert sentences into the target formalism, streamlining the process and making custom CNL development more accessible and efficient.

1.4 Outline

After introducing in this chapter the problems addressed and our contribution in this thesis, the following chapter presents the syntax, semantics, and methodology of the ASP language, which serves as the core tool used throughout this work. The remainder of the thesis is then divided into three parts.

Scheduling problems in healthcare. In the second part of this thesis, we introduce a solution to the Pre-Operative Assessment Clinic scheduling problem and a solution to the scheduling of Periodic Treatments problem, respectively in Chapter 3 and Chapter 4.

Integrating Controlled Natural Languages. In the third part of this thesis, we present a set of tools for supporting CNLs in KRR formalisms. In particular, the CNL2ASP tool is presented in Chapter 5, followed by CNL2TEL, i.e., the temporal extension of CNL2ASP, in Chapter 6. In Chapter 7, we present SBVR2ASP, a formal framework for representing business rules in a structured and human-readable manner and, finally, in the last chapter (Chapter 8) we introduce CNLWizard, a general framework for facilitating the development of custom CNLs targeting arbitrary formalisms.

Conclusions. The last part of the thesis is devoted to the related works and conclusions, in Chapter 9 and Chapter 10, respectively.

Chapter 2

Background on Answer Set Programming

Answer Set Programming (ASP) [Brewka et al., 2011] is a declarative programming paradigm within the field of knowledge representation and reasoning, designed for solving complex combinatorial problems. It enables users to specify a problem through rules and constraints, from which ASP systems automatically compute all possible solutions that satisfy these specifications.

The combination of ASP's clear syntax, intuitive semantics, and the availability of efficient solvers [Gebser et al., 2018a] makes it a powerful approach for tackling difficult reasoning tasks. Over the years, the ASP community has developed a rich ecosystem of tools and methodologies for explanation and debugging [Fandinno and Schulz, 2019], which are also widely used in real applications.

Moreover, modern ASP solvers, as CLINGO [Gebser et al., 2016] and WASP [Alviano et al., 2019a], support optimized grounding and solving techniques making them perfectly suitable for large-scale reasoning tasks, and more recent solvers, e.g., [Dodaro et al., 2024b] work via compilation.

Thanks to these developments, ASP has gained increasing popularity in both academic [Erdem et al., 2020, Erdem and Patoglu, 2018, Palù et al., 2018, Gavanelli et al., 2015] and industrial settings [Grasso et al., 2011]. In particular, ASP has been successfully employed to solve various real-world scheduling problems in the healthcare context, including the Nurse Scheduling Problem [Dodaro and Maratea, 2017, Alviano et al., 2017b], the Operating Room Scheduling Problem [Dodaro et al., 2018, 2019a, 2022], the Chemotherapy Treatment Scheduling Problem [Dodaro et al., 2021] (see [Alviano et al., 2020] for a recent survey). Moreover, modern ASP solvers, as CLINGO [Gebser et al., 2016] and WASP [Alviano et al., 2019a], support optimized grounding and solving techniques making them perfectly suitable

for large-scale reasoning tasks, and more recent solvers, e.g., [Dodaro et al., 2024b] work via compilation.

In the remainder of this section, we overview the language of ASP. More detailed descriptions and a more formal account of ASP, including the features of the language employed in this thesis, can be found in [Brewka et al., 2011, Calimeri et al., 2013]. Throughout this discussion, we assume the reader is familiar with logic programming conventions.

2.1 Syntax

The syntax of ASP is similar to the one of Prolog. Variables are strings starting with an uppercase letter, and constants are non-negative 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 $\{Terms_1 : Conj_1;\cdots;Terms_t : Conj_t\}$, where $t > 0$, and for all $i \in [1,t]$, each $Terms_i$ is a list of terms such that $|Terms_i| = k > 0$, and each $Conj_i$ is a 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, e.g., the function *#count* computes the number of terms.

An *aggregate atom* is of the form $f(S) \prec T$, where $f(S)$ is an aggregate function, $\prec \in \{<,\leq,>,\geq,\neq,=\}$ is a 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:

$$a_1 \mid \ldots \mid a_n \text{ :- } b_1,\ldots,b_k, not\ b_{k+1},\ldots, not\ b_m.$$

where $a_1,\ldots,a_n$ are standard atoms, $b_1,\ldots,b_k$ are atoms, $b_{k+1},\ldots,b_m$ are standard atoms, and $n,k,m \geq 0$. A literal is either a standard atom a or its negation $not\ \ a$. The disjunction $a_1|\ldots|a_n$ is the *head* of r, while the conjunction $b_1,\ldots,b_k, not\ b_{k+1}, \ldots, not\ b_m$ 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* [Buccafurri et al., 2000] ω is of the form:

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

where w and l are the weight and level of ω, respectively. (Intuitively, $[w@l]$ is read as "weight w at level l", where the weight is the "cost" of violating the condition in the body of 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.

2.2 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 literal ℓ (resp., *not* ℓ) is true w.r.t. I if $\ell \in I$ (resp., $\ell \notin I$), and false (resp., true) 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 the rules of a program. Given a ground program G_P and an interpretation I, the *reduct* [Faber et al., 2011] 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. 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) [Faber et al., 2011].

Given a program with weak constraints $\Pi = \langle P, W \rangle$, the semantics of Π extends from the basic case defined above. Thus, let $G_\Pi = \langle G_P, G_W \rangle$ be the instantiation of Π; a constraint $\omega \in G_W$ is violated by an interpretation I if all the literals in ω are true w.r.t. I. An *optimum*

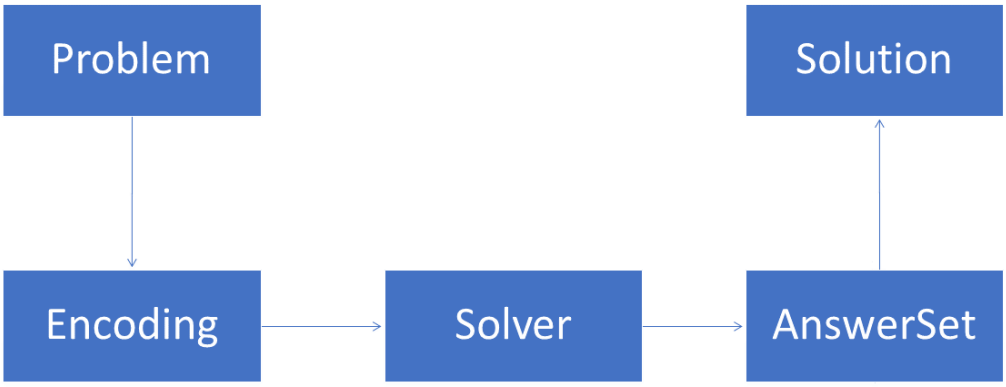


Figure 2.1 Resolution methodology schema

answer set for Π is an answer set of G_P that minimizes the sum of the weights of the violated weak constraints in G_W in a prioritized way.

Syntactic shortcuts. In the following, we also use *choice rules* of the form $\{p\}$, where p is an atom. Choice rules can be viewed as a syntactic shortcut for the rule $p \mid p'$, where p' is a fresh new atom not appearing elsewhere in the program, meaning that the atom p can be chosen as true.

2.3 Solving methodology

In figure 2.1, a representation of a solution based on a declarative programming approach as our ASP solution is reported. It is made up of five steps:

- Problem: this block represents the problem formulation to be modeled and solved.
- Encoding: this block is the formal representation of the problem, using ASP. It is made up of a set of rules as presented above.
- Solver: the solver takes in input the encoding of the problem and generates the answer set.

- AnswerSet: this block corresponds to the output of the solver, i.e. to the set of atoms that satisfy all the rules of the encoding.
- Solution: this block represents the final solution of the problem.

More in detail, ASP enables the representation of problems in a declarative manner. The expressive strength of disjunctive rules allows it to capture problems that go beyond NP complexity. Moreover, by optionally distinguishing between a fixed, non-ground program and an input database, ASP facilitates uniform solutions across different problem instances.

Problems of relatively high computational complexity can be naturally addressed using the *Guess&Check* programming approach, proposed in [Leone et al., 2006]. This method operates as follows: a database of facts defines a specific instance of the problem, while a collection of (often disjunctive) rules, the guessing part, describes the search space. A further, optional set of rules, the checking part, then constrains this space by enforcing admissibility conditions.

To illustrate the concept, consider the well-known NP-complete problem 3-COLORING. Given an undirected graph $G = (V, E)$, the task is to assign one of three colors (red, green, or blue) to each vertex so that no two adjacent vertices share the same color. This problem can be modeled in ASP as follows:

```
1 % Fact database defining the instance
2 vertex(v). % for each v in V
3 edge(i,j). % for each (i, j) in E

4 % Uniform non-ground program solving the problem
5 col(X, red) | col(X, green) | col(X, blue) :- vertex(X). % guessing
    part
6 :- edge(X, Y), col(X, C), col(Y, C). % checking part
```

The first two rules declare the facts that describe the input graph G. The next rule specifies that each vertex must be assigned exactly one color, while the final rule acts as an integrity constraint, ruling out any case where two connected vertices share the same color. More details can be found in [Leone and Ricca, 2015].

Part II

Scheduling problems in Digital Health

Chapter 3

The Pre-Operative Assessment Clinic scheduling problem

The Pre-Operative Assessment Clinic (PAC) scheduling problem is the task of assigning patients to a day, in which the patient will be examined and prepared to a surgical operation, taking into account, e.g., patients with different priority levels, due dates, and operators availability. The PAC consists of several exams needed by patients to ensure they are well prepared for their operation. This allows patients to stay at home until the morning of the surgery, instead of being admitted to the hospital one or two days before the scheduled operation; moreover, this allows to reduce waiting time between the exams, thus increasing patients' satisfaction [Harnett et al., 2010], and to avoid the cancellation of the surgery [Ferschl et al., 2005]. A proper solution to the PAC scheduling problem is vital to improve the degree of patients' satisfaction and to reduce surgical complications.

In our solution, the problem, whose specifications have been provided by a potential SurgiQ[1] client, is divided into two sub-problems [Edward et al., 2008a]. In the first sub-problem, patients are assigned to a day taking into account a default list of exams, and the solution has to schedule patients before their due date and prioritize the assignments to patients with higher priority. In the second sub-problem, the scheduler assigns a starting time to each exam needed by the patients, considering the available operators and the duration of the exams. Our two-phase solution is not guaranteed to find the best possible overall solution, but has been designed in this way because: (i) it simplifies the overall encoding and its practical use, and (ii) it mimics how PAC schedules are often computed (manually) in hospitals.

[1] https://surgiq.com/.

In this chapter, we first present a mathematical formulation of both sub-problems: In the solution of the first sub-problem, the scheduler minimizes the number of unassigned patients, while in the second sub-problem, using as input the result of the first sub-problem, the time each patient stays at the hospital is minimized. We then apply ASP to model and solve the PAC scheduling problem, by presenting two ASP encodings for both sub-problems. We also propose a rescheduling solution that may come into play when the scheduling solution can not be applied fully, due to, e.g., unavailable patients, or an operator who is suddenly missing. We run an experimental analysis on synthetic PAC benchmarks with realistic sizes and parameters inspired by data seen in the literature, varying the number of scheduled patients and the available operators. Overall, results using the state-of-the-art ASP solver CLINGO [Gebser et al., 2012] show that ASP is a suitable solving methodology also for the PAC scheduling and rescheduling problems. We finally present the implementation and show the usage of a web application that allows to insert the main parameters of the problem, run our scheduling solution and analyze the results graphically without the need to install any ASP software locally.

3.1 Problem Description

Pre-hospitalization is the phase in which the patient enters the facility to perform laboratory tests, radiological and cardiological examination, counseling for anesthesiological suitability for surgery, and any other examinations that the medical specialists deem appropriate for the definition of the pre-intervention diagnostic pathway (pneumological examination, nephrological examination, etc). The management of the pre-intervention pathway is crucial within healthcare organizations as it involves the alignment of multiple facilities to ensure that services are delivered on time and in the correct manner. However, the high number of services involved in this pathway can create critical issues in the patient's journey in terms of service delivery time and alignment between the various providers (Cardiology, Pulmonology, Nephrology, etc).

In many hospitals, including the one from which the specifications we employ have been taken, the date of the pre-operative procedures is set at a very early stage, before the exact number of necessary exams and visits is known. Later on, after a first contact with the patient, the number and nature of the exams and visits are settled, and can be scheduled precisely, assigning to each of them a starting time during the day set in the first phase. Often, this problem is still solved as a manual process, having critical issues such as the scheduling of appointments and the alignment between different providers. The automatic scheduling aims

to ensure completion of the pathway in one day, to avoid repeated patient accesses and to allow the patients to perform all services on time.

In the first of the two phases outlined above, we consider that each patient requires a default list of exams according to his specialty, i.e., the lists of exams are equal for patients requiring the same specialty but differ among specialties, and the scheduler assigns the day of PAC without considering the starting time of the exams, but just assigning a temporary starting time of the first and the last exam required by each patient to ensure that such patients can be assigned in the same day. Thus, in the first sub-problem, the solution assigns patients overestimating the duration and the number of exams needed. In particular, all the optional exams, such as exams required by smokers or patients with diabetes, are assigned to all the patients in the first phase. The overestimation is important in order to guarantee the solvability of the second sub-problem. Then, when the operation day is closer, the hospital knows exactly the exams needed by each patient and can assign the starting time of each exam. Going into more details, the first sub-problem consists of scheduling appointments in a range of days for patients requiring a surgical operation. Each patient is linked to a due date, a target day, and a priority level: The due date is the maximum day on which (s)he can be assigned, the target day is the optimal day on which to schedule the appointment, while the solution prioritizes patients with a higher priority level. There are several exam areas, corresponding to the locations in which patients will be examined. Each exam area needs operators to be activated and has a limited time of usage. Each operator can activate three different exam areas, but they can be assigned to just one exam area for each day. The solution must assign the operators to the exam areas, to activate them, and the day of PAC to patients, ensuring that the total time of usage of each exam location is lower than its limit. Since in this first sub-problem the list of exams needed by patients is not the final list, i.e., just the first and the last exam are the same for every patient, and in the second sub-problem some exams could be added, the solution schedules patients leaving some unused time to each exam area. An optimal solution minimizes the number of unassigned patients, giving priority to patients with higher priority levels, and ties are broken by minimizing the difference between the day assigned and the target day of each patient, giving again precedence to patients with higher priority.

In the second sub-problem, patients are linked to their real exams, so the solution has to assign the starting time of each exam, having the first sub-problem already assigned the day. The input consists of registrations, exams needed by patients and the exam areas activated. Exams are ordered, so the solution must assign the starting time of each exam respecting their order and their duration, by considering that each exam area can be used by one patient

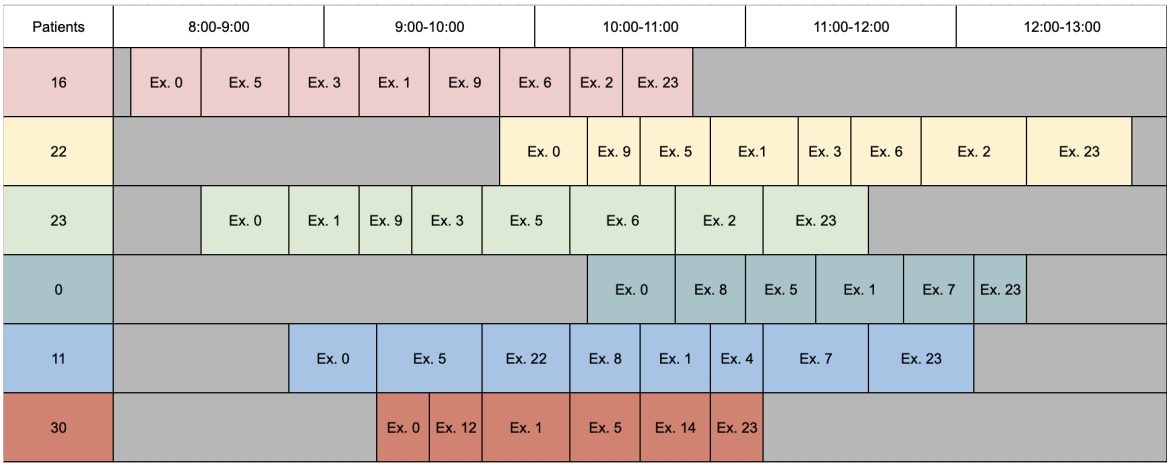


Figure 3.1 Schedule example: assignments of the starting time of the different exam locations of each patient in a single day.

at a time. Finally, the solution minimizes the difference between the starting time of the first exam and the last exam of each patient.

Figure 3.1 shows an example of the schedule that we want to compute, by assigning the starting times (x-axis) to each patient (y-axis) on a particular day. The patients that must be scheduled are the same that are set by the first sub-problem on this day. As can be seen in Figure 3.1, in this case, the scheduler is able to assign all patients optimally; indeed, all patients have no waiting time among the exams and thus the time spent in the hospital is reduced to the minimum, while respecting the constraints of the problem.

3.2 Formalization of the PAC Scheduling problem

In this section, we provide a mathematical formulation of the two sub-problems.

Definition 1 (first PAC sub-problem)**.** *Let*

- *R be a finite set of registrations;*
- *$E = \{e_1, \ldots, e_m\}$ be a set of m exams;*
- *$EL = \{el_1, \ldots, el_n\}$ be a set of n exam locations;*
- *O be a finite set of operators;*
- *$D = \{d : d \in [1..14]\}$ be the set of all days;*
- *$T = \{t : t \in [1..ts]\}$ be the set of all time slots;*

- $\delta : R \mapsto \{1,2,3,4\}$ *be a function associating a registration to a priority;*
- $\rho : R \times E \mapsto \mathbb{N}$ *be a function associating a registration and an exam to a duration such that for a registration r and an exam e if* $\rho(r,e) > 0$ *then the registration r requires the exam e;*
- $\omega : R \mapsto \mathbb{D}$ *be a function associating a registration to a due date;*
- $\lambda : R \mapsto \mathbb{D}$ *be a function associating a registration to a target day;*
- $\sigma : E \mapsto \mathbb{EL}$ *be a function associating an exam to the exam location;*
- $\mu : EL \times D \mapsto \mathbb{N}$ *be a function associating an exam location and a day to the maximum number of registrations that can be assigned concurrently, such that* $\mu(el,d) = n$ *if at most n registrations can be assigned concurrently to the exam location el in the day d.*
- $\tau : EL \times D \mapsto \mathbb{N}$ *be a function associating an exam location and a day to the required number of operators to be activated, such that* $\tau(el,d) = n$ *if the exam location el in the day d requires n operators to be activated;*
- $\theta : O \times EL \times D \mapsto \{0,1\}$ *be a function such that* $\theta(o,el,d) = 1$ *if the operator o is assigned to the exam location el in the day d, and* 0 *otherwise;*
- *ts be a constant that is equal to the number of time slots.*

Let $x : R \times D \times T \mapsto \{0,1\}$ *be a function such that* $x(r,d,t) = 1$ *if the registration r is assigned to the day d and the time slot t, and* 0 *otherwise. Moreover, for a given x, let* $A_x = \{(r,d,t) : r \in R, d \in D, t \in T, x(r,d,t) = 1\}$.

Then, given sets R, E, EL, O, D, T, and functions δ, ρ, ω, λ, σ, μ, τ, θ, *the first PAC sub-problem is defined as the problem of finding a schedule x such that*

(c_1) $|\{(d,t) : (r,d,t) \in A_x\}| \leq 1 \quad \forall r \in R, d \leq \omega(r)$;

(c_2) $|\{(d,t) : (r,d,t) \in A_x\}| = 0 \quad \forall r \in R, d > \omega(r)$;

(c_3) $t \leq ts - \sum_{e \in E} \rho(r,e) \quad \forall (r,d,t) \in A_x$;

(c_4) $|\{r : (r,d,t') \in A_x, t \geq t', t < t' + \rho(r,e_1)\}| \leq \mu(el) \quad \forall d \in D, t \in T, el = \sigma(e_1)$;

(c_5) $|\{r : (r,d,t') \in A_x, t \geq t' + \sum_{e \in E} \rho(r,e) - \rho(r,e_m), t < t' + \sum_{e \in E} \rho(r,e)\}| \leq \mu(el) \quad \forall d \in D, t \in T, el = \sigma(e_m)$;

(c_6) $|\{o : \theta(o,el,d) = 1\}| = \tau(el,d) \quad \forall (r,d,t) \in Ax, \rho(r,e) > 0, el = \sigma(e);$

(c_7) $|\{el : \theta(o,el,d)\}| \leq 1 \quad \forall o \in O, \forall d \in D.$

Condition (c_1) ensures that each registration is assigned at most one time in the days before the due date associated with the registration. Condition (c_2) ensures that each registration is not assigned after the due date associated with the registration. Condition (c_3) ensures that for each registration the sum of the duration of the exams required plus the time slot assigned to the registration is less than the constant ts. Condition (c_4) ensures that the number of registrations having the first exam at a time slot t is less than or equal to the allowed value for each time slot. Condition (c_5) ensures that the number of registrations having the last exam at a time slot t is less than or equal to the allowed value for each time slot. Condition (c_6) ensures that for each exam location used by at least one registration, the required number of operators is assigned. Condition (c_7) ensures that each operator is assigned to at most one exam location each day.

Definition 2 (Unassigned registrations)**.** *Given a solution x, let* $U_x^{pr} = \{r : r \in R, \delta(r) = pr, r \notin A_x\}$. *Intuitively,* U_x^{pr} *represents the set of registrations of priority pr that were not assigned to any day.*

Definition 3 (Distance target day)**.** *Given a solution x, let* $t_x^{pr} = \sum_{x(r,d,t) \in A_x, \delta(r) = pr} | d - \lambda(r) |$. *Intuitively,* t_x^{pr} *represents the sum of the distance between the day assigned to the registrations of priority pr and the target day associated.*

Definition 4. *A solution x is said to dominate a solution x' if* $|U_x^{pr}| < |U_{x'}^{pr}|$ *for the biggest pr for which* $|U_x^{pr}| \neq |U_{x'}^{pr}|$ *or if* $|U_x^{pr}| = |U_{x'}^{pr}|$ *for all the pr and* $|t_x^{pr}| < |t_{x'}^{pr}|$ *for the biggest pr for which* $|t_x^{pr}| \neq |t_{x'}^{pr}|$. *A solution is optimal if it is not dominated by any other solution.*

Definition 5 (second PAC sub-problem)**.** *Let*

- $\beta : R \times E \mapsto \mathbb{N}$ *be a function associating a registration and an exam to a value corresponding to the order in which the exam must be assigned, such that* $\beta(r,e) = n$ *and* $\beta(r,e') = n'$ *and* $n > n'$ *if the registration r must do the exam e after the exam e';*
- $\gamma : EL \mapsto \mathbb{N}$ *be a function associating an exam location to the starting time of the exam location, such that* $\gamma(el) = n$ *if n is the first time slot in which a registration requesting the exam location el can be assigned;*

- $\xi : EL \mapsto \mathbb{N}$ *be a function associating an exam location to the ending time of the exam location, such that* $\gamma(el) = n$ *if n is the last time slot in which a registration requesting the exam location el can be assigned.*

Let $x : R \times E \times EL \times D \times T \mapsto \{0,1\}$ *be a function such that* $x(r,e,el,d,t) = 1$ *if the registration r and the exam e are assigned to the exam location el in the day d and in the time slot t, and* 0 *otherwise. Moreover, for a given x let* $A_x = \{(r,e,el,d,t) : r \in R, e \in E, el \in EL, d \in D, t \in T, x(r,e,el,d,t) = 1\}$.

Then, given sets R, E, EL, D, T, and functions ρ, β, γ, ξ, μ, *the second PAC sub-problem is defined as the problem of finding a schedule x, such that*

(c_8) $|\{(d,t) : (r,e,el,d,t) \in A_x\}|=1 \quad \forall e \in E, \forall r \in R, \rho(r,e) > 0, \sigma(e) = el;$

(c_9) $\gamma(el) \leq t \leq \xi(el) - \beta(r,e) \quad \forall (r,e,el,d,t) \in A_x;$

(c_{10}) $(r,e,el,d,t) \notin A_x \quad \forall (r,e',el',d,t') \in A_x, \forall e \in E, \rho(r,e') = dt, \forall t \in T, t' \leq t < t' + dt;$

(c_{11}) $t > t' \quad \forall (r,e,el,d,t) \in A_x, (r,e',el,d,t') \in A_x, \beta(r,e) > \beta(r,e');$

(c_{12}) $|\{r : (r,e,el,d,t') \in A_x, \sigma(e) = el, t \geq t', t < t + \rho(r,e)\}| \leq \mu(el) \quad \forall el \in EL, t \in T.$

Condition (c_8) ensures that each exam is assigned exactly once. Condition (c_9) ensures that each exam is assigned after the starting time of the required exam location and before the closing time of the required exam location minus the duration of the exam. Condition (c_{10}) ensures that for each registration, each exam is assigned after that the exam before is ended. Condition (c_{11}) ensures that each exam is assigned after the exams lower in the ordering are assigned. Condition (c_{12}) ensures that the number of exams assigned to a location is lower than the maximum availability for each location in any time slot.

Definition 6 (Time in hospital)**.** *Given a solution x, let*

$$m_x = \sum_{(r,e_1,el,d,t) \in A_x, x(r,e_m,el,d,t') \in A_x} (t' + \rho(r,e_m) - t - \sum_{e \in E} \rho(r,e)).$$

Intuitively, m_x *represents the waiting time between the exams. It is evaluated as the sum of the differences between the time spent in the hospital (evaluated as the difference between the ending time of the last exam and the starting time of the first exam) and the sum of the durations of the exams required by each registration.*

Definition 7. *A solution x is said to dominate a solution x' if* $m_x < m_{x'}$. *A solution is optimal if it is not dominated by any other solution.*

3.3 ASP Encoding

In this section, starting from the specifications in Section 3.1 and the formalization of Section 3.2, we present the ASP encoding for the two sub-problems, in two separate sub-sections, based on the input language of CLINGO [Gebser et al., 2016].

3.3.1 ASP encoding for the first PAC sub-problem

In the following, we present the ASP encoding for the first sub-problem.

Data Model. The input data is specified by means of the following atoms:

- Instances of `reg(RID,PR,TARGET,TOTDUR,DUEDATE)` represent the registrations, characterized by an id (`RID`), the priority level (`PR`), the ideal day in which the registration should be assigned (`TARGET`), the sum of the durations of the exams needed by the registration (`TOTDUR`), and the due date (`DUEDATE`).
- Instances of `exam(RID,AREAID,DUR)` represent the exam linked to the registration identified by an id (`RID`), the exam area (`AREAID`), and the duration (`DUR`).
- Instances of `examLoc(AREAID,NOP,DAY,N)` represent the exam areas, characterized by an id (`AREAID`), which requires `NOP` operators to be activated in a day (`DAY`), and can be concurrently assigned up to `N` registrations.
- Instances of `operators(ID,AREAID,DAY)` represent the operators, characterized by an id (`ID`), that can be assigned to the exam area (`AREAID`) in a day `(DAY)`.
- Instances of `day(DAY)` represent the available days.
- The constant ts represents the number of time slots considered.
- The constants $first\_exam$ and $last\_exam$ correspond to e_1 and e_m in Section 3.2 (i.e., the first and the last exam required by every registration, respectively).
- The constants $totRegsP1$, $totRegsP2$, $totRegsP3$, and $totRegsP4$ represent the number of registrations with priority 1, 2, 3, and 4, respectively.

The output is an assignment represented by an atom of the form `x(RID,PR,ST,ET,DAY)`, where the intuitive meaning is that the exams of registration with id `RID` and priority level `PR`

```
1  {x(RID,PR,ST,ST+TOTDUR,DAY) : day(DAY),time(ST), DAY < DUEDATE, ST <=
       ts-TOTDUR} 1 :- reg(RID,PR,TARGET,TOTDUR,DUEDATE).
2  :- x(RID,_,_,_,DAY), exam(RID,AREAID,_), not
       examLoc(AREAID,_,_,DAY,_).
3  :- #count{RID: x(RID,_,ST,_,DAY),exam(RID,first_exam,DUR), T >= ST, T
       < ST+DUR} > N, examLoc(first_exam,_,_,DAY,N), day(DAY),time(S).
4  :- #count{RID: x(RID,_,_,ET,DAY),exam(RID,last_exam,DUR), T < ET, T
       >= ET-DUR} > N, examLoc(last_exam,_,_,DAY,N), day(DAY),time(T).
5  {operator(ID, AREAID, DAY) : operators(ID, AREAID, DAY)} == NOP :-
       examLoc(AREAID, NOP, _, DAY,_), x(RID, _, _, _,DAY), exam(RID,
       AREAID, _).
6  :- operator(ID,AREAID1,DAY), operator(ID,AREAID2,DAY), AREAID1 <
       AREAID2.
7  unassignedP1(N) :- M = #count {RID: x(RID,1,_,_,_)}, N = totRegsP1 -
       M.
8  unassignedP2(N) :- M = #count {RID: x(RID,2,_,_,_)}, N = totRegsP2 -
       M.
9  unassignedP3(N) :- M = #count {RID: x(RID,3,_,_,_)}, N = totRegsP3 -
       M.
10 unassignedP4(N) :- M = #count {RID: x(RID,4,_,_,_)}, N = totRegsP4 -
       M.
11 :~ unassignedP1(N). [N@8]
12 :~ unassignedP2(N). [N@7]
13 :~ unassignedP3(N). [N@6]
14 :~ unassignedP4(N). [N@5]
15 :~ x(RID,1,_,_,DAY), reg(RID,_,TARGET,_,_). [|DAY-TARGET|@4,RID]
16 :~ x(RID,2,_,_,DAY), reg(RID,_,TARGET,_,_). [|DAY-TARGET|@3,RID]
17 :~ x(RID,3,_,_,DAY), reg(RID,_,TARGET,_,_). [|DAY-TARGET|@2,RID]
18 :~ x(RID,4,_,_,DAY), reg(RID,_,TARGET,_,_). [|DAY-TARGET|@1,RID]
```

Figure 3.2 ASP encoding of the first sub-problem

are assigned to the day `DAY`, the temporary starting time of the first exam is `ST` and the temporary ending time of the last exam is `ET`; and an atom of the form `operator(ID,AREAID,DAY)`, where the intuitive meaning is that the operator with id `ID` is assigned to the exam area `AREAID` on the day `DAY`.

Encoding. The related encoding is shown in Figure 3.2, and is described in the following. To simplify the description, we denote as r_i the rule appearing at line i of Figure 3.2.

Rule r_1 assigns registrations to a day and to a time slot. The assignment is made by assigning a day that is before the due date and to a temporary time slot such that the sum

of the time slot and the duration of the exams required is less than the constant ts. Rule r_2 checks that every registration is assigned to a day with all the exam areas needed to be activated. Rule r_3 is used to ensure that the number of registrations having the first exam is less or equal to the allowed value in every time slot. Rule r_4 is used to ensure that the number of registrations having the last exam is less or equal to the allowed value in every time slot. Rule r_5 assigns operators to the required exam areas, while rule r_6 checks that each operator is assigned to just one exam area on every day. Rules from r_7 to r_{10} derive intermediate atoms *unassignedP*1, *unassignedP*2, *unassignedP*3, and *unassignedP*4, respectively, that are used to count how many registrations with different priorities are not assigned to a day. Weak constraints from r_{11} to r_{14} use the atoms derived by rules from r_7 to r_{10} to minimize the number of unassigned registrations according to their priority. Finally, weak constraints from r_{15} and r_{18} minimize the difference between the assigned and target day of each registration, giving precedence to higher priorities.

3.3.2 ASP encoding for the second PAC sub-problem

We now move to the second sub-problem.

Data Model. The input data is the same of the first sub-problem for the atoms `exam` and `time`, while other atoms are changed:

- Instances of `reg(RID,DAY)` represent the registrations, characterized by an id (`RID`) assigned to a day (`DAY`).
- Instances of `examLoc(AREAID,DAY,AREAST,AREAET,N)` represent the exam areas, characterized by an id (`AREAID`), which in a day (`DAY`) has a starting time (`AREAST`) and a closing time (`AREAET`), and can provide the exam to `N` registrations.
- Instances of `phase(AREAID,ORD)` represent the order (`ORD`) of the exams provided by the exam area characterized by an id (`AREAID`).

The output is represented by an atom of the form `x(RID,AREAID,ST,ET,DAY)`, where the intuitive meaning is that the exam of the registration with id `RID` is in exam area `AREAID`, starts at time `ST` and ends at time `ET` on the day `DAY`.

Encoding. The encoding consists of the rules reported in Figure 3.3. Rule r_1 assigns a starting and an ending time to each exam needed by every registration, checking that the time assigned is within the opening time of the required exam area. Rule r_2 ensures that the order

```
1 {x(RID,AREAID,ST,ST+DUR,DAY) : examLoc(AREAID,DAY,AREAST,AREAET,_),
    time(ST), ST >= AREAST, ST <= AREAET-DUR} = 1 :- reg(RID,DAY),
    exam(RID,AREAID,DUR).
2 :- x(RID,AREAID1,ST1,_,_), x(RID,AREAID2,ST2,_,_),
    phase(AREAID1,ORD1), phase(AREAID2,ORD2), ORD2 < ORD1, ST1 < ST2.
3 :- #count{AREAID: x(RID,AREAID,ST,ET,DAY), T >= ST, T < ET} > 1,
    reg(RID,DAY), time(T).
4 :- #count{AREAID: x(RID,AREAID,ST,ET,DAY), T >= ST, T < ET} > N,
    examLoc(AREAID,DAY,_,_,N), time(T).
5 :~ reg(RID,_), x(RID,first_exam,ST,_,_), x(RID,last_exam,_,ET,_).
    [ET-ST@1, RID]
```

Figure 3.3 ASP encoding of the second sub-problem

among the exams is respected. Rules r_3 checks that each registration is assigned to at most one exam for every time slot. Then, rule r_4 checks that each exam area provides the exam to at most `N` registrations for every time slot. Finally, rule r_5 minimizes the difference between the ending time of the last exam and the starting time of the first exam of each registration.

```
6 forbiddenAfter(RID,AREAID1,ST+DUR1) :- reg(RID,_,_),
    exam(RID,AREAID1,DUR1), phase(AREAID1,ORD1), #sum{DUR2:
    exam(RID,AREAID2,DUR2), phase(AREAID2,ORD2), ORD2 > ORD1 } = ST.
7 forbiddenBefore(RID,AREAID1,ST) :- reg(RID,_,_),
    exam(RID,AREAID1,DUR1), phase(AREAID1,ORD1), #sum{DUR2:
    exam(RID,AREAID2,DUR2), phase(AREAID2,ORD2), ORD2 < ORD1 } = ST.
8 {x(RID,AREAID,ST,ST+DUR,DAY): examLoc(AREAID,DAY,AREAST,AREAET,_),
    forbiddenAfter(RID,AREAID,FORB1),
    forbiddenBefore(RID,AREAID,FORB2), time(ST), ST >= AREAST, ST <=
    AREAET-DUR, ST <= ts-FORB1, ST > FORB2 } = 1 :- reg(RID,PRI,DAY),
    exam(RID,AREAID,DUR).
```

Figure 3.4 Optimized encoding for pruning exams' starting time.

Domain specific optimizations. The encoding of the second sub-problem above is general, able to find optimal solutions, but the solver needs a large amount of time to prove optimality. However, some domain specific optimizations may be added to improve its performance. We present two domain specific optimizations, presented in the following two sub-paragraphs, which rely on the knowledge of the PAC domain for pruning impossible solutions.

```
9  cost(RID, TOT) :- TOT = #sum{DUR,AREAID : exam(RID,AREAID,DUR)},
       reg(RID,_,_).
10 :~ x(RID,first_exam,ST,_,_),x(RID,last_exam,_,ET,_), cost(RID,TOT),
       ET-ST-TOT >= 0. [ ET-ST-TOT@1, RID]
```

Figure 3.5 Optimized minimization rule

Pruning of exams' starting time slots. As shown in Figure 3.3, in r_1 the starting time of the exams is guessed between all the available daily time slots, expressed by the atom `time`. Given that the exams must be assigned following an order, the minimum number of time slots that each patient needs to stay before and after each exam is known. Thus, the guess rule can be improved by reducing the number of possible starting time slots of each exam with the following constraints:

- an exam cannot start in a time slot if the remaining time slots are less than the minimum number of time slots required to complete all the following exams;
- an exam cannot start in a time slot if the time slots before are less than the minimum number of time slots required to complete the previous exams.

The encoding for pruning exams' starting time slots is obtained by substituting r_1 from Figure 3.3 with the rules reported in Figure 3.4, explained in the following. In rules r_6 and r_7 two new atoms `forbiddenAfter` and `forbiddenBefore` are defined as the minimum amount of time slots needed by each patient after (resp. before) each exam obtained by computing the sum of the duration of the exams with the greater (resp. lower) phase value. In r_8 the two new atoms are used in the guess rule, so that the starting time is after the value computed by rule r_6, and after the difference between `lastTimeSlot`, that corresponds to the last time slot, and the value computed by r_7.

Minimization with lower bound. As can be seen also in Figure 3.3, rule r_5 minimizes the time spent in the hospital by each patient, computed as the difference between the ending time of the last exam and the starting time of the first exam. However, the time spent in the hospital by each patient cannot be lower than the sum of the duration of all the required exams. Therefore, the minimization rule can be improved by computing the minimum time required by each patient and using it as a lower bound, so that candidate solutions below this value are pruned.

The enconding with the optimized weak constraint is obtained by substituting rule r_5 from Figure 3.3 with the rules shown in Figure 3.5. In rule r_9, the time to fully complete

all the exams for each patient is computed as the sum of the duration of all the exams as the new auxiliary atom `cost`. The weak constraint in rule r_{10} then minimizes the difference between the planned total time (i.e., the difference between the ending time and the starting time of the last and first exam) and the lower bound previously computed, activating the weak constraint only when such a difference is greater or equal to zero.

3.4 Experimental Results

In this section, we report the results of an empirical analysis of the PAC scheduling problem via ASP. For the first sub-problem (second subsection), data have been randomly generated using parameters inspired by literature and real world data (first subsection), then the results of the first sub-problem have been used as input for the second sub-problem (third subsection). The last subsection is then devoted to a comparison to alternative logic-based formalisms. The experiments were run on an AMD Ryzen 5 3600 CPU @ 3.60GHz with 16 GB of physical RAM. The ASP system used was CLINGO [Gebser et al., 2016] 5.4.1, using parameters *--restart-on-model* for faster optimization and *--parallel-mode 6* for parallel execution.

3.4.1 PAC benchmarks

Data are based on the sizes and parameters of a typical mid-sized hospital, with 24 different exam areas. The solution schedules patients in a range of 14 days, 5 hours per day, and for each day there are 60 time slots, thus the constant ts is set to 60, corresponding to 5 minutes per time slot. To test scalability, we generated 3 different benchmarks of different dimensions, having 40, 60, and 80 patients, respectively. Each benchmark was tested 10 times with different randomly generated input.

In particular, each registration is linked to a surgical specialty, and needs a number of exams between 5 and 13, according to the specialty, while the duration of each exam varies between 3 and 6 time slots. The priorities of the registrations have been generated from an even distribution of four possible values (with weights of 0.25 for registrations having priority 1, 2, 3, and 4, respectively). Moreover, a due date is randomly generated for each registration. For all the benchmarks, as said, there are 24 exam areas and the operators, that are 35, can be assigned to 3 different exam areas. So, by increasing the number of patients while maintaining fixed the number of operators, we tested different scenarios with low, medium and high requests.

For the second sub-problem, we used the results of the first sub-problem as input. Thus, the number of patients and the exam locations activated depend on the assignments made by the first sub-problem. Patients require all the same first and last exam, while the other exams required by each patient are linked to an order that is randomly assigned and that must be respected by the scheduler. In the second sub-problem, clinics know the actual list of exams needed by patients: To simulate this scenario, we randomly selected the optional exams assigned to patients in the first sub-problem. The number of such exams depends on the specialty, and on the "status" of the patient: For example, optional exams are needed by patients who are over 65 years old or smokers.

3.4.2 Results for the first sub-problem

The first optimization criterion in the PAC scheduling sub-problem is to assign as many patients as possible, starting from patients with higher priority. On all the tested instances, our solution is able to assign a day to 406 out of 437 patients with the highest priority; moreover, in 23 out of 30 instances, the solution assigns a day to all or all but one patients with the highest priority.

Table 3.1 summarizes the results obtained in this first sub-problem, with a timeout set to 300 seconds for each instance, and a preliminary analysis on CLINGO done also with other parameters, e.g., `-opt-strategy=usc` that allows CLINGO to employ a different optimization algorithm: In particular, the table shows the average percentage of patients from the 10 instances assigned with 40, 60, and 80 patients according to their priority level. The test with 80 patients shows in particular how the solution behaves in a scenario with high demand and low resource availability, since the number of exam areas and operators is fixed. From the table, we can observe that the percentage of patients with priority 1 and 2 assigned is just slightly lower than in the other benchmarks, while the percentage of patients assigned with lower priorities decreases, due to the few resources available compared to the demand.

Compared to the previous version of the encoding [Caruso et al., 2021] used for the first sub-problem, this new version is able to assign more patients without leading to unsatisfiable problems in the second sub-problem. What is changed is that, while in the old version of the encoding we used a formula to overestimate the time required by each patient, in this new version we assign a temporary starting time for the exams in the first sub-problem. The starting time is used to check that the first and the last exam (which are mandatory for every patient) are not concurrently required by more than the allowed number of patients. This change in the encoding leads to more patients assigned in the first sub-problem and,

Table 3.1 Percentage of assigned patients according to their priority level.

Total #Patients	%P1 ASSIGNED	%P2 ASSIGNED	%P3 ASSIGNED	%P4 ASSIGNED
40	92%	92%	91%	83%
60	95%	89%	82%	65%
80	91%	87%	60%	44%

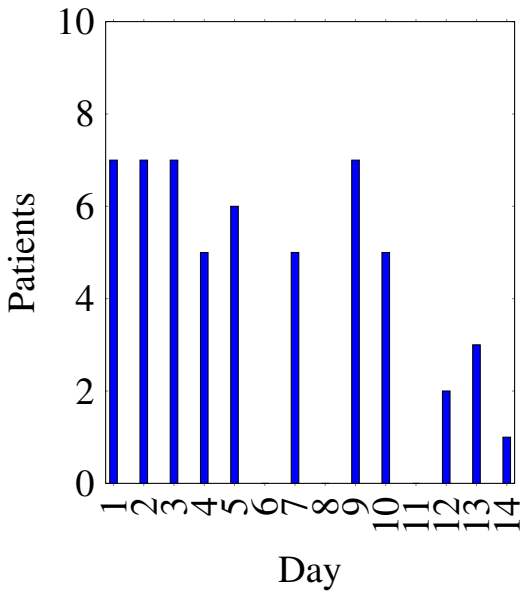


Figure 3.6 Number of patients assigned to each day by the scheduler with 60 patients as input.

in particular, while the previous version assigned 80% of patients with the highest priority, with the new version we are able to assign approx. 93% of patients with the highest priority. Moreover, testing the previous encoding using the instances with 80 patients, the old encoding was able to assign a date just to 45% of the patients with priority 1 and 2, while the new version assigns a date to 89% of patients.

The second optimization criterion is to have an assigned day that is as close as possible to the target day. This optimization criteria is able to assign some patients close to their target day, while for other patients, quality decreases. This is due to two reasons: The first one is that there is another optimization criteria with higher priority; thus, the scheduler already tries to assign as many patients as possible, without taking into account the target days of the patients, and the second one is that some patients have a target day in a day without their exam locations available, so, even in an optimal solution the assigned day to some patients would not be in the target day.

To corroborate the general explanation above, Figure 3.6 reports the result obtained by the scheduler with one of the instances with 60 patients as input. What can be seen from the graph is that some patients are assigned in the last days, while on some days before, there are no patients assigned. This can be explained by the fact that the scheduler tries to assign as many patients as possible and does not try to assign as soon as possible. Moreover, on some days, patients can not be assigned due to the unavailability of the exam locations. Other instances behave similarly.

3.4.3 Results for the second sub-problem

This subsection reports the results of the second sub-problem, with the same experimental setting of the first. In the second sub-problem, the solution assigns the starting time of each exam of the patients. The input is taken from the results obtained in the first sub-problem. The solution minimizes the difference between the ending time of the last exam and the starting time of the first exam. While minimizing this value, the solution tries to minimize the time spent in the hospital by all patients. In this sub-problem, the scheduler is able to reach an optimal solution in all the instances tested. Moreover, in every instance tested, there is at most one patient who has to wait for one time slot between two exams, while all the other patients have no waiting time between the exams. In particular, in all but three instances, patients are assigned without a waiting time among the exams.

The timings required by the scheduler in the second sub-problem are shown in Figure 3.7, which represents the range of seconds required to reach the optimal solution in all the instances tested with the different number of patients, identified by the minimum and maximum times for solving the instances in the set, together with the mean and the median time. From the figure, it can be seen that it takes 16 seconds on average to find the optimal solution, considering instances with 40 patients. While considering instances with 60 and 80 patients, the scheduler finds the optimal solution on average in 33 seconds and 44 seconds, respectively. Moreover, even in the worst case, with 80 patients, the scheduler is able to find the optimal solution in less than 70 seconds.

To obtain these results, we used the encoding defined in Section 3.3.2, including the domain specific optimizations. We are now interested in understanding the contribution that the two optimizations bring. Table 3.2 presents the mean time required to reach an optimal solution with the different versions of the encoding. In particular, the second column of the table contains results for the encoding without optimizations, defined as BASIC, the third and fourth columns contain results with the two optimizations, defined as OPT1 and

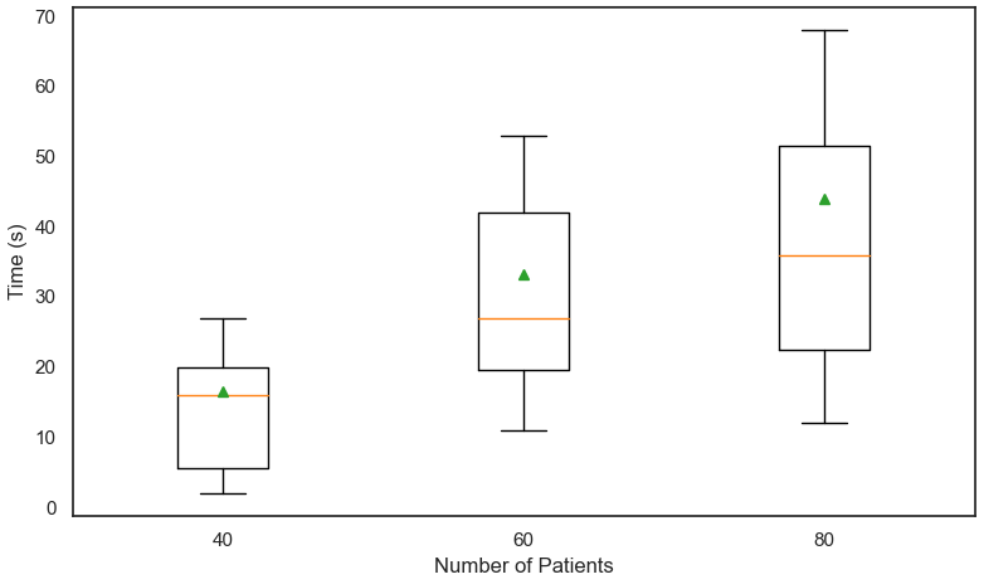


Figure 3.7 Results obtained by solving 10 instances generated from the results of the first sub-problem considering 40, 60, and 80 patients. The box starts from the first quartile and ends at the third quartile. The mean time is represented by the (green) triangle, while the (orange) line represents the median value.

OPT2, respectively, while the fourth column, denoted ENC, reports the results with both optimizations. From Table 3.2, it can be noted that, while the plain encoder is able to reach the optimal solution on 16,7% of the instances (last row), the encoders featuring the OPT2 optimization are able to reach the optimal solution on all instances.

Moreover, the performance increase due to the OPT1 optimization is not evident, at least with the current timeout, despite the advantages obtained in terms of the number of rules generated, as shown in Figure 3.8. The figure reports the number of rules generated by the PLAIN and by PLAIN+OPT1 options using the instances with 40 patients, and shows that the OPT1 optimization allows to decrease the number of rules generated by approx. one third. With PLAIN+OPT2, the performance increases noticeably, leading to the optimal solution in a few seconds. Finally, adding both OPT1 and OPT2 led to better results, being able to reach an optimal solution in all the instances in less time than the other encodings. Moreover, in it can be noted that, paired with OPT2 optimization, OPT1 optimization helps further increasing the performance of the solutions.

Table 3.2 Comparison of the mean time required to reach the optimal solution with the different versions of the encoding for the second sub-problem.

Total #Patients	CPU TIME PLAIN (S)	CPU TIME PLAIN+OPT1 (S)	CPU TIME PLAIN+OPT2 (S)	CPU TIME ENC (S)
40	260.7	268.1	30.4	16.6
60	283.5	295.5	57.1	33.3
80	295.3	300.0	61.9	44.1
Percentage Optimal	16,7%	16,7%	100%	100%

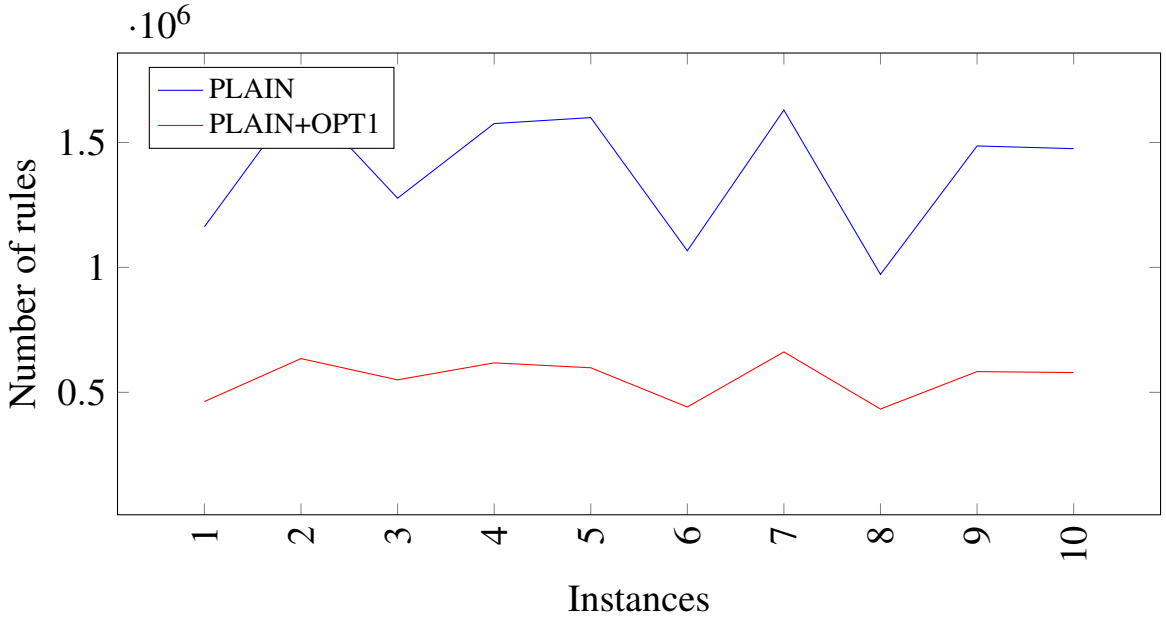


Figure 3.8 Comparison between the number of rules generated using the plain encoding and using the plain encoding with the OPT1 optimization.

3.4.4 Comparison to alternative logic-based formalisms

In the following, we present an empirical comparison of our ASP-based solution with alternative logic-based approaches, obtained by applying automatic translations of ASP instances. In more detail, we used the ASP solver WASP [Alviano et al., 2019b], with the option `-pre=wbo`, which converts ground ASP instances into pseudo-Boolean instances in the wbo format [Olivier Roussel and Vasco Manquinho, 2012]. Then, we used the tool PYPBLIB [Ansótegui et al., 2019] to encode wbo instances as MaxSAT instances. Moreover, in order to provide a fair comparison, given that the other formalisms/solvers can not deal with multi-level optimizations, we also processed our ASP instances using WASP with the option `-pre=lparse`, which collapses all weak constraint levels into one single level using exponential weights. In this way, the costs found by the different approaches can be compared.

Table 3.3 Comparison of the ASP solution to the first sub-problem with alternative logic-based solutions.

Instance	TIME (S) CLINGO-ROM	TIME (S) CLINGO-USC	TIME (S) MAXHS	TIME (S) OPEN-WBO	TIME (S) RC2	TIME (S) GUROBI
1	9.1	1.4	15.6	1.3	1.4	10.5
2	0.1	0.1	26.2	0.1	0.3	1.1
3	TIME	30.1	TIME	30.5	27.7	39.7
4	1.4	0.4	TIME	0.2	0.8	2.2
5	9.2	1.7	41.6	1.0	–	26.5
6	TIME	1.4	28.5	0.6	1.9	11.7

We started from the instances with 40 patients for this analysis: Out of the 10 instances, we were able to effectively use 6 instances for the comparison, since on 4 instances WASP was not able to produce the corresponding single-level instance.

Then, we considered three state-of-the-art MaxSAT solvers, namely MAXHS [Saikko et al., 2016], OPEN-WBO [Martins et al., 2014], and RC2 [Ignatiev et al., 2019], and the industrial tool for solving optimization problems GUROBI [Gurobi Optimization, LLC, 2021], which is able to process instances in the wbo format. We also re-run CLINGO on the generated single-level instances, and used other than the option `restart-on-model` (CLINGO-ROM), the option `-opt-strategy=usc` (CLINGO-USC). The latter enables the usage of algorithm OLL [Morgado et al., 2014], which is the same algorithm employed by the MaxSAT solver RC2, and differently from previous experiments, is not dominated by CLINGO-ROM. The timeout was set to 60 seconds, given that the grounding phase and instance adaptation have been done offline.

Results are shown in Table 3.3, where for each solver and instance we report the time in seconds to reach the optimal solution and we write "TIME" if it reaches the time limit with a solution, or a dash in case the solver outputs no solution within the time limit. As a general observation, CLINGO-USC, OPEN-WBO, and GUROBI are the only solvers that are able to find an optimal solution in all the tested instances. Moreover, while GUROBI finds the optimal solution on average in approx. 15 seconds, OPEN-WBO and CLING-USC obtain the best performance overall, since they are able to find the best solution on average in approx. 5 seconds. Concerning CLINGO-ROM and MAXHS, they are able to find an optimal solution in 4 instances. However, CLINGO-ROM is faster to find the optimal solutions on average. Finally, RC2 is able to find an optimal solution in 5 out of 6 instances, but in the fifth instance RC2 is not able to output a solution within the time limit.

Then, we used the results obtained by the different solvers with the instances of the first sub-problem to generate 35 instances for testing the second sub-problem. The 35 instances

correspond to the 31 optimal solutions we obtained by testing the first sub-problem, plus the 4 solutions that were obtained anyway reaching the time limit, though not guaranteed to be optimal. In particular, each solution obtained with the instances of the first sub-problem by the solvers represents a schedule in input for the second sub-problem. We repeated the same automatic translation we did to obtain the instances of the first sub-problem. In the following, we summarize the results. In the second sub-problem, CLINGO-USC, RC2, and OPEN-WBO are able to obtain an optimal solution within the time limit on each instance. Moreover, the average time required to solve the instances by these solvers is very similar: CLINGO-USC requires, on average, approx. 16 seconds, while RC2 and OPEN-WBO require on average approx. 13 seconds. Concerning the other solvers, MAXHS is able to find an optimal solution to all the instances but one while CLINGO-ROM cannot output the optimal solution to 4 out of the 35 instances tested. Finally, GUROBI cannot find the optimal solution to any of the instances tested within the time limit. We have analyzed its behavior, and we noticed that GUROBI spends, on these instances, a lot of time in the pre-solving phase: In order to find optimal solutions, the timeout should be moved to the order of tens of minutes.

Finally, we discuss advantages and reasons for the ASP approach compared to the other formalisms, to complement the analysis focused on performance above. Arguably, ASP offers a number of advantages, including: (i) The ASP specifications are often appreciated even by non-experts since they find them readable. This is important when the solution has to be used in real applications; (ii) ASP allows expressing and solving multi-level optimizations; (iii) There are free and open source systems (like CLINGO), whose performances are often comparable (or even better) to the solvers for the other formalisms (as confirmed by our experiments).

3.5 Rescheduling

In this section, we present a solution to the PAC rescheduling problem, considering different scenarios in which rescheduling could be required. In the hospital context is vital to be able to react to problems with a scheduled solution; in fact, it can happen that, due to some unpredictable events, it is not possible to implement the computed schedule. Therefore, it is necessary to find a new schedule that guarantees the proper execution of the pre-operative assessment for every patient assigned in the original scheduling. We consider three different scenarios in which rescheduling can be required:

1. Some patients do not show up on the assigned day.

2. Some operators are missing for some days.

3. Exam locations are unavailable for some days.

Thus, given a schedule and the information of why it is not possible to implement it, the PAC rescheduling problem consists of moving patients and operators in order to find a new assignment that takes into account the new information and the constraints of the initial problem. In particular, we considered the rescheduling after the second sub-problem: This is because either we want to reschedule, in case the result of the whole process, and, given the first sub-problem is usually done well in advance, it can be more appropriate to perform another scheduling from scratch. The optimization of the rescheduling problem consists in minimizing the changes made to the initial scheduling, according to some criteria: The priority is to maintain the day on which patients were assigned and, in case it is not possible, to minimize the distance between the new and the old assigned day. Then, if a patient is rescheduled on the same day we want to minimize the distance between the new and the old exams starting time, otherwise, since the day is changed, it is not important to maintain the same exams starting time, thus it is minimized the patient's total length of the pre-operative assessment as it was done in the second sub-problem of the PAC scheduling problem (Section 3.3.2). Finally, if possible, operators are assigned in the same exam areas they were initially assigned; but, while a solution that does not change the assigned day to the operators is preferred, a proper solution must still activate all the exam areas required by the patients. Thus, operators can be moved from one day to another, in order to activate the required exam areas. Moreover, in case of unavailable operators, if it is possible to replace the operator so that the exam area in which (s)he was assigned can still be activated, then no changes are needed; otherwise, patients requiring that exam area must be rescheduled on a new day.

3.5.1 ASP encoding of the rescheduling

Data Model. The input data is the same of the PAC scheduling problem plus the following atoms:

- Instances of `reg(RID,PR,DUEDATE)` represent the registration of a patient, characterized by an id (`RID`), the priority level (`PR`), and the duedate (`DUEDATE`) (i.e., the input of the scheduling without TARGET, which is not anymore useful, and TOT_DUR, which is not anymore valid).

- Instances of `x(RID,AREAID,ST,ET,DAY)` represent the previous scheduling, i.e., the output of the encoding in Section 3.3.2, without the priority field.

- Instances of `operator(ID,AREAID,DAY)` represent the previous scheduling, i.e., the output of the encoding in Section 3.3.1.
- Instances of `forbidden(RID,DAY)` represent the day (`DAY`) in which a patient (`RID`) can not show up.
- Instances of `notAvailableExamLoc(AREAID,DAY)` represent the day (`DAY`) in which an exam location (`AREAID`) is not available.
- Instances of `notAvailableOperator(ID,DAY)` represent the day (`DAY`) in which an operator (`ID`) can not show up.

Output. The output is similar to the output of Section 3.3.2, where the new schedule is represented by two atoms, `y(RID,AREAID,ST,ET,DAY)` and `operatorY(ID,AREAID,DAY)`, having the same meaning of atoms `x` and `operator` of Section 3.3.2.

Encoding. The ASP encoding of the PAC rescheduling problem we realized is made by the rules in Figure 3.9, where we denote with r_i rule appearing at the line i, plus rules r_5 and r_6 from Figure 3.2, where atom `operatorY` replaces atom `operator`, rules r_2-r_4 from Figure 3.3 (corresponding to the constraints of the second sub-problem), rules r_6 and r_7 from Figure 3.4 (for computing atoms `forbiddenAfter` and `forbiddenBefore` in the first optimization), and rule r_9 in Figure 3.5 (corresponding to the second optimization), where atom y replaces atom x.

In Figure 3.9, rules r_1 and r_2 define two auxiliary atoms that represent the possible days in which a registration can be assigned and the possible starting times of the exams, respectively. Rules from r_3 to r_5 define an auxiliary atom (`changeDay`) that represents the days in which there are changes due to patients or operators (un)availability or changes in the availability of the exam locations. Rule r_6 assigns a new day and a new starting time to all the exams required by the registrations that can not show up in the previously assigned day. Rule r_7 assigns a new day and a new starting time to all the exams required by the registrations that were assigned on a day involved in the changes. Rule r_8 reassigns the same day and starting time to all exams of the still available registrations that were assigned on a day without changes. Then, rule r_9 ensures that, for each registration, all his/her exams are assigned on the same day. Finally, weak constraints r_{10}-r_{13} scecify the optimization in a prioritized way. Weak constraints r_{10} and r_{11} minimize the distance between the old and the new date assigned to each registration, and between the old and the new starting time assigned to every exam, respectively. Whereas, weak constraint r_{12} is used to minimize the time spent in the

```
1 allowedDay(RID,DAY):- reg(RID,_,DUEDATE), time(DAY,_), not
    forbidden(RID,DAY), DAY < DUEDATE.
2 allowedTime(RID,AREAID,ST..60-ET) :-
    forbiddenBefore(RID,AREAID,ST),forbiddenAfter(RID,AREAID,ET).
3 changedDay(DAY) :- y(RID,_,_,_,DAY), x(RID,_,_,_,DAY1), DAY != DAY1.
4 changedDay(DAY) :- changedExamLoc(_,DAY).
5 changedDay(DAY) :- changedOperator(_,DAY).
6 1{y(RID,AREAID,ST,ST+DUR,DAY): allowedTime(RID,AREAID,ST),
    allowedDay(RID,DAY)}1 :- reg(RID,_,DUEDATE), exam(RID,AREAID,DUR),
    forbidden(RID,_).
7 1{y(RID,AREAID,ST,ST+DUR,DAY): allowedTime(RID,AREAID,ST),
    allowedDay(RID,DAY)}1 :- reg(RID,_,DUEDATE), exam(RID,AREAID,DUR),
    not forbidden(RID,_), x(RID,_,_,_,DAY1), changedDay(DAY1).
8 y(RID,AREAID,ST,ET,DAY) :- x(RID,AREAID,ST,ET,DAY), not
    changedDay(DAY), not forbidden(RID,_).
9 :- y(RID,_,_,_,DAY1), y(RID,_,_,_,DAY2), DAY1 > DAY2.
10 :~ y(RID,_,_,_,DAY),x(RID,_,_,_,DAYX), changedDay(DAY).
    [|DAYX-DAY|@4,RID]
11 :~ y(RID,AREAID,STy,_,DAY),x(RID,AREAID,STx,_,DAY).
    [|STx-STy|@3,RID,AREAID]
12 :~ y(RID,first_exam,ST,_,_), y(RID,last_exam,_,ET,_),
    changedDay(DAY), cost(RID,TOT), ET-ST-TOT >= 0. [ET-ST-TOT@2,RID]
13 :~ operatorY(ID,AREAID1,DAY), operators(ID,AREAID2,DAY), AREAID1 !=
    AREAID2. [1@1,ID,DAY]
```

Figure 3.9 ASP encoding of the rescheduling problem.

hospital by patients, while weak constraint r_{13}, which has the lowest priority level, minimizes the number of operators assigned to different exam locations than on the previous scheduling.

3.5.2 Results of the rescheduling

Here, the results of an empirical analysis of the PAC rescheduling problem are presented. As for the PAC scheduling problem, the ASP system used was CLINGO [Gebser et al., 2016], ver. 5.4.0, using parameters *--restart-on-model* for faster optimization and *--parallel-mode 6* for parallel execution. The time limit was set to 60 seconds.

Since the rescheduling problem consists of, given a scheduling that cannot be implemented, moving operators and patients to find a new assignment, the initial scheduling was taken from all the results of the second sub-problem run with CLINGO, considering

the benchmarks with 40 patients. We started from the results obtained in the second subproblem with 40 patients. Then, according to the different scenarios, we added the unavailability of patients, exam locations, or operators using the atoms `forbidden(RID,DAY)`, `notAvailableExamLoc(AREAID,DAY)`, and `notAvailableOperator(ID,DAY)` as presented in Section 3.5.1. In particular, an unavailable operator will be set unavailable for 5 consecutive days, while an unavailable exam location will be set unavailable for 3 consecutive days. To test the scalability of the solution, we considered for each scenario 1, 2, and 4 among exam locations, operators or patients unavailable. The unavailability was randomly selected among the assigned one in the original scheduling.

Results for the three identified scenarios are described in the next three paragraphs, while a fourth paragraph is devoted to presenting an example of the rescheduling.

Table 3.4 Average time required to obtain the optimal solution in the different scenarios.

Resource Unavailable	TIME (S) SCENARIO 1	TIME (S) SCENARIO 2	TIME (S) SCENARIO 3
1	31.0	27.1	19.0
2	49.5	31.2	26.2
4	51.6	49.6	48.6

First scenario: Patients require a new day. In this scenario, it is not possible for one (or more) patient(s) to have the pre-operative assessment on his/her planned day of the initial scheduling.

We tested the 10 instances with 1, 2, and 4 patients to be rescheduled. With 1 patient unavailable, the solution is able to find the optimal solution in 7 out of 10 instances tested; while with 2 and 4 patients unavailable, the solution finds the optimal solution in 2 out of 10 instances. About the timings, with 1 patient to reschedule, the solution was able to reach the optimal solution on average in 31 seconds, while with 2 and 4 patients, the results are obtained on average in approx. 49 and 51 seconds, respectively.

Second scenario: Operators are not available. In this scenario, it is considered the case in which an operator (or more) is (are) not available.

We tested the 10 instances with 1, 2, and 4 operators not available on the days in which they were scheduled. With both 1 and 2 operators unavailable, the rescheduler is able to find the optimal solution in 7 out of 10 instances and to find such solution in approx. 27 and 31 seconds on average, respectively. Increasing to 4 the number of unavailable operators leads to more patients involved in the rescheduling, thus, the number of instances in which the

rescheduler finds the optimal solution decreases to 2, with an average time required to find such solutions of approx. 49 seconds.

Third scenario: Exam areas are not available. In this scenario, it is supposed that an (or more) exam area(s) is (are) not available.

We tested this scenario with the 10 instances and considering the unavailability of 1, 2, and 4 exam areas. With 1 exam area unavailable, the solution obtains the optimal solution in 7 out of 10 instances and the solution is obtained on average in 19 seconds. Increasing the unavailable exam areas and testing the solution with 2 unavailable exam areas, the rescheduler still gets the optimal solution in 6 out of 10 instances, while with 4 unavailable exam areas, it finds the optimal solution in just 2 out of 10 instances. The average CPU times for computing optimal solutions for these last two cases are around 26 and 48 seconds, respectively.

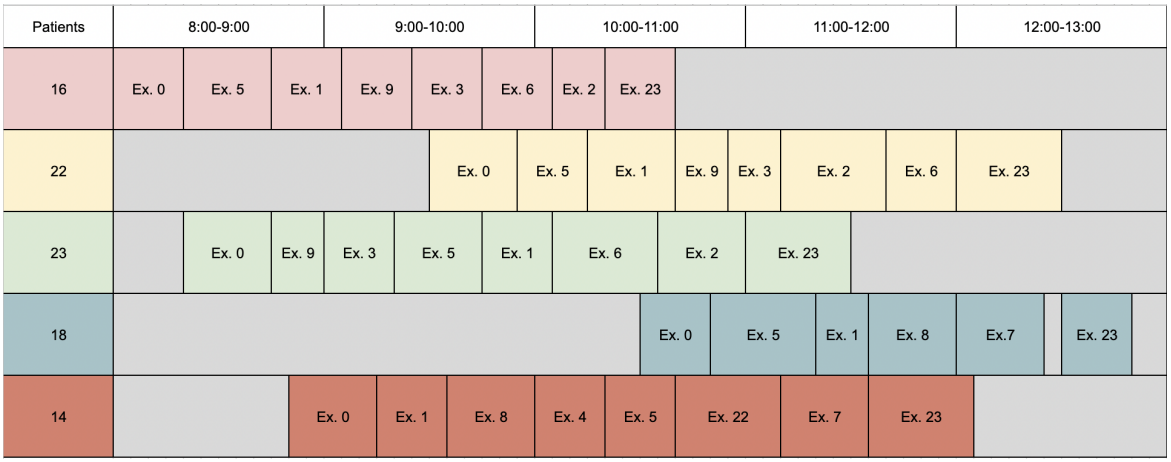


Figure 3.10 Rescheduling example related to registrations 16, 22, and 23 of Figure 3.1 rescheduled to a new day because of unavailability of exam location 3.

Example of rescheduling. We performed an analysis starting from the schedule in Figure 3.1 in which three patients have been rescheduled on the day shown due to the unavailability of the exam location with id equal to 3 required by them, whose result is shown in Figure 3.10. The three patients were scheduled originally the day before, Figure 3.1, thus, they are rescheduled on the first available day to minimize the distance between the original and the new schedule. Moreover, the image shows that there is a time slot of waiting time for patient 18. This waiting time cannot be reduced because the patients wouldn't be able to fully complete all the exams without overlapping and the rescheduler prefers to reassign this patient with one time slot of waiting time rather than move one patient to another day, i.e.,

minimizing the distance between the original day and the new one has higher priority than minimizing the waiting times.

3.6 Web Application

In view of a practical application, the solution provided in this article should be accessible to users with no or low knowledge of ASP. Thus, we developed a web application that allows any hospital operators to interact with our solution and understand the results through a graphical interface.

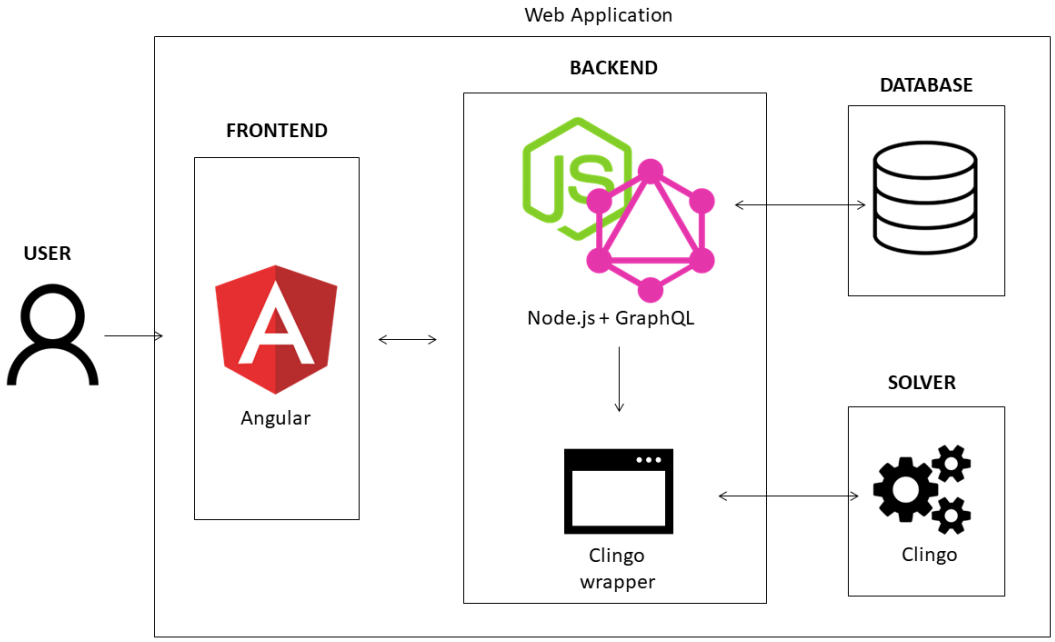


Figure 3.11 Representation of the web application architecture, together with the employed technologies.

The system architecture is reported in Figure 3.11. The graphical interface has been developed using the frameworks Angular for the frontend, Node.js for the backend, and GraphQL for data queries. We also used a wrapper inside the backend which calls the solver CLINGO. Planning results are then stored in a database. The workflow of the architecture is the following:

1. the user selects a period to plan through the frontend; data are shown according to the selection, and then (s)he can start the first planning. The result of the first planning represents the solution of the first sub-problem of the PAC scheduling problem.

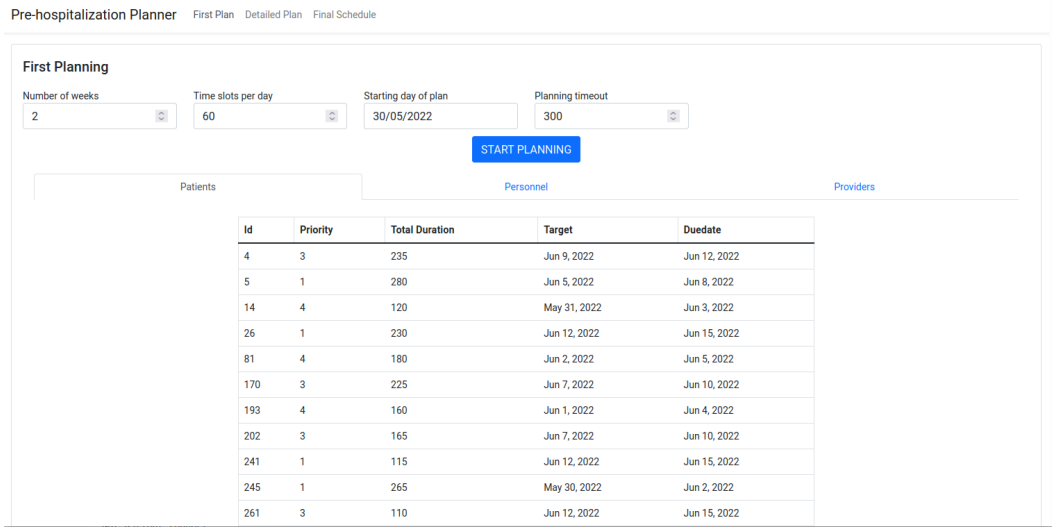


Figure 3.12 Data preparation.

2. Once the first planning is completed, the result is stored and shown. Then, the user has to select, from a default list of exams, all the exams necessary for each patient. Now, the detailed planning can start. This corresponds to the second sub-problem of the PAC problem.

3. The final scheduling is stored and shown with the precise starting time of each exam.

These three steps are further elaborated in the next three paragraphs.

Data preparation and first sub-problem. In Figure 3.12 the first page is shown: Here the user can select the period to plan and the solver timeout. In this screen, data are read-only because in a real-world application, data should be fed through direct integration with the hospital system and its database. Three tabs then allow the user to check the elements involved in the planning:

- Patients tab, to monitor all the patients with a target date included in the planning period (Figure 3.12).
- Personnel tab, to monitor the operators available in the selected period (Figure 3.13). A drop-down menu allows to select which operator to inspect.
- Providers tab, to check all the exam areas enabled in the selected time period (Figure 3.14). A drop-down menu allows to select the exam area to inspect.

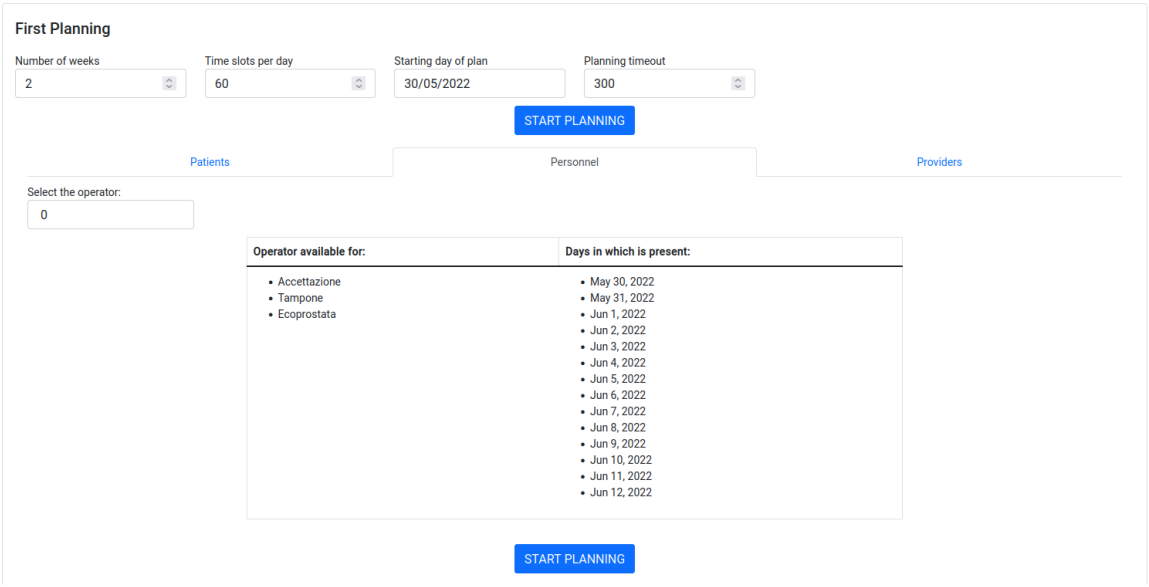
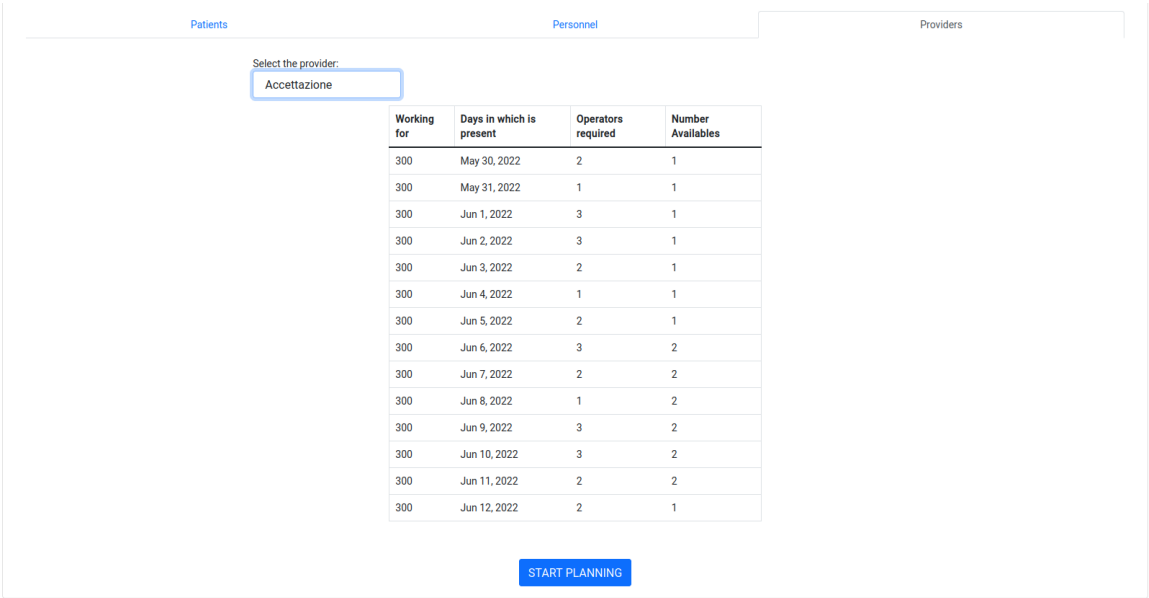


Figure 3.13 Personnel tab.



Working for	Days in which is present	Operators required	Number Availables
300	May 30, 2022	2	1
300	May 31, 2022	1	1
300	Jun 1, 2022	3	1
300	Jun 2, 2022	3	1
300	Jun 3, 2022	2	1
300	Jun 4, 2022	1	1
300	Jun 5, 2022	2	1
300	Jun 6, 2022	3	2
300	Jun 7, 2022	2	2
300	Jun 8, 2022	1	2
300	Jun 9, 2022	3	2
300	Jun 10, 2022	3	2
300	Jun 11, 2022	2	2
300	Jun 12, 2022	2	1

Figure 3.14 Providers tab

Then, by clicking the "START PLANNING" button, the solver is called through the wrapper, and the solution of the first PAC sub-problem is calculated. A loading screen will appear and once the solver finishes the planning (optimum found or timeout reached), the user is redirected to the next page, in which the results are shown.

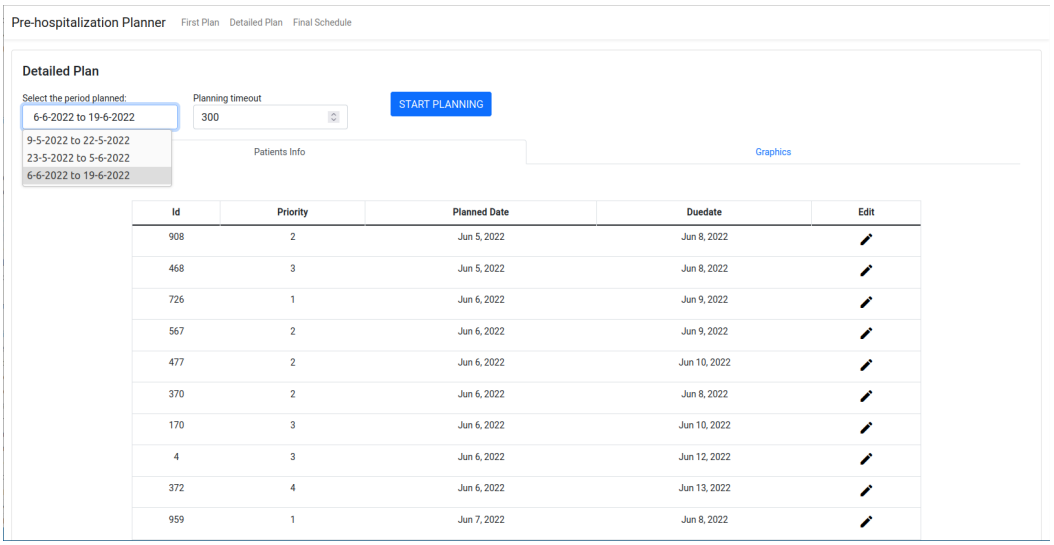


Figure 3.15 Edit patient exams.

Second sub-problem. Here, the user can look at the results of the first sub-problem; each schedule can be selected through a drop down menu. At this point, the user has to select the final list of exams required by each patient. Thus, by clicking on the edit icon, a menu will appear listing all the patient's exams, where a check box allows to select which exams are necessary, as shown in Figure 3.15.

Moreover, a second tab called "Graphics" displays the results obtained through two kinds of graphs: A bar chart (Figure 3.16) in which the bars represent the number of patients assigned to each day of the scheduled period, and four doughnuts charts (Figure 3.17) that represent the percentage of patients scheduled over the total number of patients, grouped by the patients' priority. Finally, by clicking on the "START PLANNING" button, again the solver is called on the second sub-problem and the detailed planning with the precise starting time of all the exams is computed. Once the solver terminates, the user is redirected to the final page.

Final schedule. As shown in Figure 3.18, the final schedule page is made up of a drop down menu, from which the scheduled days can be selected, and two tabs: The patients info tab and the graphics tab. In the patients info tab, patients are sorted in a table and by clicking on the icon in the "Show Exams" column, a table will appear listing all the exams and their starting time. Figure 3.19, instead, reports a chart shown in the graphics tab, in which on the x-axis there is the timeline and on the y-axis the patients, and each segment corresponds to an exam. A similar chart in the graphics tab contains the exam locations on the y-axis.

Figure 3.16 Bar chart.

Figure 3.17 Doughnut chart.

3.7 Conclusion

In this chapter, we have presented an analysis of the PAC scheduling problem modeled and solved with ASP. We started from a mathematical formulation of the problem, whose specifications come from a real scenario, and then presented our ASP solution. Results on synthetic data show that the solution is able to assign a high number of patients with high priority, and compares well to other logic-based formalisms. We have then modeled and solved the PAC rescheduling problem, which comes into play when the computed scheduling can not be implemented due to some unavailability. We finally presented a web application for easy usage of our scheduling solution.

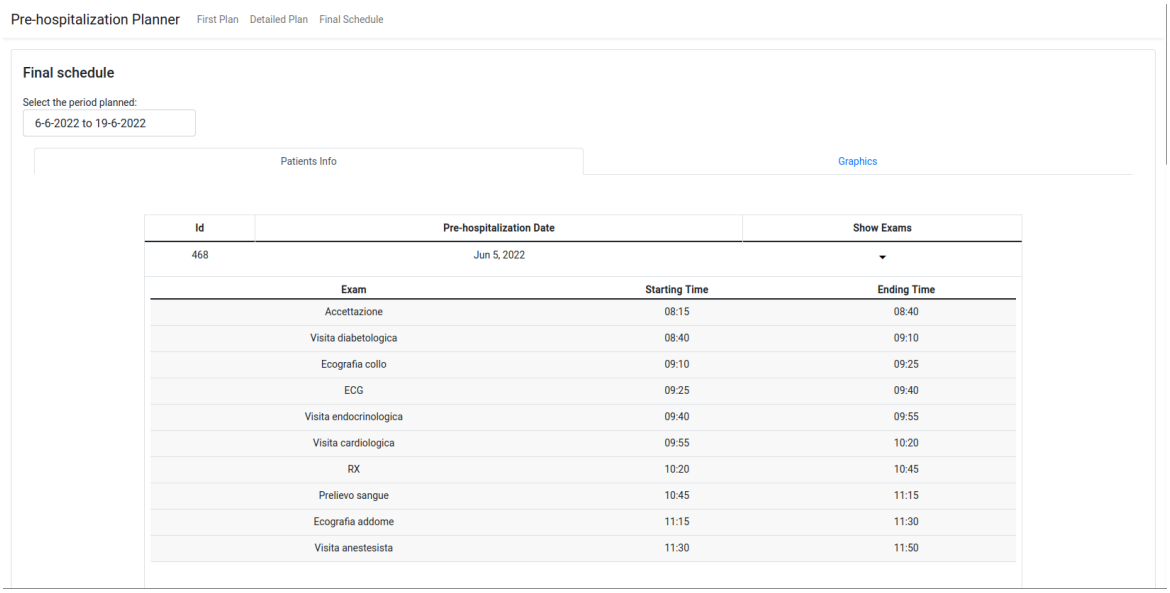


Figure 3.18 Final schedule page.

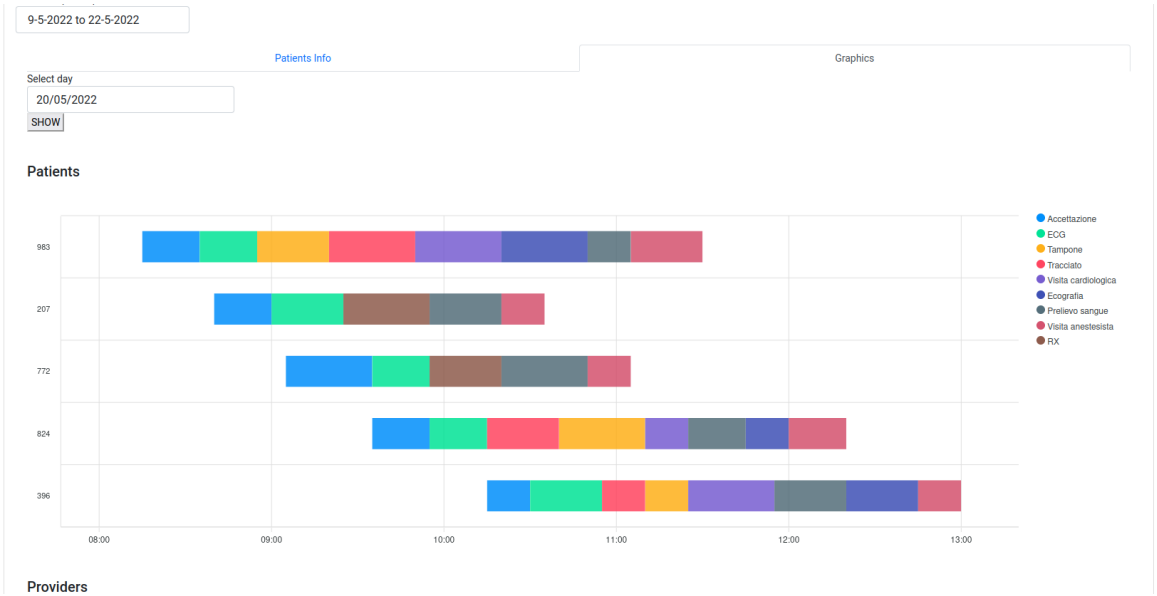


Figure 3.19 Final schedule page, graphics tab, patients chart

Chapter 4

Scheduling Periodic Treatments

In this chapter, we present the problem of computing a schedule of periodic treatments, where patients must follow a predetermined care path of treatments. For example, patients may need to attend a specific number of physiotherapy sessions followed by a certain number of minutes of exercise. Treatments must be assigned to a day taking into account patients' preferences, operators' availability, the possibility of needing instruments/machinery, and of having different facilities in which operators can work and move around during the working time. This problem has been formulated to have a certain degree of flexibility and to be suitable for different contexts where periodicity is required, e.g. rehabilitation or planning a cycle of drug therapies.

We propose an ASP encoding to compute the scheduling of periodic treatments over a fixed period of time. We then present the results of a preliminary experimental analysis, demonstrating that the ASP system CLINGO [Gebser et al., 2016], when executed on the presented encoding, achieves good performance in solving randomly-generated instances.

4.1 Problem Description

The problem of Scheduling Periodic Treatments consists in assigning multiple appointments to patients according to a care path in a period of several weeks. These assignments should take into account the availability of any instrument needed and the presence of the medical staff, if necessary. The patient's care path consists of sets of treatments that the patient should receive in the span of the planning period. Any set of treatments should be performed on the same day, minimizing the waiting times between the various treatments. A typical use case is that, after a medical examination, the patient comes with a precise list of treatments, e.g., 10 sessions of physiotherapy followed up by 10 minutes of cardio training, and asks

for the scheduling of a certain number of appointments based on their availability and on the availability of the care facility. Patients may require more than one type of treatment; in this case, the different treatments are typically scheduled on the same day, one immediately after the other, so that the patient can come just one time for all of them. In this case, it is possible that an order is provided to be respected. Patients may express time and day of the week preferences for their treatments as well as unavailable times and days.

Treatments can also have a weekly periodicity, e.g., every week or every two weeks and a maximum and minimum number of sessions per week, e.g., once or twice a week; they can also require machines/instruments and operators with a specific qualification. It is possible that operators are not necessary for the total duration of the session, e.g., the operator starts the machine and then he/she does other tasks. Moreover, care facilities can be distributed among multiple locations, and some operators can move from one to another during the day. This means that when assigning appointments, the travel time of the operator should be taken into account. Every care location can feature the relevant instruments/machinery for the type of care supported in the facility, and has a number of qualified operators assigned accordingly.

An optimal solution assigns the patients' appointments in their desired days of the week, as close as possible to a given target week, i.e., the week prescribed by the doctor and minimizes waiting time between treatments, while the operators' working time is daily maximized and weekly balanced among the available operators.

The main use case that our algorithm aims to solve is the situation when a patient calls a healthcare provider and receives a list of appointments in order to fulfill their care path. Then the patient can accept the proposed appointments or reject them and negotiate for other times, dates and locations.

However, another use case is when the healthcare provider collects a waiting list of patients and assigns their care paths at the same time. This approach has the advantage of being able to better optimize the working times of the operators and the instruments; however, it makes the negotiating phase with the patients more difficult.

4.2 ASP Encoding

In this section, we first present the data model and the expected output (Section 4.2.1), then we describe the encodings for scheduling a single patient (Section 4.2.2) and multiple patients (Section 4.2.3).

4.2.1 Data Model

The input data is specified by means of the following atoms:

- Instances of `reg(RID,CPID,PR,F)` represent the registrations characterized by an id (`RID`), the id of the associated care path (`CPID`), a priority level (`PR`) and a flag (`F`) that can be either `def` or `all`. The latter indicates that the patient requires that all the appointments must be scheduled on the same day of the week and all with the same starting time.

- Instances of `carePath(CPID,PER,NMINW,NMAXW,TAR)` represent a care path characterized by an id (`CPID`), a periodicity i.e. every how many weeks the treatments must be scheduled (`PER`), a minimum number of treatments per week (`NMINW`) and a maximum number of treatments per week (`NMAXW`) and the optimal week in which to start the care path (`TAR`).

- Instances of `treatment(TID,PCID,Q,TST,NOP,DORD,GORD)` represent a treatment characterized by an id (`TID`), the id of the associated care path (`PCID`), the necessary operator qualification (`Q`), the desirable starting time (`TST`), the number of necessary operators (`NOP`), an order value identifying the order between the treatments that have to be scheduled in the same day (`DORD`) and a global order identifying the order between the treatments to be scheduled in different appointments (`GORD`).

- Instances of `treatmentRequires(TID,PID,IID)` represent the place (`PID`) and the instrument (`IID`) required by a treatment (`TID`).

- Instances of `treatmentRequiresPlace(TID,PID)` represent the case in which a treatment (`TID`) requires only a place (`PID`) and no instruments.

- Instances of `treatmentDuration(TID,TOTDUR,OPDUR,OPS,IDUR,IS)` represent the duration of a treatment characterized by the treatment id (`TID`), the total duration of the treatment (`TOTDUR`), the duration with the operator(s) (`OPDUR`), the number of time slots that must pass from the start of the treatment before the operator(s) is needed (`OPS`), the duration with the instrument (`IDUR`) and the instrument starting time (`IS`).

- Instances of `place(PID,ST,ET,D,W,MAXPAT)` represent the places characterized by an id (`PID`), a starting time (`ST`), an ending time (`ET`), a day (`D`), the corresponding week (`W`) and the maximum capacity of the place (`MAXPAT`).

- Instances of `instrument(IID,PID,ST,ET,D,W,MAXPAT)` represent the instruments characterized by an id (`IID`), the place in which the instrument is located (`PID`), a starting time (`ST`), an ending time (`ET`), a day (`D`), the corresponding week (`W`) and the maximum capacity of the instrument (`MAXPAT`).
- Instances of `operator(OPID,ST,ET,D,W,Q,MAXPAT)` represent an operator characterized by an id (`OPID`), a starting time (`ST`), an ending time (`ET`), a day (`D`) and the corresponding week (`W`), the qualification/task assigned to the operator in that day (`Q`) and the maximum patients the operator can assist at the same time (`MAXPAT`).
- Instances of `travellingDuration(PID1,PID2,DUR)` represent the duration (`DUR`) of an operator movement from one place (`PID1`) to another (`PID2`).
- Instances of `compatible(PID,IID1,IID2)` represent the compatibility for an operator to stay at the same time on two different instruments (`IID1` and `IID2`) and the place in which the instruments are located (`PID`).
- Instances of `desirableDays(RID,D)` represent the patients' (`RID`) preference of a day (`D`).
- Instances of `forbidden(RID,ST,ET,D,W)` represent a forbidden time for a patient characterized by a patient id (`RID`), a starting time (`ST`), an ending time (`ET`), the day (`D`) and the corresponding week (`W`).
- Instances of `time(T,D,W)` represent the available time slots (`T`) in a day (`D`) of a week (`W`).
- All the atoms presented in the next paragraph represent the patients and operators already scheduled.

Output. The output is specified by means of the following atoms:

- Instances of `x(RID,TID,ST,ET,D,W,DORD,GORD)` represent an assignment of a treatment characterized by a patient id (`RID`), a treatment id (`TID`), a day (`D`) and the corresponding week (`W`), a starting time (`ST`), an ending time (`ET`), the daily order of the treatment (`DORD`) and the global order value (`GORD`).
- Instances of `assignedPlace(PID,TID,RID,ST,ET,D,W)` represent the assignment of a patient to a place characterized by a place id (`PID`), a treatment id (`TID`), a patient

id (RID), a day (D) and the corresponding week (W), a starting time (ST) and an ending time (ET).

- Instances of assignedInstrument(IID,PID,TID,RID,ST,ET,D,W) represent the assignment of a patient to an instrument/machine characterized by an insrument id (IID), a place id (PID), a patient id (RID), a treatment id (TID), an instrument id (IID), a day (D) and the corresponding week (W), a starting time (ST) and an ending time (ET).
- Instances of assignedOpR(OPID,RID,TID,PID,IID,D,W) represent the assignment of an operator to the resources of a treatment characterized by an operator id (OPID), a patient id (RID), a treatment id (TID), a place id (PID), an instrument id (IID), a day (D) and the corresponding week (W).
- Instances of assignedOpT(OPID,RID,TID,ST,ET) represent the assignment of an operator (OPID) to a patient's (RID) treatment (TID) with a starting time (ST) and an ending time (ET).
- Instances of prevWork(OPID,W,S) represent the sum of all the working hours (S) of the operators (OPID) for each week (W).

4.2.2 Encoding for Single Patient Scheduling

The encoding for the scheduling of a single patient is described in the following.

As the first operation, we assign a day and a week to the first treatment of the new registration using the following rule:

```
1  {assignedDay(RID,TID,D,W,0,0): time(_,D,W)} = 1 :-
     reg(RID,CPID,_,def), carePath(CPID,_,_,_,_),
     treatment(TID,CPID,_,_,_,0,0).
```

Then, for each treatment, we compute the assignable weeks using the following rule:

```
2 assignableWeek(RID,TID,W) :- reg(RID,CPID,_,_),
  assignedDay(RID,_,_,SW,0,0), time(_,_,W),
  treatment(TID,CPID,_,_,_,0,GORD), carePath(PCID,PER,NMIN,NMAX,_),
  (SW-W)\PER = 0, W >= SW + (GORD/NMAX), W <= SW + (GORD/NMIN).
```

Assignable weeks range between the minimum and the maximum number of weekly visits, and they are used by the following rule:

```
3 {assignedDay(RID,TID,D,W,0,GORD) : assignableWeek(RID,TID,W),
    time(_,D,W), W >= PW} <= 1 :- reg(RID,CPID,_,def),
    carePath(CPID,_,_,_,_), treatment(TID,CPID,_,_,_,0,GORD), GORD >
    0, assignedDay(RID,_,_,PW,_,GORD-1).
```

The above rule assigns, for each initial daily treatment (different treatments can be needed on the same day), a day and a week (among the ones in the assignable weeks).

We then handle the case where the patient requires that all the appointments must be scheduled on the same day of the week and all with the same starting time (expressed by the latest parameter of the registration set to `all`). In this case, the following rules are used to first select the days of the week, and the starting times of the treatments, and then, to assign the day and the week for each treatment:

```
4 {chosenDays(RID,D): time(_,D,_)} = NMAX:-
    carePath(PCID,PER,NMIN,NMAX,_), reg(RID,PCID,_,all).
5 {chosenDays(RID,D,T): time(T,D,_)} = 1 :- chosenDays(RID,D).
6 {assignedDay(RID,TID,D,W,0,0): time(_,D,W), chosenDays(RID,D,T),
    availableTime(RID,TID,T,D,W)} = 1 :- reg(RID,CPID,_,all),
    carePath(CPID,_,_,_,_), treatment(TID,CPID,_,_,_,0,0).
7 {assignedDay(RID,TID,D,W,0,GORD) : assignableWeek(RID,TID,W),
    time(_,D,W), chosenDays(RID,D,T), availableTime(RID,TID,T,D,W), W
    >= PW } <= 1 :- reg(RID,CPID,_,all), carePath(CPID,_,_,_,_),
    treatment(TID,CPID,_,_,_,0,GORD), assignedDay(RID,_,_,PW,_,GORD-1).
```

Subsequently, if there are multiple treatments to be performed on the same day, they are assigned to the initial daily treatment using the following rule:

```
8 assignedDay(RID,TID,D,W,DORD,GORD) :- reg(RID,CPID,_,_),
    carePath(CPID,_,_,_,_), assignedDay(RID,_,D,W,0,GORD),
    treatment(TID,CPID,_,_,_,DORD,GORD), DORD > 0.
```

Then, we have to ensure that *(i)* between two weeks there is not a skipped week, taking into account that the periodicity of the care path; *(ii)* the order between the treatments is respected, therefore the treatments with greater global order are assigned after those with lower global order; *(iii)* the minimum and maximum numbers of weekly treatments are fulfilled. The following constraints are used in this respect:

```
9 :- assignedDay(RID,_,_,W1,_,_), assignedDay(RID,_,_,W2,_,_), not
    assignedDay(RID,_,_,W1+PER,_,_), W2 > W1, carePath(CPID,PER,_,_,_).
```

```
10 :- assignedDay(RID,TID1,D1,W,0,GORD1),
      assignedDay(RID,TID2,D2,W,0,GORD2), GORD2 > GORD1, D2 <= D1.
11 :- assignedDay(RID,_,_,W,_,_), #max{WM: assignedDay(RID,_,_,WM,_,_)}
      > W, reg(RID,CPID,_,_), #count{D: assignedDay(RID,_,D,W,_,_)} = N,
      N < NMIN, carePath(CPID,_,NMIN,_,_).
12 :- assignedDay(RID,_,_,W,_,_), reg(RID,CPID,_,_), #count{D:
      assignedDay(RID,_,D,W,_,_)} = N, N > NMAX,
      carePath(CPID,_,_,NMAX,_).
```

Then, we define a support atom, namely `availableTime(RID,TID,T,D,W)`, representing the time slots in which each treatment can be scheduled:

```
13 forbiddenT(RID,STF..ETF,D,W) :- reg(RID,_,_,_),
      forbidden(RID,D,STF,ETF,W).
14 maxStartingTime(RID,TID1,eot-TOTDUR1-M) :- reg(RID,CPID,_,_),
      treatment(TID1,CPID,_,_,_,DORD1,GORD),
      treatmentDuration(TID1,TOTDUR1,_,_,_,_,_), #sum{TOTDUR2,TID2:
      treatment(TID2,CPID,_,_,_,DORD2,GORD),
      treatmentDuration(TID2,TOTDUR2,_,_,_,_,_), DORD2 > DORD1} = M.
15 minStartingTime(RID,TID1,M) :- reg(RID,CPID,_,_),
      treatment(TID1,CPID,_,_,_,DORD1,GORD), #sum{TOTDUR2,TID2:
      treatment(TID2,CPID,_,_,_,DORD2,GORD),
      treatmentDuration(TID2,TOTDUR2,_,_,_,_,_), DORD2 < DORD1} = M.
16 availableTime(RID,TID,T,D,W) :- maxStartingTime(RID,TID,MAX),
      minStartingTime(RID,TID,MIN), time(T,D,W),
      treatmentDuration(TID,DUR,_,_,_,_,_), T >= MIN - DUR, T <= MAX.
```

This represents a domain specific optimization, since there is an order between the treatments we know the first and last possible assignable time slot for each treatment, so that the treatments that come before and/or after have enough space.

The defined support atom is then used to assign a starting time to each treatment by means of the following rules:

```
17 {x(RID,TID,ST,ST+TOTDUR,D,W,0,GORD) : availableTime(RID,TID,ST),
      time(ST,D,W), treatmentDuration(TID,TOTDUR,_,_,_,_,_) } = 1 :-
      reg(RID,_,_,def), assignedDay(RID,TID,D,W,0,GORD).
18 x(RID,TID,ST,ST+TOTDUR,D,W,0,GORD) :- reg(RID,_,_,all),
      availableTime(RID,TID,ST,D,W), chosenDays(RID,D,ST),
```

```
    treatmentDuration(TID,TOTDUR,_,_,_,_),
    assignedDay(RID,TID,D,W,0,GORD).
{x(RID,TID,ST,ST+TOTDUR,D,W,DORD,GORD) : availableTime(RID,TID,ST),
    time(ST,D,W), treatmentDuration(TID,TOTDUR,_,_,_,_), ST >= ET } =
    1 :- reg(RID,_,_,_), assignedDay(RID,TID,D,W,DORD,GORD),
    x(RID,_,_,ET,D,W,DORD-1,GORD).
```

In particular, it is first assigned the starting time of the first treatment, i.e., the one with daily order equal to zero and then it is assigned the starting time to the following treatments, ensuring that they are assigned after the ending time of the one before.

Then, we compute the starting and the ending time of the usage of an instrument by a patient and we ensure that, for each time slot, the capacity of the instruments is not exceeded.

```
assignedInstrument(IID,PID,TID,RID,ST+MIS,ST+MIS+MIDUR,D,W) :-
    reg(RID,_,_,_), instrument(IID,PID,STM,ETM,D,W,_), ST+MIS >= STM,
    ST+MIS+MIDUR <= ETM, x(RID,TID,ST,ET,D,W,_,_),
    treatment(TID,_,_,_,_,_,_,_), treatmentRequires(TID,PID,IID),
    treatmentDuration(TID,_,_,_,MIDUR,MIS).
:- #count{RID: assignedInstrument(IID,PID,TID,RID,ST,ET,D,W), T >=
    ST, T < ET} > MAXPAT, instrument(IID,PID,_,_,D,W,MAXPAT),
    time(T,D,W).
```

Similarly, we compute the starting time and ending time of the assignment of a patient to a place and we ensure that, for each time slot, the capacity of the place is not exceeded:

```
assignedPlace(PID,TID,RID,ST,ET,D,W) :- reg(RID,_,_,_),
    place(PID,STL,ETL,D,W,_), ST >= STL, ET <= ETL,
    x(RID,TID,ST,ET,D,W,_,_), treatment(TID,_,_,_,_,_,_,_),
    treatmentRequires(TID,PID,IID).
assignedPlace(PID,TID,RID,ST,ET,D,W) :- reg(RID,_,_,_),
    place(PID,STL,ETL,D,W,_), ST >= STL, ET <= ETL,
    x(RID,TID,ST,ET,D,W,_,_), treatment(TID,_,_,_,_,_,_,_),
    treatmentRequiresPlace(TID,PID).
:- #count{RID: assignedPlace(PID,_,RID,ST,ET,D,W), T >= ST, T < ET} >
    MAXPAT, place(PID,_,_,D,W,MAXPAT), time(T,D,W).
```

Then, we assign to each treatment the required operators:

```
25 {assignedOp(OPID,RID,TID,STO,ETO) : operator(OPID,STO,ETO,D,W,Q,_) }
       = NOP :- reg(RID,_,_,_), treatment(TID,_,Q,_,_,_,NOP,_,_).
```

Subsequently, we compute the place and the instrument assigned to the operator on a given day and the corresponding patient, and the starting and ending time of the operator assignment:

```
26 assignedOpR(OPID,RID,TID,PID,IID,D,W) :-
       assignedOp(OPID,RID,TID,_,_), x(RID,TID,_,_,D,W,_,_),
       reg(RID,_,_,_), treatmentRequires(TID,PID,IID).
27 assignedOpR(OPID,RID,TID,PID,-1,D,W) :- assignedOp(OPID,RID,TID,_,_),
       x(RID,TID,_,_,D,W,_,_), reg(RID,_,_,_),
       treatmentRequiresPlace(TID,PID).
28 opidSt(RID,TID,STX+OPS,STX+OPS+OPDUR) :- reg(RID,_,_,_),
       treatmentDuration(TID,_,OPDUR,OPS,_,_), x(RID,TID,STX,ETX,_,_,_,_,_).
29 assignedOpT(OPID,RID,TID,ST,ET) :- assignedOp(OPID,RID,TID,_,_),
       opidSt(RID,TID,ST,ET).
30 :- assignedOpT(OPID,RID,TID,ST,ET), assignedOp(OPID,RID,TID,STO,ETO),
       ST < STO.
31 :- assignedOpT(OPID,RID,TID,ST,ET), assignedOp(OPID,RID,TID,STO,ETO),
       ET > ETO.
```

Then, we ensure that the operators do not move in different places in less than the necessary traveling time and that their capacity is not exceeded for each time slot:

```
32 :- assignedOpR(OPID,RID1,TID1,PID1,_,D,W),
       assignedOpT(OPID,RID1,TID1,_,ET1),
       assignedOpR(OPID,RID2,TID2,PID2,_,D,W),
       assignedOpT(OPID,RID2,TID2,ST2,_), ST2 >= ET1,
       travellingDuration(PID1,PID2,DUR), ST2 < ET1 + DUR.
33 :- #count{RID: support(OPID,RID,T,_,_,D,W)} > MAXPAT,
       operator(OPID,_,_,_,_,_,_,MAXPAT), time(T,D,W).
```

Moreover, we also ensure that an operator is not assigned to two different places at the same time and to two different instruments that are not compatible at the same time:

```
34 support(OPID,RID,ST..ET-1,PID,IID,D,W) :-
       assignedOpT(OPID,RID,TID,ST,ET),
       assignedOpR(OPID,RID,TID,PID,IID,D,W).
```

```
35 :- support(OPID,_,T,LID1,_,D,W), support(OPID,_,T,LID2,_,D,W), LID1 <
     LID2.
36 :- support(OPID,_,T,PID,MID1,D,W), support(OPID,_,T,PID,MID2,D,W),
     not compatible(PID,MID1,MID2), MID1 < MID2.
```

Finally, we are ready to express the preferences among different solutions by means of the following rules:

```
37 :~ assignedDay(RID,_,_,W,_,0), carePath(CPID,_,_,_,TAR). [|W-TAR|@10]
38 :~ reg(RID,_,_,_), treatment(TID,_,_,_,_,_,_,_), not
     assignedDay(RID,TID,_,_,_,_). [1@9,TID]
39 :~ reg(RID,_,_,_), assignedDay(RID,_,_,W,_,_). [1@8,RID,W]
40 :~ reg(RID,_,_,_), assignedDay(RID,TID,D,_,_,_), not
     desirableDays(RID,D). [1@7,TID]
41 :~ reg(RID,_,_,_), x(RID,TID1,_,ET1,_,_,_,_),
     x(RID,TID2,ST2,_,_,_,_,_), treatment(TID1,_,_,_,_,DORD,GORD),
     treatment(TID2,_,_,_,_,DORD+1,GORD), ST2-ET1 > 0.
     [ST2-ET1@6,TID1,TID2]
42 :~ reg(RID,_,_,_), x(RID,TID,ST,_,_,_,_,_),
     treatment(TID,_,_,TST,_,0,_). [|ST-TST|@5,TID]
43 :~ reg(RID,_,_,_), assignedOp(OPID1,RID,TID1,_,_),
     assignedOp(OPID2,RID,TID2,_,_), TID1 < TID2, OPID1 != OPID2.
     [1@4,TID1,TID2]
44 :~ time(_,D,W), #count{OPID: assignedOpR(OPID,_,_,_,_,_,D,W)}=N.
     [N@3,D,W]
45 movement(OPID,LID1,LID2,ET1,D,W) :- reg(RID1,_,_,_),
     treatment(TID1,_,_,_,_,_,_,_), RID1 != RID2,
     assignedOpR(OPID,RID1,TID1,LID1,_,D,W),
     assignedOpT(OPID,RID1,TID1,ST1,ET1),
     assignedOpR(OPID,RID2,TID2,LID2,_,D,W),
     assignedOpT(OPID,RID2,TID2,ST2,_), ST2 > ST1, LID2 != LID1,
     #count{ST3: assignedOpR(OPID,RID3,TID3,_,_,D,W), RID1 != RID3,
     assignedOpT(OPID,RID3,TID3,ST3,_), ST3 > ST1, ST3 < ST2} < 1.
46 movement(OPID,LID2,LID1,ET2,D,W) :- reg(RID1,_,_,_),
     treatment(TID1,_,_,_,_,_,_,_), RID1 != RID2,
     assignedOpR(OPID,RID1,TID1,LID1,_,D,W),
     assignedOpT(OPID,RID1,TID1,ST1,_),
```

```
    assignedOpR(OPID,RID2,TID2,LID2,_,D,W),
    assignedOpT(OPID,RID2,TID2,ST2,ET2), ST1 > ST2, LID2 != LID1,
    #count{ST3: assignedOpR(OPID,RID3,TID3,LID3,_,D,W), RID1 != RID3,
    assignedOpT(OPID,RID3,TID3,ST3,_), ST3 > ST2, ST3 < ST1} < 1.
47 :~ movement(OPID,LID1,LID2,T,D,W). [1@3,OPID,LID1,LID2,T,D,W]
48 :~ time(_,D,W), support(OPID,_,T,_,_,D,W), not
    support(OPID,_,T+1,_,_,D,W). [1@2,OPID,T,D,W]
49 totWorkOp(OPID,W,0..S+T) :- time(_,_,W), prevWork(OPID,W,S),
    #sum{OPDUR,TID: assignedOpR(OPID,RID,TID,_,_,_,W), reg(RID,_,_,_),
    treatmentDuration(TID,_,OPDUR,_,_,_)} = T,
    operator(OPID,_,_,_,_,_,_,_).
50 :~ time(_,_,W), totWorkOp(OPID1,W,T), operator(OPID1,_,_,_,_,_,_,_),
    not totWorkOp(OPID2,W,T), operator(OPID2,_,_,_,_,_,_,_). [1@1,W]
```

In particular, rule 37 is used to minimize the distance in absolute value between the week in which the care path actually starts and the optimal one. Then, since it is possible that not all the treatments are scheduled (for instance, when there are not enough operators), we have to maximize the number of assigned treatments (rule 38), and we also maximize the number of weekly treatments (rule 39). This is obtained by minimizing the total weeks. In rule 40, it is penalized the case where a treatment is not assigned to a day preferred by the patient and then, in the subsequent rule, the waiting time between treatments is minimized. Rule 42 is used to minimize the distance between the starting time of a treatment and the time desired by the patient, whereas rule 43 enforces, as much as possible, assigning the same operator to a patient for each treatment. Then, in rules 44 and 47 we have the two possible operator optimization strategies, the care facility can choose the one to adopt. In particular, rule 44 minimizes the number of operators working each day, whereas rule 47 minimizes the daily movements of the operators. Finally, rules 48 and 50 balance the total working times of the operators for each week.

4.2.3 Encoding for Multiple Patients Scheduling

For scheduling multiple patients at the same time, we require as input a list of patients with the corresponding care pathway, available instruments, operators and facilities, and information inherent to instrument compatibility, duration moves and patient preferences.

The encoding is then the same presented in Section 4.2.2, where only the following rules are modified to be adapted to the presence of multiple patients.

In particular, rule 9 is replaced as follows:

```
9 :- reg(RID,CPID,_,_), assignedDay(RID,_,_,W1,_,_),
     assignedDay(RID,_,_,W2,_,_), not assignedDay(RID,_,_,W1+PER,_,_),
     W2 > W1, carePath(CPID,PER,_,_,_).
```

In this case, we ensure that, for each patient, a week of treatment is not skipped.

Moreover, rule 32 is replaced as follows:

```
32 :- support(OPID,RID1,T1,PID1,_,D,W),
     support(OPID,RID2,T2,PID2,_,D,W), RID1 != RID2, T1 < T2, T2 < T1 +
     DUR + 1, travellingDuration(PID1,PID2,DUR).
```

As in the previous encoding, we ensure that the operators do not move in different places in less than the necessary traveling time.

Then, rules 37, 38, and 42 are slightly modified as follows to be suitable for the scheduling of a list of patients:

```
37 :~ reg(RID,CPID,_,_), assignedDay(RID,_,_,W,_,0),
     carePath(CPID,_,_,_,TAR). [|W-TAR|@10,RID]

38 :~ reg(RID,CPID,_,_), treatment(TID,CPID,_,_,_,_,_), not
     assignedDay(RID,TID,_,_,_,_). [1@9,TID]

42 :~ reg(RID,CPID,_,_), x(RID,TID,ST,_,_,_,_,_),
     treatment(TID,CPID,_,TST,_,0,_). [|ST-TST|@5,TID]
```

4.3 Preliminary Experimental Analysis

In this section we report the results of an empirical analysis conducted to assess the performance of the proposed encoding. The experiments were run on an AMD Ryzen 5 3600 CPU @ 3.60GHz with 16 GB of physical RAM. As ASP system, we used CLINGO [Gebser et al., 2016] version 5.4.1, configured with the option --parallel-mode 6 for parallel execution. This setting is the result of a preliminary analysis done on the same instances where we also tested other parameters, e.g., --opt-strategy=usc for optimization. The time limit was set to 60 seconds for the scheduling of single patients and 300 seconds for the scheduling of multiple patients. Encodings and benchmarks employed in this section can be found at: https://drive.google.com/file/d/1OhYaP_7dfSRzZ8GyWzRgAdb84SRWtkOp/view.

Table 4.1 Results of the experiments in Scenario 1.

Number of treatments	Avg time
8 treatments (in 4 appointments)	3.39s
12 treatments (in 5 appointments)	4.95s
18 treatments (in 6 appointments)	7.51s

Benchmarks. The data used in the experiments were randomly generated. In particular, we considered a period of 10 weeks, each with 7 working days, and for each day, we considered 60 time slots. Moreover, we considered a scenario with 4 operators that can move between two facilities, one with 3 instruments and the other with 2 instruments. Then, for each patient, we generated a minimum and maximum number of weekly treatments. For each patient, we considered two possible ranges of such values, i.e., 1 (min) and 2 (max), or 2 (min) and 3 (max). The optimal week in which to start the treatments was set in the first 4 weeks and the number of treatments per appointment, instead, varied between 2 and 3 treatments, with a duration ranging from 3 to 12 time slots.

While these parameters were held constant, patient preferences and their number of treatments were varied. Due to the high flexibility of the problem, several scenarios are possible:

1. Scenario 1: we have a scheduling of a single patient at a time with a variable number of treatments, and a daily minimization of the total number of operators, and for each patient, we have at least 1 and at most 2 weekly treatments.

2. Scenario 2: we have a scheduling of a single patient at a time with a variable number of treatments, and a daily minimization of the total number of operators, and for each patient we have at least 2 and at most 3 weekly treatments.

3. Scenario 3: as in Scenario 2, where patients require the same weekly day and start time.

4. Scenario 4: as in Scenario 2, where instead of minimizing the number of daily working operators, the number of operators' displacements is minimized.

5. Scenario 5: as in Scenario 1 (Scenario 5a) and in Scenario 2 (Scenario 5b), where we consider blocks of patients of sizes 5 and 10 per instance.

Results with patients scheduled one by one. Tables 4.1, 4.2, and 4.3 show the results when it is required to minimize the daily number of working operators. Table 4.1 shows the

Table 4.2 Results of the experiments in Scenario 2.

Number of treatments	Avg time
12 treatments (in 6 appointments)	5.18s
18 treatments (in 7 appointments)	8.1s
24 treatments (in 12 appointments)	10.77s
27 treatments (in 9 appointments)	13.44s
36 treatments (in 12 appointments)	18.84s

Table 4.3 Results of the experiments in Scenario 3.

Number of treatments	Avg time
12 treatments (in 6 appointments)	5.91s
18 treatments (in 7 appointments)	10.49s
27 treatments (in 9 appointments)	20.4s

Table 4.4 Results of the experiments in Scenario 4.

Number of treatments	Avg time
12 treatments (in 6 appointments)	6.5s
18 treatments (in 7 appointments)	10.32s
27 treatments (in 9 appointments)	15.25s

average solving time for the instances with the minimum and the maximum number of weekly appointments for each patient set to 1 and 2, while Table 4.2 shows the average solving time for the instances with the minimum and the maximum number of weekly appointments for each patient set to 2 and 3.

It is possible to observe that the solving time depends on the number of treatments. Table 4.3 shows the results obtained by generating patients who always require keeping the weekly days and start times fixed. In Table 4.4 are shown the results using the operator optimization strategy: minimize daily movements. Results show that the optimum is always found with an average time of less than 20 seconds. Patients are always assigned in their ideal starting week, respecting weekly day preferences, and in most cases, all treatments are scheduled; however, as the number of patients increases and the agenda becomes saturated, some treatments are not assigned. In addition, waiting times between treatments are minimal, increasing as the agenda becomes thicker due to the fact that greater priority is given to respect the target start week and patient day preferences. Moreover, the analysis shows that the operators' weekly workloads are correctly distributed, with an average distance, for each operator, from the weekly average of less than 15 time slots for all the instances.

Table 4.5 Results of the experiments in the Scenario 5a and 5b.

Scenario	N.Pat/instance	Avg.Treat/instance	Avg.Time	% OPT
5a	5 patients	57	48.52s	100%
5a	10 patients	125	246.54s	60%
5b	5 patients	93	126.94s	90%
5b	10 patients	188	299s	10%

Results scheduling multiple patients at the same time. Scheduling blocks of patients allows us to study the scalability of the proposed solution and can be used to optimize the work time of operators and instruments. The results are shown in Table 4.5. We used blocks of 5 and 10 patients and varied the average number of total treatments per instance, i.e, the average number of treatments per patient. The results show that for 5 patients, the solver in the majority of the cases finds the optimum solution. However, the first levels are minimized also in the non-optimal solutions, ensuring a certain quality of the scheduling so that patient preferences and waiting times are respected.
For blocks of 10 patients, on the other hand, as the input grows, more time is spent in finding solutions and several instances do not reach the optimum, but a (sub-optimal) solution is still found before the timeout. Only in one case, the optimization stops at the first level, while in the other, sub-optimal results of the first levels are minimized respecting patient preferences, assigning all assignable visits, and minimizing waiting times.

4.4 Conclusion

In this chapter, we provided an ASP solution for the problem of computing a schedule of periodic treatments for patients who must follow a predetermined care path of treatments. The results of a preliminary experimental analysis show that CLINGO executed on the presented encoding achieves good performance in solving randomly-generated instances.

Part III

Controlled Natural Languages

Chapter 5

CNL2ASP

As already mentioned in the previous chapters, 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., 2017a].

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, e.g., 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.

Thus, 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, e.g., the ones related to a business, using natural language, and by PENG^{ASP}, a CNL defined by [Schwitter, 2018].

The development of the tool has been oriented towards four different types of use cases, i.e., *(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 is open source and publicly available at https://github.com/dodaro/cnl2asp.

5.1 Background on formal grammars

A *grammar* [Chomsky, 1959] is a tuple (N,T,S,P), where N and T are disjoint sets of non-terminal and terminal symbols, respectively, $S \in N$ is the starting symbol, and P is a finite binary relation defined on $(N \cup T)^* \circ N \circ (N \cup T)^* \times (N \cup T)^*$, called *production rules*, where $*$ is the Kleene star operator [Hopcroft and Ullman, 1979], while $\circ$ is the concatenation operator. According to the Chomsky Hierarchy, four types of grammar can be classified depending on the expressiveness allowed by P. For the purpose addressed in this work, we use *type 2* or *context-free grammars*, where P is restricted to relations defined on $N \times (N \cup T)^*$. Each production rule is in the form: $A \longrightarrow \alpha$, where $A \in N$ is a non-terminal symbol, and $\alpha \in (N \cup T)^*$ is a string of terminals and/or non-terminal symbols. This means that A can be replaced with α in the derivation of strings. To make grammars easier to write and read, especially for complex languages, an extended notation called Extended

Backus-Naur Form (EBNF) is often used. EBNF introduces several shorthand notations to describe repetition, optional elements, and grouping:

- $\alpha*$ indicates that α may appear zero or more time.
- $\alpha+$ indicates that α may appear zero or more time.
- (α) represents a group of elements so that the $*$ and $+$ operators apply to the entire group.
- $[\alpha]$ represents an optional symbol, α may appear zero or one time.
- $\alpha|\beta$ denotes a choice between α or β, either option can be selected but not both.

Throughout this chapter and in the next chapters, we will consistently use EBNF notation to define grammar rules. An intuitive example of a production rule using this notation is shown below:

```
symbol ⟶ non_terminal_symbol "terminal symbol" [optional]
```

5.2 CNL and tool implementation

This section deals with the specification of CNL language and with the implementation of the tool CNL2ASP, whose architecture is depicted in Figure 5.1. 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, i.e., 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

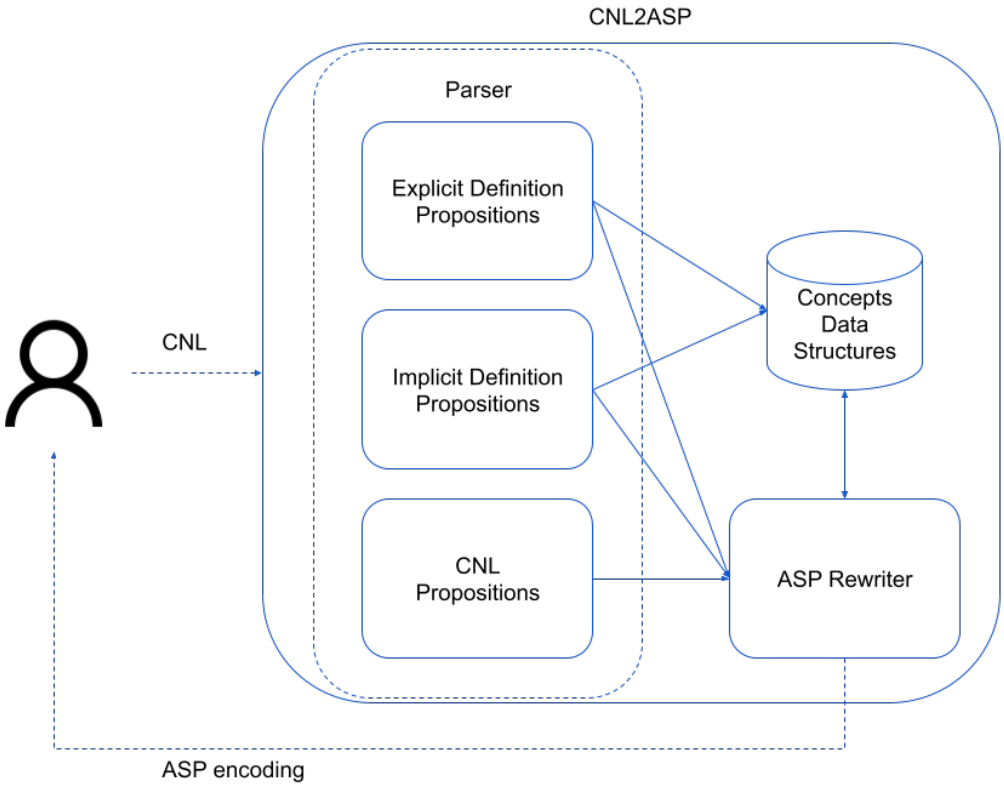


Figure 5.1 Architecture of the tool CNL2ASP.

propositions, and (standard) CNL propositions. In particular, the starting production rule is the following:

```
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 a 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 5.2.1, 5.2.2, and 5.2.3), and then we report a brief description of the usage of the tool (Section 5.2.4).

5.2.1 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:

```
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.

Domain definition

Domain definitions start with a subject optionally followed by the sentence `"is identified by"` and the definition of the keys, i.e., 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:

```
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., 2025a] 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:

```
A movie is identified by an id, and has a title, a director, and a
    year.
A director is identified by a name.
A topMovie is identified by an id.
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 5.2.3.

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:

```
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:

```
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:

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

5.2.2 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.

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.

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 5.2.2.

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.

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, i.e., `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).

5.2.3 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:

```
standard_proposition ⟶ (
        whenever_then_clause_proposition |
        fact_proposition |
        quantified_choice_proposition |
        negative_strong_constraint_proposition |
        positive_strong_constraint_proposition |
        weak_constraint_proposition |
        )CNL_END_OF_LINE
```

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

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:

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

They start with `whenever` clauses, i.e., 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:

```
Whenever there is a movie with director equal to spielberg, with id X
    then we must have a topmovie with id X.
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.
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:

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

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:

```
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:

```
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.

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

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:

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

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, i.e., 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 5.2.3; 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.

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:

```
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, i.e., 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, i.e., 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.

5.2.4 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.

5.3 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.

5.3.1 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 $i = 1..7$).

5.3.2 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.

5.3.3 Maximal clique

The third problem is the maximal clique problem, which is the problem of finding a clique, i.e., 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 $i = 1..9$).

5.4 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 5.4.1; [Dodaro and Maratea, 2017]), the Manipulation of Articulated Objects Using Dual-Arm Robots (Section 5.4.2; [Bertolucci et al., 2021a]), and the Chemotherapy Treatment Scheduling (CTS) Problem (Section 5.4.3; [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,

i.e., 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 seconds 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 Appendix A.

5.4.1 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 are 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 5.2, 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 to 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

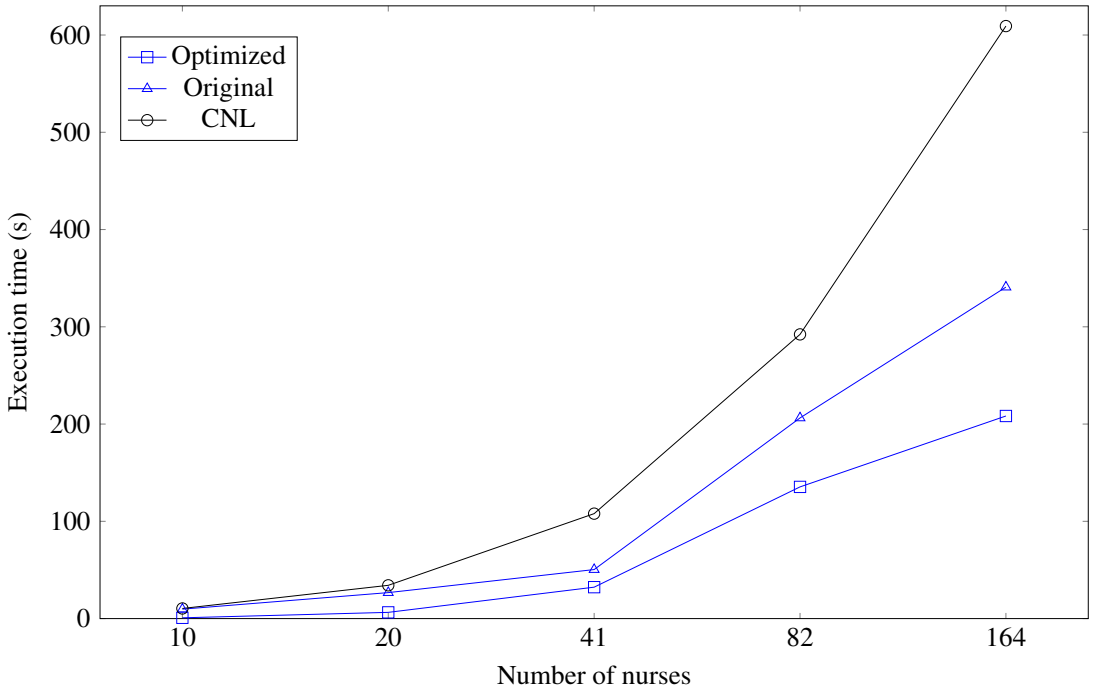


Figure 5.2 Time comparison of the performance of the original, the optimized and the CNL encodings to solve instances of the NSP.

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.

5.4.2 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., 2021a] 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., 2021a], 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 5.2.1, 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, e.g., 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

Table 5.1 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.

	12 joints, 180 granularity		12 joints, 90 granularity		14 joints, 180 granularity	
Instance	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

```
    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 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., 2021a], 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 5.1, 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.

5.4.3 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, e.g., 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 minutes.
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.
```

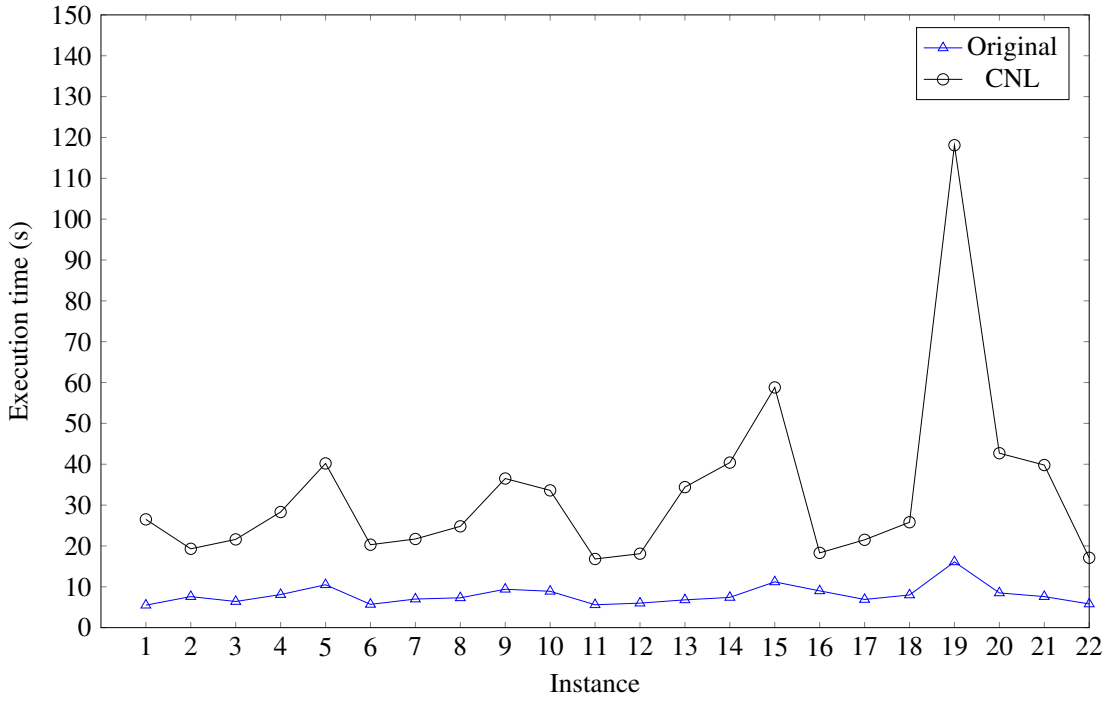


Figure 5.3 Time comparison of the performance of the original and CNL encodings to solve instances of the CTS problem.

```
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.
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 5.3. 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 seconds to compute a solution, with a peak of 2 minutes on the hardest instance.

Table 5.2 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

5.5 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 1st, 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 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.

5.5.1 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 5.2. 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, 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.

5.5.2 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.

Table 5.3 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

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 Nurse Scheduling Problem (NSP) as described by [Dodaro and Maratea, 2017]. The second group received the CNL specifications described in Section 5.4.1. 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 5.3. 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.

5.6 Conclusion

In this chapter, we introduced a CNL specifically designed for expressing and solving complex combinatorial problems. We presented CNL2ASP, a tool that automatically translates sentences written in this CNL into ASP encodings, supporting the main ASP features such as disjunctive and choice rules, aggregates, and weak constraints. We illustrated the effectiveness of our approach through a variety of use cases, including well-known synthetic domains and real-world problems from the literature. Moreover, we also provided an experimental analysis comparing the performance of automatically generated encodings with those produced by human experts. Finally, we carried out a preliminary user validation to assess the usability and readability of the proposed CNL.

Chapter 6

CNL2TEL

In this chapter, 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 [Cabalar et al., 2019], 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.

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.

6.1 Background

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

6.1.1 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 6.1. 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 $\bullet p \vee \mathsf{I}$, 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 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:

- $a \mathsf{S} b$ is true whenever there is a state in which b is satisfied and then, in the following state, a is satisfied;
- $a \mathsf{T} b$ 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 [Cabalar et al., 2018]. As [Cabalar et al., 2019] 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 $\mathscr{A}$, 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 $\{a, \neg a, \bullet a, \neg \bullet a, \mid a \in \mathscr{A}\}$, while for initial and final rules as regular literals $\{a, \neg a \mid a \in \mathscr{A}\}$. 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.

Table 6.1 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
$\bullet$ (previous)	$<A$ $'A$	before A
$\widehat{\bullet}$ (weak previous)	$<: A$	before A or it is the initial states
$\blacklozenge$ (eventually before)	$<? A$	before A that eventually holds eventually A that holds since before
$\blacksquare$ (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
$\circ$ (next)	$> A$ A'	after A
$\widehat{\circ}$ (weak next)	$>: A$	after A or it is the final state
$\lozenge$ (eventually after)	$>? A$	after A that eventually holds eventually A that holds since after
$\square$ (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

6.1.2 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$\{\varphi\}$, where φ is made of temporal operators, whose syntax is shown in the second column of Table 6.1, 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`.

6.2 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 5.2, 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 6.1.

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 6.1), 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.

6.3 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:

```
1 #program always.
2 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:

```
1 #program dynamic.
2 gun("shooting") | gun("loading") | gun("waiting") :- shooter(X).
3 gun("loaded") :- gun("loading").
4 gun("loaded") :- not gun("unloaded"), 'gun("loaded").
5 gun("unloaded") :- gun("shooting"), 'gun("loaded"), not gun("broken").
6 gun("unloaded") :- 'gun("unloaded"), not gun("loaded").
7 :- gun("loading"), 'gun("loaded").
8 gun("broken") :- gun("shooting"), not not &tel {(<* gun("unloaded"))
    & (< <? gun("shooting"))}.
9 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:

```
1 #program final.
2 :- 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:

```
An object is identified by an id.
A vehicle is identified by a object.
A truck is identified by a object.
An airplane is identified by a object.
A package is identified by an id.
A location is identified by an id.
A city is identified by a location, and by a name.
An airport is identified by a location.
A goal is identified by a package, and by a location.

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

The following propositions always apply except in the initial state:
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.
Every vehicle V can unload a package P, when a package with id P is
    previously loaded in vehicle V.
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.
A package P is loaded, when package P is loaded in a vehicle V.
A package P is loaded in vehicle V, when a vehicle V loads package P.
It is prohibited that a vehicle V1 loads a package P and also vehicle
    V2 loads package P, where V1 is different from V2.
A vehicle V has a task when vehicle V load package P.
A vehicle V has a task when vehicle V unload package P.
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.
```

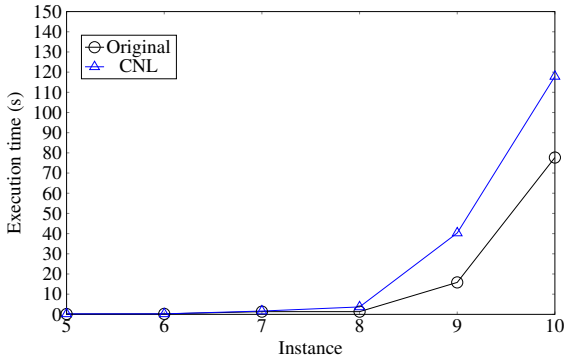
After the domain definitions in lines 1-9, the concepts of vehicles and locations are introduced. The former is a general name for trucks and aircraft while the latter is a part of city. Then, from line 14, the *dynamic* rules are defined. First, it is specified when a vehicle can load and unload a package (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.
```

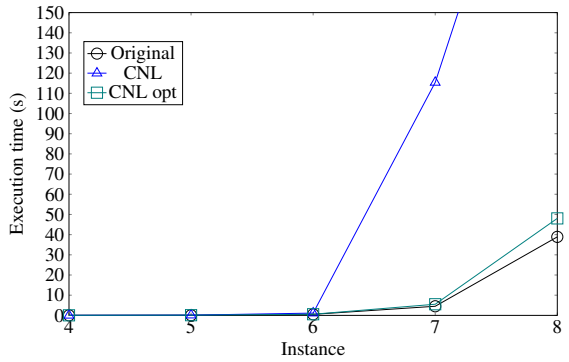
```
2 vehicle(T) :- truck(T).
3 vehicle(A) :- airplane(A).
4 location(L) :- city(L,_).

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,_).
```



(a) Logistic



(b) Hanoi

Figure 6.1 Time comparison of the performance of the original and the CNL encodings.

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 6.1. Overall, as expected, the original human-written 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.

6.4 Conclusion

In this chapter, 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.

Chapter 7

SBVR2ASP

The Semantics of Business Vocabulary and Business Rules (SBVR) [The Object Management Group, 2019] is a standard established by the Object Management Group (OMG) for the specification of business rules in a structured, yet human-readable, natural language format. SBVR enables domain experts to formally express business constraints and policies without requiring knowledge of traditional programming languages or formal logics. This makes it an essential component in model-driven development, regulatory compliance, and business process management. By fostering a shared vocabulary and unambiguous rule definitions, SBVR promotes consistency, traceability, and verifiability across enterprise systems [Ceci et al., 2016, Abi-Lahoud et al., 2013].

However, rule conflicts, defined as logical contradictions or inconsistencies within a set of sentences, can undermine system reliability, introduce ambiguities, and obstruct automation. Consequently, conflict detection in SBVR is a crucial step in the validation and verification of business rules. As a matter of fact, despite significant interest in this area, a recent systematic survey of existing approaches conducted by [Schumann and Gómez, 2024] reveals that current techniques suffer from the following significant limitations:

- Some strategies do not consider certain types of sentences, but they only include prohibitions and permissions. However, excluding some sentences from the conflict detection can lead to undetected inconsistencies.
- Many approaches have a limited context window and they only compare individual sentences. In this case, it is not possible to determine whether the missing information hidden in the other sentences can lead to inconsistencies.
- The detection is often rule-based or requires training data for supervised machine learning, which leads to two problems: the scarcity of the data due to their high costs

and it is unclear how well these methods generalize to diverse regulatory documents, as they may overfit to the specific characteristics of the data they were trained on.

- Many studies define conflict using only deontic logic (e.g., obligation vs. permission), but some situations don't involve full contradictions but discrepancies, such as when one statement is more specific than another.

- The available datasets include a limited number of conflicts or synthetically generated.

We aim to overcome the first four limitations, and partially address the fifth, by introducing a novel approach based on Answer Set Programming (ASP) [Brewka et al., 2011, Gelfond and Lifschitz, 1988] for SBVR representation and for conflict detection, that we also implemented in an open-source tool called SBVR2ASP. The novel approach offers a number of advantages compared to existing ones. Indeed, ASP is a powerful and expressive formalism that can easily model rules, constraints, and logical dependencies. Thus, our approach is capable of capturing a wide range of SBVR sentence types, supports global reasoning across an unrestricted context window, and operates without supervision or manually defined rules, relying only on the logical specification of the problem. Another advantage of our ASP-based approach lies in the fact that the ASP community has developed a rich ecosystem of tools and methodologies for explanation and debugging [Fandinno and Schulz, 2019], employed also in real applications, e.g., Healthcare [Bertolucci et al., 2021b, Dodaro et al., 2018, 2019b,a, Alviano et al., 2020, Cardellini et al., 2021, Cappanera et al., 2023a,b, Caruso et al., 2023, Dodaro et al., 2024a], also in combination with machine learning techniques (see, e.g., [Bruno et al., 2021, 2022]).which can be used to analyze and manage conflicts of the SBVR specification. Moreover, modern ASP solvers, as CLINGO [Gebser et al., 2016] and WASP [Alviano et al., 2019a], support optimized grounding and solving techniques making them perfectly suitable for large-scale reasoning tasks, and more recent solvers, e.g., [Dodaro et al., 2024b] work via compilation.

Among the limitations identified by [Schumann and Gómez, 2024], the reliance on synthetic data is particularly difficult to overcome. Unlike other issues, which can potentially be addressed through improved methodologies or technologies, the lack of real-world data is not a problem that can be solved at the technical level. It is inherently tied to the nature of the domain: real regulatory documents, business rules, and compliance policies often contain sensitive, proprietary, or strategically valuable information. As a result, organizations are generally reluctant to share such data, even in anonymized form. Indeed, these documents may include confidential legal clauses, internal governance policies, or competitive business logic that could reveal strategic insights or introduce legal risks if disclosed. Furthermore,

in many cases, the rule sets themselves represent a significant intellectual asset with direct economic value, making companies even more protective of their dissemination.

Consequently, unless stakeholders are willing to provide access to such data under specific agreements or within closed evaluation settings, the only viable alternative remains the construction of synthetic datasets. This constraint has been consistently acknowledged by [Schumann and Gómez, 2024], and our work aligns with this trend.

In line with previous studies, we evaluated our approach using publicly available SBVR specifications combined with randomly generated synthetic data. While synthetic, this setup allows for controlled, large-scale experimentation and reproducibility. Our experiments demonstrate that our ASP-based approach is able to efficiently handle large datasets of business rules, even when they contain thousands of conflicts.

7.1 Background on SBVR

SBVR [The Object Management Group, 2019], developed by the Object Management Group (OMG), is a standard designed to describe complex systems, such as businesses, in both a formal and natural way. It offers a structured method for defining business vocabulary and rules in a format that is both precise and easy to understand. The SBVR specification consist of two key components: vocabulary and rules. The vocabulary is a set of terms and definitions representing concepts, facts, and relationships within a business domain. More precisely, the vocabulary is made of nouns and verbs. Noun concepts can be classified as either general concepts or individual concepts. A general concept refers to a category that groups things based on shared properties, while an individual concept represents a specific, singular object. Verbs define relationships between two or more noun concepts or describe a characteristic of a noun concept. In the following, we present examples based on the EU-Rent SBVR specification, a car rental company provided by KDM Analytics[1]. EU-Rent operates in multiple countries, renting cars to customers through its branches. The specification includes vocabulary definitions and business rules governing its operations.

Example 1 (Vocabulary). A vocabulary is a sequence of elements of the form:

```
rental
requested car group
period
rental period
```

[1] https://www.kdmanalytics.com/sbvr/EU-Rent.html

```
    General Concept: period
full
    Concept type: individual concept
rental includes rental period
rental has requested car group
```

where `rental`, `requested car group`, `period`, `rental period` and `full` are concepts. Moreover, `rental period` specializes the `period` concept, while `full` is an individual concept. Instead, `includes` and `has` are verbs that define relationships: `includes` links `rental` to `rental period`, and `has` connects `rental` to `requested car group`.

Business rules, instead, are logical statements that define guidelines, constraints, or conditions that govern how a business operates. They define what can, must, or must not happen in a business process to ensure consistency, compliance, and efficiency. Business rules help enforce policies, regulations, and best practices within an organization.

Example 2 (Business rules). The following represent two examples of business rules:

```
It is necessary that each rental has exactly one requested car
    group.
It is necessary that each rental includes exactly one rental
    period.
```

It is possible to observe that the concepts introduced in the vocabulary are used in the business rules, together with SBVR keywords, to define constraints. In more details, SBVR allows facts and business rules to be expressed in various ways, including statements, diagrams, or a combination of both, depending on the intended purpose. One common method is through a Controlled Natural Language (CNL), i.e., a simplified subset of natural language (such as English) designed for clarity and consistency. Instead of the full complexity of natural language, a CNL employs a limited set of structures and common words to create a straightforward and structured representation of business knowledge.

We focus on SBVR Structured English (SBVR-SE), a CNL described in Annex A of the OMG SBVR specification [The Object Management Group, 2019], as SBVR-SE can be seen as an effective compromise for bridging the gap between business experts and information technology professionals.

Table 7.1 presents the main keywords of the SBVR-SE grammar, while a complete and formal description can be found in [The Object Management Group, 2019]. We have three different types of keywords: quantification operators that precede nouns, logical operations,

Table 7.1 SBVR-SE keywords. In the table, *n* and *m* represent numbers, whereas *p* and *q* denote expressions or propositions that allow to combine the different operators into sentences.

Quantification Operators	Logical Operations	Modal Operations
each	it is not the case that *p*	it is obligatory that *p*
some	*p* and *q*	it is prohibited that *p*
at least one	*p* or *q*	it is necessary that *p*
at least *n*	*p* or *q* but not both	it is impossible that *p*
at most one	if *p* then *q*	it is possible that *p*
at most *n*	*q* if *p*	it is permitted that *p*
exactly one	*p* if and only if *q*	*p* must *q*
exactly *n*	not both *p* and *q*	*p* must not *q*
at least *n* and at most *m*	neither *p* nor *q*	*p* need not *q*
more than one	*p* whether or not *q*	*p* can *q*
no		*p* may *q*

and modal operations. By combining nouns, verbs, and the operators listed in Table 7.1, SBVR rules can be formulated.

Example 3 (Modal and quantification operators). The following business rule uses the modal operator `It is obligatory that` and the quantification operators `each` and `exactly one`:

```
It is obligatory that each rental car is owned by exactly one
   branch.
```

In this statement, `rental car` and `branch` are nouns, and `is owned by` is a verb; these concepts are defined in the vocabulary.

Moreover, it is possible to use modal operators combined with the keyword `only if` to invert the modality.

Example 4 (Modal operators combined with **only if**). The following business rules have the same meaning:

```
A car may be rented only if the car is available.
A car must not be rented if the car is not available.
```

Additionally, SBVR-SE supports other keywords, such as `the`, `a`, `an`, used as quantification or as introduction of a name of an individual thing. The keyword `that` has different purposes depending on its position: *(i)* Before a designation for a noun concept, it is a binding

to a variable (similar to `the`); and *(ii)* After a designation for a noun concept and before a designation for a verb concept, it is used to introduce a restriction on things denoted by the designated entity as shown in the following example.

Example 5 (Usage of `that`). The following two business rules represent an example of the possible usage of the `that` keyword:

```
It is necessary that the scheduled pick-up date/time of each
   advance rental is after the booking date/time of the rental
   booking that establishes the advance rental.
```

Similarly, `who` has the same meaning as the second use of `that`, but it is specifically used to refer to persons.

Example 6 (Usage of `who`). In the following sentence, the `who` keyword is used to refer to the renter:

```
It is permitted that a rental is open only if an estimated rental
   charge is provisionally charged to a credit card of the renter
   who is responsible for the rental.
```

Additionally, `of` establishes a relationship between the two preceding and following nouns. The expression `p of q` is equivalent to `q has p`.

Example 7 (Usage of `of`). In the following, an example of a relation established by the `of` keyword:

```
If the renter of a rental requests a price conversion then it is
   obligatory that the rental charge of the rental is converted to
   the currency of the price conversion.
```

In this sentence, we have an implication introduced by the operators `If p then q` and the modal operator `it is obligatory that`. Additionally, the `of` operator is again used to express relationships between nouns.

7.2 Handling SBVR with ASP

In this section we describe our ASP-based approach and its implementation, SBVR2ASP, whose architecture is shown in Figure 7.1.

SBVR2ASP processes a SBVR specification written in SBVR-SE, which, as mentioned above, consists of two main elements: vocabulary and (business) rules, which are handled by

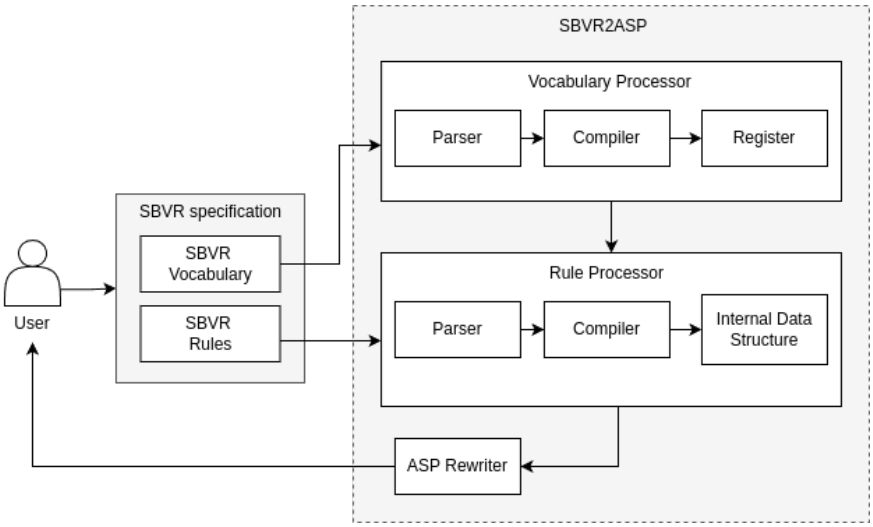


Figure 7.1 Architecture of SBVR2ASP.

the two main components of the tool, i.e., the Vocabulary Processor and the Rule Processor, respectively.

The Vocabulary Processor includes a Parser that tokenizes the input vocabulary and constructs an abstract syntax tree (AST). The AST is then processed bottom-up by the Compiler, which initializes a data structure, called Register, that tracks all entities of the specification and their relationships. In particular, this data structure saves the information retrieved from the vocabulary. Since SBVR nouns can be multi-word expressions of arbitrary length (e.g., `requested car group` from Example 1 is a single concept), the Parser alone cannot determine where a noun ends and a new token begins. Indeed, recognizing an undetermined sequence of characters can lead to ambiguity. To resolve this issue, the tool first assigns a unique identifier (ID) to every noun and verb in the Register. Then, these identifiers temporarily replace the corresponding terms within the SBVR specification ensuring that each concept is a single word during parsing, while still allowing the original expressions to be retrieved later using their assigned IDs.

Next, the Rule Processor parses the business rules using another Parser, constructing an AST. For example, Figure 7.2 shows a possible target AST for the following business rule:

```
It is necessary that the renter of each points rental is a club
    member.
```

However, as already mentioned, for the Parser, understanding that `points rental` is a single node is difficult; the parser could recognize, for example, `each points rental` or `renter of each points rental` as a single concept. Our solution is that each concept

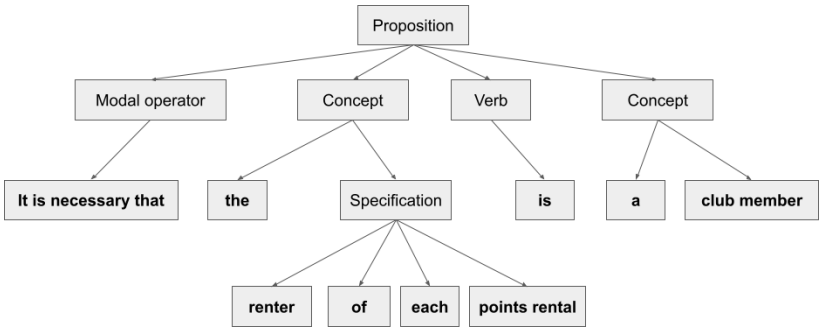


Figure 7.2 Example of an AST.

must be a single word; thus, `points rental` is temporarily replaced with its unique ID during parsing and then substituted back to its original form in the final AST. Once the AST is built, it is processed bottom-up. This step initializes an Internal Data Structure, which facilitates data manipulation by reconstructing and organizing relationships between entities. Finally, the processed data is passed to the ASP Rewriter, which converts the extracted business rules into ASP rules. In the following, we show the main type of sentences for business rules using the EU-Rent specification introduced in Section 7.1. Due to space constraints, we only include sentences that illustrate distinct SBVR-SE constructs (see https://github.com/simocaruso/sbvr2asp for the full specification).

7.2.1 Simple and complex sentences

Simple sentences are translated into ASP in a quite intuitive way, where each noun, i.e., the one declared in the vocabulary, is mapped to a predicate with a field serving as an identifier. On the other hand, verbs are represented as atoms of arity two, linking the corresponding nouns that define their relationship. As an example the following SBVR sentences:

```
It is necessary that each rental has exactly one requested car
   group.
It is necessary that each rental includes exactly one rental
   period.
It is necessary that each rental has exactly one return branch.
```

are translated into:

```
1  :- rental(REN), #count{requested_car_group(REQCARGRO):
      has(rental(REN),requested_car_group(REQCARGRO))} != 1.
```

```
2   :- rental(REN), #count{rental_period(RENPER):
       includes(rental(REN),rental_period(RENPER))} != 1.
3   :- rental(REN), #count{return_branch(RETBRA):
       has(rental(REN),return_branch(RETBRA))} != 1.
```

Note that the quantifier `each` does not require a direct translation in ASP due to grounding, while `exactly one` is represented with an aggregate. However, since the SBVR sentence begins with `It is necessary that` and ASP only supports integrity constraints, the condition of `exactly one` is flipped to be represented as *different from one*.

SBVR2ASP also supports more complex sentences that incorporate additional constructs, as the following one:

```
It is necessary that the scheduled pick-up date/time of each
   advance rental is after the booking date/time of the rental
   booking that establishes the advance rental.
```

As in the case of simple sentences, verbs and nouns are converted into atoms. Here, the keyword `after` is translated using the operator $<=$ due to the presence of `It is necessary that` (as it is part of an ASP constraint). Additionally, the keyword `of` establishes a relationship between two entities, e.g., between `rental date/time` and `rental booking`, though the relation itself is not explicitly named. To address this, the vocabulary is processed first, as explained in the tool architecture above, and the relevant information is stored in the Register. This allows us to dynamically retrieve the relationship and initialize the appropriate atom. Thus, the relationship:

```
the booking date/time of the rental booking
```

is translated into:

```
1   booking_date_time(BOODATTIM), rental_booking(RENBOO),
       has(rental_booking(RENBOO),booking_date_time(BOODATTIM))
```

Therefore, the translation of the whole sentence is the following:

```
1   :- scheduled_pick_up_date_time(SPUDT), booking_date_time(BDT),
       SPUDT <= BDT, advance_rental(ADR),
       has(advance_rental(ADR),scheduled_pick_up_date_time(SPUDT)),
       rental_booking(RB),
       has(rental_booking(RB),booking_date_time(BDT)),
       establishes(rental_booking(RB),advance_rental(ADR)).
```

It is important to observe that sentences may also contain conjunctions, which are translated into different ASP rules. As example, consider the following sentence:

> It is obligatory that at the actual return date/time of each in-country rental and each international inward rental the local area of the return branch of the rental owns the rented car of the rental.

This is translated into two different ASP rules:

```
1  :- in_country_rental(ICR), actual_return_date_time(ARDT),
   has(in_country_rental(ICR), actual_return_date_time(ARDT)),
   #count{rented_car(RENCAR):
   owns(local_area(LA),rented_car(RENCAR)), local_area(LA),
   rented_car(RENCAR), in_country_rental(ICR),
   has(in_country_rental(ICR),rented_car(RENCAR))} < 1,
   return_branch(RETBRA),
   has(in_country_rental(ICR),return_branch(RETBRA)),
   is_included_in(return_branch(RETBRA),local_area(LA)).

2  :- international_inward_rental(IIT),
   actual_return_date_time(ARDT),
   has(international_inward_rental(IIT),
   actual_return_date_time(ARDT)), #count{rented_car(RENCAR):
   owns(local_area(LA),rented_car(RENCAR)), local_area(LA),
   rented_car(RENCAR), international_inward_rental(IIT),
   has(international_inward_rental(IIT),rented_car(RENCAR))} < 1,
   return_branch(RETBRA),
   has(international_inward_rental(IIT),return_branch(RETBRA)),
   is_included_in(return_branch(RETBRA),local_area(LA)).
```

where the first rule handles `in-country rentals`, while the second one applies the same rule to `international inward rentals`.

7.2.2 Only if and implication

As mentioned before, `only if` is used as a logical connector with a very specific meaning, aligned with formal logic, as it corresponds to a necessary condition. The following example shows the usage of such a construct:

> `It is permitted that a rental is open only if an estimated rental charge is provisionally charged to a credit card of the renter that is responsible for the rental.`

This is translated as the following ASP rule:

```
:- rental(REN), open(OPE), REN = OPE, #count{credit_card(CC):
   is_provisionally_charged_to(estimated_rental_charge(ERC),
   credit_card(CC)), estimated_rental_charge(ERC),
   credit_card(CC), renter(RENT), has(renter(RENT),
   credit_card(CC)), is_responsible_for(renter(RENT),rental(REN))}
   < 1.
```

The resulting translation is that we cannot have a rental open if the second part of the sentence does not hold; that is, an estimated rental charge must be provisionally charged to a renter's credit card. The negation is handled by an aggregate in which it is ensured that at least one credit card provisionally charges the rental. On the other hand, a sentence of the form `if ... then ...` is an implication. An example of a sentence of such kind is the following:

> `If the renter of a rental requests a price conversion then it is obligatory that the rental charge of the rental is converted to the currency of the price conversion.`

The translation is similar to the previous case, and also here an aggregate ensures that there exists a currency to which the rental charge is converted. Therefore, the resulting ASP rule is the following:

```
:- renter(RENT), price_conversion(PRICON),
   requests(renter(RENT),price_conversion(PRICON)), rental(REN),
   has(rental(REN),renter(RENT)), #count{currency(CUR):
   is_converted_to(rental_charge(RENCHA),currency(CUR)),
   rental_charge(RENCHA), currency(CUR), price_conversion(PRICON),
   has(price_conversion(PRICON),currency(CUR))} < 1, rental(REN),
   has(rental(REN),rental_charge(RENCHA)).
```

7.2.3 Arithmetic operations and time constraints

SBVR allows the usage of arithmetic operations, as shown in the following example:

> It is necessary that the booking date/time of a points rental is at least 5 days before the scheduled start date/time of the rental.

where `at least 5 days before` is an arithmetic operation. This sentence is translated nicely into ASP by simply adding the number 5 to the booking date/time to enforce the required condition. The resulting rule is the following:

```
:- booking_date_time(BDT), scheduled_start_date_time(SSDT),
   points_rental(PR), BDT+5 >= SSDT,
   has(points_rental(PR),booking_date_time(BDT)),
   has(points_rental(PR),scheduled_start_date_time(SSDT)).
```

There are also sentences involving generic time constraints, as the following:

> It is obligatory that the start date of each reserved rental is in the future.

In this case, we consider `future` as being after a constant representing the current date/time, denoted as `now`. Therefore, the corresponding ASP rule is the following:

```
:- start_date(STADAT), STADAT <= now, reserved_rental(RESREN),
   has(reserved_rental(RESREN),start_date(STADAT)).
```

7.2.4 Values and labels

SBVR sentences may include specific values that are defined in the vocabulary. The following sentence contains the term `full`, which is declared in the vocabulary as a possible value for the fuel level:

> At the actual start date/time of each rental it is obligatory that the fuel level of the rented car of the rental is full.

In ASP, this is handled by using a constant named `full`. The resulting translation ensures that the fuel level cannot be anything other than full at the start of the rental, as follows:

```
:- rental(REN), actual_start_date_time(ASDT),
   has(rental(REN),actual_start_date_time(ASDT)),
   fuel_level(FUELEV), FUELEV != full, rental(REN),
```

Table 7.2 Average times (in seconds) for evaluating instances with no conflicts.

EU-Rent		Loan		Photo Equipment	
#Rentals	Time	#Loans	Time	#Photos	Time
500	0.29	100	0.01	10 – 100	0.01
5500	7.83	1 000	0.10	500 – 1 000	0.10
10 500	24.85	5 000	0.50	2 500 – 10 000	0.86
15 500	51.31	10 000	1.05	12 500 – 20 000	2.54
20 500	88.26	15 000	1.64	22 500 – 30 000	4.06
25 500	143.63	20 000	2.24		
30 500	169.14	25 000	3.03		
		30 000	3.50		

```
rented_car(RENCAR), has(rental(REN), rented_car(RENCAR)),
has(rented_car(RENCAR), fuel_level(FUELEV)).
```

Finally, SBVR allows for labels that distinguish different instances of the same concept, as shown in the following example:

```
If rental1 is not rental2 and the renter of rental1 is the renter
   of rental2 then it is obligatory that the rental period of
   rental1 does not overlap the rental period of rental2.
```

The following is translated into the following ASP rule:

```
1  :- rental(REN_1), rental(REN_2), REN_1 != REN_2, renter(RENT_1),
   renter(RENT_2), RENT_1 = RENT_2,
   has(rental(REN_1),renter(RENT_1)),
   has(rental(REN_2),renter(RENT_2)), rental_period(RENPER_1),
   rental_period(RENPER_2),
   overlap(rental_period(RENPER_1),rental_period(RENPER_2)),
   includes(rental(REN_1),rental_period(RENPER_1)),
   includes(rental(REN_2),rental_period(RENPER_2)).
```

It is important to observe that the two `rental` instances are assigned different variables, and they are explicitly treated as distinct (`REN_1 != REN_2`), ensuring that the constraint applies only when different rentals share the same renter.

Table 7.3 Average times (in seconds) for detecting conflicts.

Number of conflicts	EU-Rent	Loan	Photo Equipment
0 – 1 500	0.28	0.05	0.09
1 501 – 3 000	7.43	0.55	0.78
3 001 – 4 500	27.10	1.18	1.51
4 501 – 6 000	55.13	1.77	2.26
6 001 – 7 500	87.35	2.27	3.27
7 501 – 9 000	152.87	3.07	3.86
9 001 – 10 000	156.47	3.53	4.52

7.3 Experiments

The performance of SBVR2ASP has been empirically evaluated on three publicly available datasets containing a SBVR specification, including vocabularies and rules, namely EU-Rent [The Object Management Group, 2019], Loan, and Photo Equipment[2] [Karpovic et al., 2016, 2014]. The first one has been used as running example in this chapter and describes a fictional car rental company operating across multiple countries, renting vehicles to customers through its various branches. The second one includes a limited set of rules regarding interactions among debtors, banks, and loans. Finally, the Photo Equipment specification contains simple rules related to components of cameras.

As previously discussed, [Schumann and Gómez, 2024] identified several limitations in existing datasets. In particular, although they are publicly available, there are no real-world instances. Furthermore, the available synthetic datasets include only a limited number of conflicts (typically no more than a few hundred). Therefore, in designing our experiment, we generated synthetic data while deliberately increasing both the dataset size and the number of conflicts. In more details, for the EU-Rent dataset, we created instances with up to 30 500 rentals, varying the number of branches, countries, and customers. For the Loan dataset, we generated instances with up to 30 000 people and up to 20 different banks. Finally, for the Photo Equipment dataset, we modeled up to 30 000 cameras and related components. Concerning conflicts, we identified a set of potential inconsistencies within each dataset. Then, we assigned a probability (approx. 4%, as we observed it was a good empirical parameter to produce a sufficient number of conflicts) to each type of conflict, determining whether it would be activated and included in a given instance. This resulted in a number of conflicts up to 10 000 in the largest instances.

[2]Loan and Photo Equipment can be downloaded from https://s2o.isd.ktu.lt/about.php.

Each specification was translated into an ASP encoding using SBVR2ASP. We then used CLINGO (version 5.4.1) to check the satisfiability of each generated instance. All experiments were executed on a machine with an AMD Ryzen 7 5825U CPU @ 2.0GHz and 16 GB of RAM.

Table 7.2 reports the average solving times (in seconds) for instances without conflicts, across the three benchmark domains. CLINGO is able to process conflict-free instances efficiently, with solving times remaining low even as instance sizes grow significantly. In the EU-Rent setting, instances with up to 30 500 rentals are solved in less than 3 minutes. Moreover, in the Loan and Photo Equipment domain, even the largest tested instance (30 000 loans/photos) requires less than 5 seconds to be evaluated. Table 7.3 shows the average solving times for instances with conflicts, grouped by increasing conflict counts. As expected, solving times grow with the number of conflicts, but remain within reasonable limits across all domains. In the EU-Rent domain, the increase is more pronounced, with solving times reaching approximately 2.5 minutes for instances with up to 10 000 conflicts. Conversely, the Loan and Photo Equipment datasets exhibit lower absolute times: even at the highest conflict levels tested, CLINGO completes the analysis in under 5 seconds for Photo Equipment and under 4 seconds for Loan. It is important to note that the solving times reported for both conflict-free and conflicting instances are largely dominated by the grounding phase performed by CLINGO. In our encodings, the solving process itself is essentially trivial: the programs are fully deterministic and do not involve any non-deterministic choices. As a result, once grounding is complete, the actual computation of the answer set is immediate. The observed execution times, therefore, reflect the cost of instantiating the rules over the (potentially large) input datasets.

7.4 Conclusion

In this chapter, we proposed a novel approach based on ASP for processing SBVR specification. Our method directly addresses several limitations highlighted in a recent survey by [Schumann and Gómez, 2024], particularly those related to conflict detection in SBVR. To evaluate its effectiveness, we carried out an experimental analysis on synthetically generated datasets containing thousands of conflicts. The results demonstrate that our approach is well-suited for inconsistency checking in SBVR specification. Finally, the tool and all the material needed to reproduce the experiments are available at https://github.com/simocaruso/sbvr2asp.

Chapter 8

CNLWizard

Besides Answer Set Programming, there are other powerful computational paradigms widely used such as Constraint Programming (CP) [Rossi et al., 2006], and Satisfiability Modulo Theories (SMT) [Barrett et al., 2021], each offering distinct approaches to problem modelling and solving. CP focuses on representing a problem through variables, domains, and constraints. The goal is to find assignments to variables that satisfy all constraints, making CP particularly effective for problems with intricate combinatorial structures, such as scheduling, planning, and resource allocation [Wallace, 1996, Hooker and van Hoeve, 2018]. SMT builds upon the foundation of Boolean Satisfiability (SAT) by extending it to more expressive theories like arithmetic, bit-vectors, and arrays [de Moura and Bjørner, 2011]. SMT solvers determine the satisfiability of logical formulas within these theories, enabling efficient handling of problems in verification, model checking, and software synthesis, where mathematical precision is crucial. Despite their unique approaches, CP, SMT, and ASP share a common purpose: solving complex combinatorial problems. They can be seen as complementary formalisms within the same broader framework of declarative problem-solving, where each formalism excels in different problem domains, yet they can be integrated or used in tandem to leverage their respective strengths.

Integrating CNLs with ASP, CP, and SMT allows for a more natural and efficient approach to complex problem-solving, combining the precision of formal methods with the accessibility of natural language, broadening the applicability and impact of these powerful computational paradigms. Nevertheless, implementing custom CNLs is a time-consuming and challenging process. Indeed, from a technical perspective, creating a custom CNL involves writing the grammar, processing the Abstract Syntax Tree (AST), and generating code that translates the natural language input into the specific formalism. The first step requires defining a set of syntactic rules capable of capturing the nuances of the language while maintaining precision

and clarity. The grammar must be comprehensive enough to cover the wide variety of constructs that users may wish to express, yet restricted enough to avoid ambiguity. Once the CNL input is parsed according to the grammar, it must be converted into an AST. This stage is complex because it involves resolving ambiguities, managing scope, and ensuring compliance with all semantic rules of the CNL. The final step involves translating the AST into code that conforms to the specific formalism, whether it be ASP, CP, or SMT. This translation process is intricate, as it requires mapping high-level CNL constructs to the lower-level constructs of the formal language. In this chapter, we present a novel framework, called CNLWizard, designed to reduce the effort involved in developing custom CNL. It enables the user to specify a grammar for multiple target languages abstractly, automatically suggests a possible default implementation for some elements involved in the pipeline, and provides auxiliary data structures that make the construction of the CNL language more flexible and guided. CNLWizard processes inputs described in a simple YAML-based language and automatically generates the grammar for the CNL, along with a set of pre-implemented imperative functions that minimise boilerplate code. This allows developers to focus primarily on writing the specific code required to convert sentences into the target formalism, streamlining the process and making custom CNL development more accessible and efficient. Indeed, as a practical evaluation, we compared the lines of code required by CNLWizard with those required by CNL2ASP to define and convert CNL sentences into ASP rules, demonstrating that CNLWizard consistently requires significantly fewer lines of code than CNL2ASP.

8.1 Framework

We recall that a *grammar* [Chomsky, 1959] is a tuple (N, T, S, P), where N and T are disjoint sets of non-terminal and terminal symbols, respectively, $S \in N$ is the starting symbol, and P is a finite binary relation defined on $(N \cup T)^* \circ N \circ (N \cup T)^* \times (N \cup T)^*$, called *production rules*, where $*$ is the Kleene star operator [Hopcroft and Ullman, 1979]. According to the Chomsky Hierarchy, four types of grammar can be classified depending on the expressiveness allowed by P. For the purpose addressed in this work, we use *type 2* or *context-free grammars*, where P is restricted to relations defined on $N \times (N \cup T)^*$. Let F be a set containing KR formalisms. We represent with $fn : P, F \mapsto \mathscr{I}$ the function that maps a production rule $p \in P$ and a KR formalism $k \in F$ into an imperative function invoked when applying p in the Abstract Syntax Tree (AST), where an element in $\mathscr{I}$ is a triple $(name, \vec{args}, code)$ containing, respectively, the function's name, arguments and implementation. In other words, the imperative function defined through *fn* specifies how CNL's text matching a production

rule is "processed" to obtain statement(s) in a target KR formalism or to define the structure of the entities described.

Let $F_{Tar} \subset F$ represent a set of target KR formalisms for which a programmer aims to define a CNL grammar (and the corresponding *fn*). Our framework provides a high-level language to compactly define the CNL grammar $G_t = (N_t, T_t, S_t, P_t)$ for each formalism $t \in F_{Tar}$, together with an initialization of $fn(p,t)$ for each production rule $p \in P_t$. For the sake of simplicity, we will define how each set of production rule P_t is built and assume that N_t and T_t are implicitly derived by projecting all the non-terminal and terminal symbols occurring in P_t. Without loss of generality, the starting symbol of each grammar, S_t, can be defined as a default non-terminal symbol, S.

Our framework consists of a function $\phi : \mathscr{C} \mapsto 2^{(P,\mathscr{I},F)}$ that maps a command $c \in \mathscr{C}$ into a set of triples defining a production rule and corresponding imperative function for a formalism. Given a set of commands $\hat{\mathscr{C}} \subseteq \mathscr{C}$, we define every P_t for $t \in F_{Tar}$ as $P_t = \bigcup_{c \in \hat{\mathscr{C}}} \{p \mid (p,I,t) \in \phi(c)\}$. Then, for each $c \in \hat{\mathscr{C}}$ and each $(p,I,t) \in \phi(c)$, we initialize $fn(p,t) = I$. A command is a pair where the first argument is a keyword (such as `syntax`, `concat`, etc.) or a non-terminal symbol N and the second argument can be any element of a grammar, formalisms or a set of commands. In the following, we are going to define the set of accepted command $\mathscr{C}$, together with the definition of ϕ for each of them.

A possible command is (n, Par), where $n \in N$ and Par is a set of commands with exactly one occurrence of:

- $(\texttt{syntax}, ST)$, where $ST \subseteq (N \cup T)^*$; and
- $(\texttt{target}, TG)$, where $TG \subseteq F_{Tar}$;

and at most one occurrence of the command:

- $(\texttt{concat}, CN)$, where $CN \in (N \cup T)$.

Then, the function $\phi((n, Par))$ returns:

$$\{(n \times ST, (n, ST \cap N, \emptyset), t) \mid t \in TG\} \cup \tag{8.1}$$

$$\begin{aligned} &\{(n \times n \circ CN \circ n, \\ &(n\_\texttt{concat}, (ST \cap N)^*, \texttt{code\_con}), t) \mid \\ &t \in TG, (\texttt{concat}, CN) \in Par\}. \end{aligned} \tag{8.2}$$

This command is used to define a production rule for each target formalism and automatically bind it with the corresponding imperative function, as shown in (8.1). Notice that the

code part is left empty, since the implementation depends on the specific application. The optional command $(\texttt{concat}, CN)$ specifies an additional production rule for n, where its right-hand side consists of multiple applications of ST, concatenated by the symbol CN. The corresponding imperative function is named n_concat and has a single argument consisting of the list of non-terminal symbols that appeared in the rule, while its implementation is already provided by the framework and called code_con, as shown in (8.2).

A mandatory command is $(\texttt{start}, Par)$, where Par is a set of commands defined as in the previous item. This command is used to specify the production rule of the starting symbol for each target formalism. Indeed, $\phi((\texttt{start}, Par))$ returns the sets in (8.1) and (8.2), where n is replaced by the default starting symbol S.

Another possible command is $(\texttt{operation}, Par)$, where Par is a set of commands with exactly one occurrence of:

- $(\texttt{name}, Op)$, where $Op \in N$;
- $(\texttt{syntax}, ST)$, where $ST \subseteq (N \cup T)^*$ and $Op\_\texttt{oper} \in ST$;
- $(\texttt{target}, TG)$, where $TG \subseteq F_{Tar}$; and
- $(\texttt{operators}, (s, f_s))$, where $s \in T$ represents an operator for the operation Op, and f_s defines how to translate s when encountering its node in the AST;

and at most one occurrence of the command:

- $(\texttt{concat}, CN)$, where $CN \in (N \cup T)$.

Then, the function $\phi(\texttt{operation}, Par)$ returns:

$$\{(Op \times ST, (Op, ST \cap N, \texttt{code\_op}), t) \mid t \in TG\} \tag{8.3}$$

$$\cup \{(Op\_\texttt{oper} \times s, (Op\_\texttt{oper}, s, f_s), t) \mid t \in TG\} \tag{8.4}$$

$$\begin{aligned} \cup \{&(Op \times Op \circ CN \circ Op, \\ &(Op\_\texttt{concat}, (ST \cap N)^*, \texttt{code\_con}, t) \mid \\ &t \in TG, (\texttt{concat}, CN) \in Par\}. \end{aligned} \tag{8.5}$$

The command $(\texttt{operation}, Par)$ simplifies the definition of operations (e.g., mathematical, comparison, etc.), where the symbol Op occurring with the keyword name represents a class of operations that includes all the operator's symbols defined after the keyword operators. The right-hand side of the production rule for Op, namely ST, must contain a special

keyword, called $Op\_\texttt{oper}$, that indicates the position in the sentence of the string describing an operator. The framework then defines ϕ to return: a tuple for each $t \in TG$ containing the production rule of Op, where the implementation of the corresponding function is already provided by the framework and called `code_op`, as shown in (8.3); a tuple containing a production rule that maps $Op\_\texttt{oper}$ to each operation s, where the implementation of the corresponding function is defined by f_s, as shown in (8.4); and, lastly, the tuples obtained from the concatenation, as shown in (8.5).

Similarly to library usage for programming languages, our framework imports common patterns occurring for standard KR formalisms. Let $F_{Aux} \subset F$ represent a set of KR formalisms. For each $s \in F_{Aux}$ we assume the presence of an auxiliary context-free production rules P^s_{Aux}, where for each auxiliary rule $p \in P^s_{Aux}$ the function $fn(p,s)$ is pre-defined.

Another command is $(\texttt{import}, Par)$, where Par is a set of commands with exactly one occurrence of:

- $(\texttt{source}, s)$, where $s \in F_{Aux}$; and
- $(\texttt{target}, t)$, where $t \in F_{Tar}$;

and one or more occurrences of the command:

- $(\texttt{rules}, name(p))$, where $name(p)$ is a label identifying a production rule p.

This command can be used to import auxiliary production rules and pre-defined imperative functions. Namely, $\phi((\texttt{import}, Par)) = \{(p, fn(p,s), t) \mid (\texttt{source}, s) \in Par, (\texttt{target}, t) \in Par, (\texttt{rules}, name(p)) \in Par, p \in P^s_{Aux}\}$.

Lastly, to extend the flexibility of the grammars, our framework provides a set of data structures characterizing elements that frequently occur in KR formalisms. By activating them, the user can access auxiliary production rules. In particular, we consider: *(i) Signatures* that provide a template to define concepts, which intuitively can be traced back to types in imperative programming languages; for example, they can be used to define the structure of an atom in ASP, a variable in CP, and a literal/variable in SAT and SMT. A signature σ is composed of an entity, which includes a name and attributes; *(ii) Variables* that provide a template to define a variable that can range over different values. For example, one can state a variable X is between 0 and 10, and then our framework is able to recognise that X can range over the values and substitute it with its possible values in all of its occurrences.

8.1.1 Instantiation of the Framework

This section shows how the framework is instantiated for defining grammars that can parse CNLs tailored to the most common KR formalisms. F_{Aux} is the set {ASP, CP, SMT} and P_{Aux} is a set containing the following elements (for brevity, we show only the most important production rules), where non-terminal symbols are highlighted in bold and $\vee$ represents an or between elements, elements followed by ? are optional, **string** is a placeholder for a production rule matching any non-empty string containing letters, numbers and symbol _, and **number** is a placeholder for a production rule matching any number:

- (**signature**, (A $\vee$ An) **string** has (a $\vee$ an) **string** ((, $\vee$ and) (a $\vee$ an) **string**)*);
- (**variable**, where **string** is between **number** and **number**);
- (**there_is_clause**, (there $\vee$ There) is **entity**);
- (**positive_constraint**, It is required that **positive_constraint_body**);
- (**negative_constraint**, It is prohibited that **negative_constraint_body**);
- (**math**, (The $\vee$ the) **math_operator** between **math_first** and **math_second**);
- (**comparison**, **comparison_first** is? **comparison_operator comparison_second**);
- (**formula**, **formula_first formula_operator formula_second**);
- (**entity**, (a $\vee$ an)? **string attribute**?);
- (**verb**, (a $\vee$ an)? **string attribute**? **string**);
- (**attribute**, with **string** equal to (**string** $\vee$ **number**));
- (**simple_prop**, **entity** (have $\vee$ has $\vee$ are $\vee$ is) **verb entity**);
- (**neg_simple_prop**, **entity** (do not have $\vee$ does not have $\vee$ are not $\vee$ is not) **verb entity**);
- (**consequence**, If (**simple_prop** $\vee$ **neg_simple_prop**)

  then (**simple_prop** $\vee$ **neg_simple_prop**)).

These elements are provided by the framework and can be imported by users who want to extend the framework with new elements. In the example below, we demonstrate a simple extension of the framework.

Example 8. In the following, we assume the user wants to create a CNL that contains only sentences of the form:

- An **entity** has an **attribute** and a **attribute**.
- There is a **entity** with **attribute** equal to **value**, and with **attribute** equal to **value**.
- When there is a **entity** with **attribute** equal to **value** and **attribute** equal to **value**, then there is a **entity** with **attribute** equal to **value** and **attribute** equal to **value**.

where **entity** is an element in the signature, **attribute** is part of the entity, and **value** is either a string or a number.

To this end, the user has to define the following set of commands, with ASP as a target:

```
(import, {
    (source, asp), (target, asp),
    (rules, signature);
    (rules, there_is_clause)});
(start, {
    (syntax, signature), (target, {asp}),
    (concat, .)});
(start, {
    (syntax, there_is_clause),
    (target, {asp}), (concat, .)});
(start, {
    (syntax, rule),
    (target, {asp}), (concat, .)});
(rule, {
    (syntax, When there_is_clause, then there_is_clause), (target, {
       asp})}).
```

The following CNL sentences are thus parsed by the resulting grammar:

```
A user has an id and a name.
An admin has an id and a name.
There is a user with id equal to 1, and with name equal to john.
There is a user with id equal to 2, and with name equal to susan.
When there is a user with id equal to 2, and with name equal to
   susan, then there is an admin with id equal to 2, and with name
   equal to susan.
```

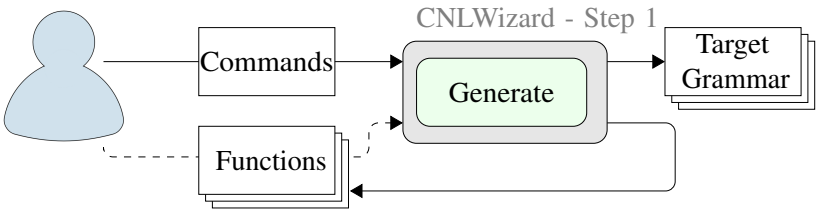


Figure 8.1 Generate Target Grammar and Functions.

8.2 CNLWizard

The framework described in the previous section has been implemented as part of a practical tool called CNLWizard, which operates through two main computational steps. The former corresponds to the framework, while the latter is responsible for the actual translation of a CNL file into a target language representation. For the first step, our system expects a YAML file (see https://yaml.org/ for the syntax of YAML) specifying the set of commands $\hat{\mathscr{C}}$, where terminal symbols are identified as double-quoted strings and non-quoted strings are either non-terminal symbols or keywords; then, CNLWizard generates a grammar and a Python file containing the corresponding imperative functions, for each target language t specified in $\hat{\mathscr{C}}$ through the keyword `target`. In the second step, CNLWizard applies the generated grammar to parse a CNL file, automatically producing an AST, and then it obtains the translated CNL by evaluating each node of the tree with the corresponding function generated in the previous step.

8.2.1 Step 1) Generate Target Grammar and Functions

Figure 8.1 depicts the first step of CNLWizard. The dotted arrow states that the user can optionally provide in input a file containing some of the auxiliary functions. If this is the case, instead of producing a new file containing all the function templates, the system just appends the missing functions.

In the following, we describe the YAML file expressing the set of commands $\hat{\mathscr{C}}$, where keywords are highlighted in bold, optional elements are enclosed in square brackets, and round brackets followed by the symbol plus indicate that multiple elements can be listed. The structure of the commands follows the template defined in the framework but relies on syntactic sugar that group commands with the same keywords, in order to further reduce the size of the file. The YAML file accepts commands of the form:

```
import:
```

```
    rules: (rule_name)+
    source: (source_lang)+
    target: (target_lang)+
```

where each `rule_name` is a reserved word identifying an auxiliary production rule provided by CNLWizard, for each formalism listed in `source` (at the moment, ASP, CP, and SMT). The imported rule is added to the grammar of the languages specified by `target`. The lists in `source` and `target` must have the same length to have a pair-wise match that reduces the length of the specification for homonymous rules. For each non-terminal symbol defined by the user, `non_term_sym`, the following entry is expected:

```
non_term_sym:
    syntax: (regex)+
   [target: (target_lang)+]
   [concat: conc_symbol]
```

where `regex` is a regular expression of terminal and non-terminal symbols, `target_lang` is the name of a KR formalism for which CNLWizard will generate the grammar and imperative functions, and `conc_symbol` is either a terminal or non-terminal symbol. If `target` is absent, then this rule is produced for every language specified with the starting symbol. Let us note that after `syntax` it is allowed to have a list of regular expressions. This is CNLWizard's syntactic sugar that compactly expresses groups of commands for the same non-terminal symbol. Additionally, CNLWizard allows unifying the definition of multiple rules, while being able to recall each of them in other regular expressions through the identifier `rule_name` specified after the keyword `name`:

```
non_term_sym:
  ( name: rule_name
    syntax: (regex)+
   [target: (target_lang)+]
   [concat: conc_symbol]
  )+
```

Then, we can have commands defining the starting symbol:

```
start:
    syntax: regex
    target: (target_lang)+
   [concat: conc_symbol]
```

where its structure matches the one used in the framework. The specification of the language operators is obtained with:

```
operation:
  ( name: op_name
    operators: (sym : op_repr)+
    syntax: (regex)+
   [target: (target_lang)+]
   [concat: conc_symbol]
  )+
```

This command is similar to the one for non-terminal symbols but additionally requires the presence of the non-terminal symbol `op_name_operator` in `regex`. After the keyword `operators`, it is expected a list of mappings from the operation name used in the CNL file, `sym`, to its representation in the target language, `op_repr`. In `op_repr`, it is possible to have either a terminal symbol or a function defined as:

```
fun:
    name: function_name
    args: (argument)+
```

allowing for distinguishing the function name and list of arguments and making them easily accessible in the function implementation.

Lastly, together with the formalism for ASP, CP and SMT, CNLWizard provides further auxiliary grammar elements and relative implementation. Indeed, it provides the definition of the most common regular expressions, such as `string` and `number`, that can be used in the entries of `syntax`. Moreover, through the following statements:

```
non_term_aux:
  ( syntax: regex
   [target: (lang)+]
   [concat: token]
  )+
```

the system allows overwriting the syntax of the auxiliary non-terminal symbols appearing in the imported rules.

8.2.2 Example of YAML and Python Functions

In this section, we describe a use case to demonstrate the usefulness of CNLWizard. Specifically, let us assume that the user wants to solve the following (simplified) scheduling problem: Given three natural numbers (namely, p_1, p_2, and b), a set of teams, and a set of employees, where each employee has a salary, the goal is to assign employees to teams such that the following conditions are met: *(i)* each employee is assigned to exactly one team; *(ii)* each team has at least p_1 employees and at most p_2 employees; *(iii)* the total salaries of the employees on each team must not exceed the given budget b. Moreover, let us assume (s)he wants to use the following CNL to describe an instance of the problem with 3 employees, 2 teams, p_1, p_2, and b set to 1, 2, and 5000, respectively:

```
An employee has a name and a salary.
A team has an id.
Assigned has an employee_name and a team_id.
There is an employee with name equal to 1, with salary equal to 2000.
There is an employee with name equal to 2, with salary equal to 3000.
There is an employee with name equal to 3, with salary equal to 2000.
There is a team with id equal to X, where X is between 1 and 2.
Employee with name equal to X is assigned to a team with id equal to
   1 or with id equal to 2, where X is between 1 and 3.
The number of employees assigned to a team with id equal to X is
   greater than or equal to 1, where X is between 1 and 2.
The number of employees assigned to a team with id equal to X is less
   than 3, where X is between 1 and 2.
The sum between the salary of the employees that are assigned to a
   team with id equal to X is less than 5000, where X is between 1
   and 2.
```

Listing 8.2.1 Example of simplified scheduling problem.

where sentences in lines 1–3 defines the concepts of employee, team, and assigned, respectively. Then, sentences in lines 4–6 define the employees and their salaries, and sentence at line 7 defines the two teams. Finally, sentences in lines 8–11 ensure that all conditions are met.

In this case, the YAML file reported in Listing 8.2.2 defines a set of commands for generating the grammar and the code needed by the aforementioned CNL, where we consider all the three different formalisms (ASP, CP, and SMT) as target.

```
import:
   rules: [attribute, there_is_clause, math, comparison,
      simple_proposition, negated_simple_proposition, entity, verb,
      variable, signature]
   source: [asp, cp, smt]
   target: [asp, cp, smt]
start:
   syntax: (proposition ".")*
   target: [asp, cp, smt]
proposition:
 - name: disjunction
     syntax: simple_clause "or" attribute
     concat: ", and"
 - name: there_is_clause
 - name: comparison
simple_clause:
   syntax: [ simple_proposition, negated_simple_proposition]
comparison_first:
   syntax: [math, aggregate]
aggregate:
   syntax: '"The number of" entity verb entity'
comparison_second:
   syntax: number
math_first:
   syntax: '"the" string "of the" entity "that are" verb entity'
```

Listing 8.2.2 YAML Specification.

The specification begins with the keyword `import`, which adds to the grammar the auxiliary production rules identified by `attribute`, `there_is_clause`, `math`, `comparison`, `simple_proposition`, `negated_simple_proposition`, `entity`, `verb`, `variable`, and `signature`. Then, the `start` keyword represents the root node of the AST, and its syntax is made of the concatenation of `proposition`. Here, the list of target formalisms, namely

asp, cp, and smt, is also defined. Following, we define the syntax of proposition, which can be a there_is_clause, a comparison, or a disjunction (which also presents the syntax and the concat keywords). In the following, other rules are defined using the same format, nevertheless, notice that some non-terminal symbols, e.g. comparison_first, comparison_second and math_first, must be defined because of the imported rules, comparison and math.

Once the specification is defined, the tool generates the two files for each target language: the grammar, and the functions. An example of an unimplemented function generated by the tool is shown in Listing 8.2.3, where the function start raises a NotImplementedError meaning that it must be implemented by the user.

```
def start(propositions):
    raise NotImplementedError
```

Listing 8.2.3 Example of a generated unimplemented Python function.

Other rules, instead, as for example the operation rule, have a default implementation as shown in Listing 8.2.4. In this, case the math function returns the joined string of the operator with the operands.

```
def math(*args):
    operator_index = 0
    operator = args[operator_index]
    args = list(args)
    args.pop(operator_index)
    return operator.join(map(str, args))
```

Listing 8.2.4 Example of a generated implemented Python function.

After the generation step, the user should implement the functions with empty code, allowing CNLWizard to invoke them during the translation step. By providing the structure for each function, we facilitate the user's task of implementing how each parsed element is mapped into the corresponding expression in the target representation language. As the functions are defined in an imperative language, it is possible to rely on external libraries for the implementation and evaluation of each expression.

Moreover, a user can define a proposition that is valid only for one target formalism, e.g. ASP. This can be done by adding the following line under proposition:

```
- name: weak_constraint
```

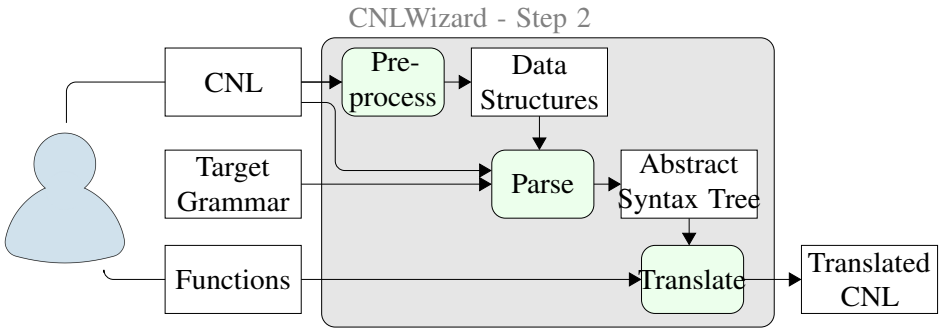


Figure 8.2 CNL translation step.

```
syntax: '"It is preferred as much as possible, that" comparison'
target: [asp]
```

Listing 8.2.5 Addition to Listing 8.2.2 to target only one formalism.

which adds an optimization statement only for the target `asp`.

8.2.3 Step 2) Translate CNL into Target Representation

Once the target grammar(s) and corresponding functions have been generated and implemented by the user, we can proceed with the translation step, illustrated in Figure 8.2. First, CNLWizard pre-processes the input CNL. In this phase, it initializes the auxiliary data structures, i.e. signatures, if imported, and substitutes the variables. The signatures support the writing of the grammar and translation into the target formalism. As mentioned above, they define the concepts of the problem and their structure, and then one can easily recall them with their defined structure in the propositions. On the other hand, variable substitution allows the straightforward application of the same statement with different values. After the pre-processing, CNLWizard applies the generated grammar to parse the processed CNL and obtains an AST, which is evaluated according to the corresponding function, obtaining the corresponding CNL translated in the target representation. As an example, consider the sentence `There is an employee with name equal to 1, with salary equal to 2000.` reported in Section 8.2.2 and consider the ASP target. This sentence is automatically parsed by the generated grammar and the following Python function is automatically created:

```
def there_is_clause(entity):
```

Table 8.1 Comparison of the lines of code needed to define CNLs using CNLWizard and CNL2ASP.

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

```
    return Fact(entity)
```

Listing 8.2.6 Generated function for `there_is_clause` sentence.

where `Fact` is the name of a Python class implemented in CNLWizard which translates the `entity` in the ASP fact `employee(1, 2000)`. Note that `entity` is also a data structure that is automatically created by CNLWizard and can be modified by accessing its internal fields, as follows:

```
def there_is_clause(entity):
    if 'salary' in entity.fields:
        entity.fields['salary'] += 'USD'
    return Fact(entity)
```

Listing 8.2.7 Modified function for `there_is_clause` sentence.

In this case, CNLWizard creates the ASP fact `employee(1, "2000USD")`. All the examples and data for the target formalisms are available at https://github.com/dodaro/CNLWizard/tree/main/examples.

8.3 Implementation and Experiments

CNLWizard has been implemented in Python and released under an open-source licence [Caruso et al., 2025b]. The target grammar is obtained and expressed using the library lark [Lark], while the file containing the templates for the functions is expressed in Python. For each target language, CNLWizard uses state-of-the-art tools, such as clingo [Gebser et al., 2016] for ASP, OR-tools [Perron et al., 2023] for CP, and Z3 [de Moura and Bjørner, 2008] for SMT. Moreover, it includes several error messages designed to identify misspelled text, and common user errors. As for the evaluation of the performance, as CNLWizard is a tool to define novel CNLs, we measured how it can help developers reduce their development time. This is often measured in Software Engineering as the number of lines of code needed to solve

a problem (see, e.g. [Nguyen et al., 2007] for a discussion about pros and cons). In our case, the problem is the generation of a grammar for a CNL and its translation to a KR formalism. Specifically, we compared the required lines of code (both grammar and python code) to implement the translations from a CNL into ASP using CNLWizard with the ones required by CNL2ASP. Concerning the problem specifications, we used the domains available from the CNL2ASP repository (available at https://github.com/dodaro/cnl2asp/tree/main/examples), namely Chemotherapy Treatment Scheduling (CTS), Graph Coloring (GC), Manipulation of Articulated Objects (MAO), and Nurse Scheduling (NSP). We also mention that CNL2ASP is a versatile tool with many constructs and functions and a total of about 6 thousand lines of code. Therefore, for a fair evaluation of the lines of codes, for CNL2ASP we only considered the lines needed for parsing and translating the specific problem. The results are presented in Table 8.1, where the column labeled “All" indicates the total lines of code required to support the CNL sentences across all the considered domains. The main advantages of CNLWizard consist of a compact way of describing the grammar of the CNLs, a quick and easy way to import parts of grammar, and import/generate Python functions, and internal management of features such as signature, concatenation, and variable templates. As a result, CNLWizard consistently generated a corresponding CNL with significantly fewer lines of code, up to 10 times less than CNL2ASP.

8.4 Conclusions

In this chapter, we introduced a novel framework and its implementation, CNLWizard, which simplifies the creation of CNLs for KR formalisms such as ASP, CP, and SMT. The framework also facilitates the development of CNLs for other languages by automating much of the CNL creation process. Using a YAML-based input language, CNLWizard streamlines grammar generation and the creation of imperative functions, reducing the need for extensive boilerplate code.

Part IV

Conclusions

Chapter 9

Related Work

9.1 Planning and Scheduling in Healthcare

Answer Set Programming has been successfully used for solving hard combinatorial and application scheduling problems in several research areas. In the Healthcare domain (see, e.g., [Alviano et al., 2020] for a recent survey), the first solved problem was the *Nurse Scheduling Problem* [Alviano et al., 2017b, Dodaro and Maratea, 2017, Alviano et al., 2018], where the goal is to create a scheduling for nurses working in hospital units. Then, the problem of assigning operating rooms to patients, denoted as *Operating Room Scheduling* [Dodaro et al., 2018, 2019b], has been treated, and further extended to include bed management [Dodaro et al., 2019a]. More recent problems include the *Chemotherapy Treatment Scheduling* problem [Dodaro et al., 2021], in which patients are assigned a chair or a bed for their treatments, and the *Rehabilitation Scheduling Problem* [Cardellini et al., 2021], which assigns patients to operators in rehabilitation sessions. In [Cappanera et al., 2022], it is proposed a solution using ASP to the problem of assigning a date to visit or therapy for multiple recurrent exams to chronic patients. The problem is split into two sub-problems to increase the performance of the solution using the Benders's decomposition method.

Concerning scheduling problems beyond the Healthcare domain, ASP encoding were proposed for the following problems: *Incremental Scheduling Problem* [Balduccini, 2011], where the goal is to assign jobs to devices such that their executions do not overlap one another; *Team Building Problem* [Ricca et al., 2012], where the goal is to allocate the available personnel of a seaport for serving the incoming ships; and the *Conference Paper Assignment Problem* [Amendola et al., 2016], which deals with the problem of assigning reviewers in the Program Committee to submitted conference papers. Other relevant papers are [Gebser et al., 2018b], where, in the context of routing driverless transport vehicles, the

setup problem of routes such that a collection of transport tasks is accomplished in case of multiple vehicles sharing the same operation area is solved via ASP, in the context of car assembly at Mercedes-Benz Ludwigsfelde GmbH, and the recent survey paper by [Falkner et al., 2018], where industrial applications dealt with ASP are presented, including those involving scheduling problems.

Recent works, are also exploring the combination of the learning capabilities of neural networks with the interpretability and formal reasoning of symbolic approaches leading to the neuro-symbolic AI [Sheth and Roy, 2024].

In this context, [Bruno et al., 2025] proposed an enhanced solution to the *Operating Room Scheduling* problem [Dodaro et al., 2018]. By using ML to estimate the surgery durations and ASP for scheduling, they demonstrated the potential of this technique on real-world data. Beyond scheduling, neuro-symbolic approaches have been applied also in other healthcare tasks. For example, [Bruno et al., 2022] defined a framework to represent and solve explicit knowledge via ASP; this approach guides neural network decisions and refines their output to provide explanations. The framework was successfully applied on semantic segmentation tasks using two datasets of biomedical images.

Further research, explore planning problems in Healthcare. [Alon et al., 2024, Alaboud and Coles, 2019] use PDDL+ to define a personalized medication plan, involving the selection, dosage, and administration schedule of medications, to achieve medical goals that are specific to the patient. [Michalowski et al., 2021] introduced MitPlan, a framework that addresses adverse interactions in multi-morbid patients by casting the reconciliation of concurrent Clinical Practice Guidelines as an automated planning problem using PDDL. Finally, [Fdez-Olivares et al., 2019] proposed a Multi-Agent Planning framework to conciliate single-disease guidelines by resolving interactions (e.g., drug or scheduling conflicts) via automated planning. In this approach, individual agents encapsulate single-disease guidelines as Hierarchical Task Networks domains and conciliate decision procedures to resolve adverse interactions, such as drug conflicts, scheduling constraints, and redundant actions, via an automated planning process. To manage potentially contradictory decisions arising from patient preferences, the system employs a Multi-Criteria Decision Making process to select the optimal personalized treatment plan. The appraoch was validated through a case study on Diabetes Mellitus and Arterial Hypertension.

9.2 Solving the Pre-Operative Assessment Clinic scheduling problem

[Edward et al., 2008a] used two simulation models to analyse the difficulties of planning in the context of PAC and to determine the resources needed to reduce waiting times and long access times. The models were tested in a large university hospital and the results were validated by measuring the level of patients's satisfaction. [Stark et al., 2015] used a Lean quality improvement process changing the process and the standard routine. For example, patients were not asked to move from one room to another for the visits, but patients were placed in a room, and remained there for the whole duration of their assessment. This and other changes to the processes led to a decrease in the average lead time for patients and to the number of patients required to return the next day to complete the visits. The authors of [Harnett et al., 2010, Ferschl et al., 2005, Correll et al., 2006, Tariq et al., 2016, Woodrum et al., 2017] studied the importance of implementing the PAC problem and the positive results obtained by having less waiting time between the exams and for the visit to the hospital. In particular, while different clinics follow different guidelines, implementing PAC has proved to be an important tool to avoid the cancellation of surgeries and to significantly reduce the risk associated with the surgery.

Differently from the works above mentioned, the solution we proposed focuses on an actual scheduling of the PAC with real requirements, making a practical system able to solve the PAC scheduling and rescheduling problems. Moreover, our solution guarantees a certain level of usability; in fact, through the two-phase approach and the web application that we developed, we can support the users in computing the scheduling.

As it can be seen, all works mentioned are about the scheduling problem. To the best of our knowledge, there is no paper dealing with the rescheduling PAC problem: Our work is thus the first one which proposes and implements a rescheduling PAC problem and related solution.

9.3 Scheduling Periodic Treatments problem

In [Marynissen and Demeulemeester, 2019] a review of the literature on multi-appointment scheduling problems in hospitals, i.e., problems in which patients need appointments for different resources, is presented and shows how research interest in these problems has grown only in recent years.

Problems similar to the one presented in this thesis are found only in particular use cases

such as rehabilitation, chemotherapy and radiotherapy. In [Condotta and Shakhlevich, 2014] a solution to the problem of chemotherapy treatments is proposed. Different treatments are considered for each patient but the days between one appointment and the following are fixed. Deciding the starting day, therefore, also fixes all the subsequent visits. The problem is modeled as an integer linear program and tests were conducted using CPLEX solver.
In [Conforti et al., 2008], a model for scheduling radiotherapy patients is presented. In this problem a certain number of weekly treatments must be scheduled for each patient. An Integer program formulation tested with LINGO is proposed.
Several studies have also been conducted in the field of physical therapy and different models have been proposed as in [Schimmelpfeng et al., 2012] and [Braaksma et al., 2014]. A series of appointments are required in this problem that also involve different resources and disciplines.
However, there are no general solutions for these kind of problems as the one presented in our work that is designed to be flexible and able to adapt to different contexts where different appointments, involving different operators and resources, are required with a certain frequency and in a time horizon of several weeks.

9.4 Controlled Natural Languges for Logic Programming

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 (CPL), 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].

[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 5.2.1), whereas Puzzle clues can be seen as the logic rules to solve the puzzle. 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 similar to our CNL2ASP, that is automatically converted into ASP.

9.5 LLMs and Logic Programming

Ongoing research is exploring the usage of LLMs to translate natural language into ASP. Indeed, recent advances in LLMs have established them as powerfull tools across a wide range of natural language tasks [Raiaan et al., 2024, Minaee et al., 2024]. In particular, in the field of automatic text-to-code translation, LLMs are used, often successfully, to generate programs from natural language input. This capability stems from extensive training on massive datasets comprising code repositories, technical forums, coding platforms, and documentation, effectively enabling LLMs to translate human intent into functional code. Furthermore, this also enables LLMs to support development in a broader sense: from synthesizing new code based on user intent to identifying bugs and providing natural language explanations for complex code segments. However, despite the vast availability of imperative code in their training sets, LLMs still struggle to generate consistently reliable code. This challenge is significantly exacerbated for KR formalisms, which are far less represented in public datasets. This scarcity, combined with the inherent stochasticity of LLMs, fundamentally undermines the precision and reliability required in knowledge representation tasks, especially in critical contexts such as business rule consistency checking. KR languages like ASP are also highly sensitive to syntax and semantics, meaning that even minor errors (common in LLMs) can lead to unusable programs, or, even worse, incorrect programs. To address these challenges, recent studies have empirically evaluated the capacity of LLMs to generate ASP code. [Yang et al., 2023] suggested that LLMs such as GPT-3 can function as few-shot semantic parsers, transforming natural language into logical forms

for ASP. However, as pointed out by the authors, some results are still unpredictable, and the LLM does not behave as intended. Other works focus on specific tasks, as [Alviano and Grillo, 2024], which translates NL sentences into ASP facts, or [Coppolillo et al., 2024], which supports some simple patterns. Thus, despite their promise, LLMs are not yet capable of reliably producing arbitrary ASP programs. CNLs instead remain a more dependable solution in domains requiring high accuracy, such as business rule inconsistency detection. However, a possible solution to this problem can be the approach proposed by [Borroto Santana et al., 2024], where natural language sentences are translated into ASP using our CNL2ASP system.

9.6 CNL2ASP

The language proposed by [Schwitter, 2018], PENG^{ASP}, presented some similarities with CNL2ASP. Albeit some aspects of PENG^{ASP}'s grammar rules are present in the grammar of our CNL, the latter is geared towards 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][1]).

```
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.
```

[1] Since PENG^{ASP} is not publicly available we could not compare the two languages on the other problems used in our evaluation.

Table 9.1 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

```
4 Red is a colour.      Green is a colour.      Blue is a colour.
5 Every node is assigned to exactly one colour.
6 It is not the case that a node X is assigned to a colour and a node Y
    is assigned to the colour and the node X is connected to the node
    Y.
```

There are two main differences with our CNL presented in Section 5.3.1. 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)`. Secondly, 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 (EBNF). 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 work are inspired by the Semantics of Business Vocabulary and Business Rules (SBVR) [Bajwa et al., 2011, The Business Rules Group, 2000], i.e., the standard proposed by the Object Management Group to formally describe complex entities, e.g., the ones related to a business, using natural language.

In Table 9.1 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.

9.7 CNL2TEL

CNL2TEL is based on TELINGO input langues. TELINGO is based on TEL over finite traces [Cabalar et al., 2018]. TEL [Aguado et al., 2013] was a first extension of ASP based on Linear Temporal Logic (LTL) [Pnueli, 1977]. Specifically, TELINGO adopts a restricted version of TEL similar to LTL over finite traces (LTL_f) [De Giacomo and Vardi, 2013], which results in a computationally more feasible formalization. Indeed, an advantage of this restriction is that it can be reduced to a less complex normal form compared to TEL [Cabalar et al., 2018, Aguado et al., 2013], which is, then, translated into ASP and incrementally solved using the multi shot feature of CLINGO [Gebser et al., 2019]. Details on the reduction of TEL programs into a normal form and the computation of temporal equilibrium models can be found in [Aguado et al., 2013]. Additionally, insights regarding LTL satisfiability checking have been provided through the experimental work conducted in one of the seminal papers by [Rozier and Vardi, 2010].

Regarding CNLs, here, 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, we refer the reader to the interesting survey in [Kuhn, 2014]. Although several CNLs have been defined with the aim of specifying logic programs in a natural language, such as Attempto CNL [Fuchs, 2005] or PENG^{ASP} [Schwitter, 2018], few of them provide an explicit way to represent temporal concepts and also their grammar is mainly domain-specific. [Holt and Klein, 1999] 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. [Esser and Struss, 2007] 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 [Caruso et al., 2024] by 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 [Borroto Santana et al., 2024]. [Schwitter, 2008] introduces the controlled natural language PENG Light and its associated language processor, which is based on a bidirectional grammar. Later, [Schwitter, 2011] presents the use of PENG Light to derive formal representations for reasoning about events and their outcomes. In this framework, specifications written in PENG Light are translated into the input language of the Simplified Event Calculus. Users can describe events, define states, and specify conditional statements; the system then generates a model that captures the knowledge about the events and the time points at which 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.

9.8 SBVR2ASP

Several studies concern the translation of SBVR into logical formalisms with the aim of detecting conflicts and inconsistencies. For example, [Solomakhin et al., 2013] proposed a method to translate SBVR, represented using Object-Role Modeling diagrams, into first-order deontic-alethic logic. Similarly, [Manaf et al., 2017] translated SBVR-SE specification into the declarative modeling language Alloy, focusing on the validation of service choreographies. However, their approach is limited to deontic rules, reflecting its domain-specific orientation. Other approaches, instead, employ SMT (Satisfiability Modulo Theories) for conflict detection. [Mitra et al., 2018] and [Chittimalli and Anand, 2016] developed a tool that converts SBVR-SE into SMT [de Moura and Bjørner, 2011] and uses SMT solvers (see, e.g., [Armando et al., 2005, de Moura and Bjørner, 2008]) for verification. However, their method does not consider all rules simultaneously in the verification phase. Instead, it first clusters business rules based on their relationships and SBVR vocabulary definitions to reduce the computational load. While this clustering improves efficiency, it may lead to undetected conflicts due to the exclusion of certain sentences during verification. In contrast, our approach, based on ASP, does not suffer from this limitation, as only grounded rules are processed by the solver. Alternative approaches employ machine learning techniques to identify inconsistencies. For example, [Aires et al., 2018] developed an approach to

detect conflicts using sentence embeddings, obtaining an accuracy of 95%. In [Aires and Meneguzzi, 2020], the authors introduced a two-phase approach using traditional machine learning and convolutional neural network to compare business rules in documents and detect conflicts. They obtained an accuracy of 84%. Similarly, [Huang et al., 2024] developed NeuralConflict, a convolutional neural network-based model, and tested it on English and Chinese datasets, achieving F1-scores of 0.979 and 0.958, respectively. A significant limitation of these approaches lies in the scarcity of training data, and researchers must often construct custom datasets, which are typically limited in size. For a comprehensive review of existing approaches and their limitations, readers are referred to [Schumann and Gómez, 2024]. Finally, it is important to note that a comparative evaluation across these tools is challenging. Many of them are not publicly available, and the datasets used in their evaluations are often inaccessible, limiting reproducibility and benchmarking.

9.9 CNLWizard

As already highlighted many CNLs have been proposed for different domains and tasks, our work can support researchers interested in proposing novel CNLs by reducing the effort required for their development. CNLWizard is a grammar-based system with a scope similar to the traditional compiler-compiler frameworks such as the Cornell Program Synthesizer [Teitelbaum and Reps, 1981]. However, while these frameworks generate parsers and compilers for formal programming languages through syntax-directed grammars and semantic rules, CNLWizard focuses on translating CNLs into KR formalisms. Moreover, unlike the interactive, template-based editing of the Synthesizer that guides users in building correct programs, CNLWizard generates a grammar using commands provided in a YAML-based format and processes a complete CNL specification. Though both are grammar-driven, CNLWizard serves domain experts modeling knowledge, not programmers building executable systems. Moreover, as already mentioned, CNLs can be used for improving the performance of LLMs, as shown by [Borroto Santana et al., 2024], where CNLs were used as an intermediate layer to map NL to ASP. Our tool could be used to define new CNLs, which in turn can serve as targets for converting NL sentences using their method.

Chapter 10

Conclusions and Future Work

This thesis addressed key challenges in scheduling problems within digital health and introduced novel approaches based on Knowledge Representation and Reasoning, with a particular focus on Answer Set Programming. In addition, it contributed to improving the usability and accessibility of KR formalisms by proposing tools that bridge the gap between formal logic representations and human-understandable language.

Regarding scheduling problems, we investigated and modelled different scheduling problems in digital health, focusing on two main applications: the Pre-operative Assessment Clinic Scheduling, and the scheduling of Periodic Treatments. For these problems, we developed ASP-based solutions and conducted an experimental analysis that demonstrated that ASP is a suitable solution for solving these types of problems, even with realistic, complex instances.

Beyond domain-specific modelling, a major contribution of this work lies in improving the usability and accessibility of KR formalisms, enabling non-expert users to interact with these formalisms more effectively.

To this end, we introduced CNL2ASP, a tool that translates a Controlled Natural Language, that we specifically designed for this purpose, into ASP. CNL2ASP was evaluated across multiple domains, showing that the CNL is both general and practical, supporting the specification of different ASP programs through an higher level language. A user validation confirmed the usability and effectiveness of the tool in facilitating the usability by non-technical users.

Additionally, CNL2ASP was extended into CNL2TEL, supporting temporal constructs in the CNL and converting it into the language of TELINGO. The tool has been applied and evaluated on the domains of the TELINGO suite: results show that our solution does not introduce significant overhead over TELINGO executed on the original encodings.

Then, we proposed a novel approach based on ASP for processing SBVR specification with SBVR2ASP. Our method directly addresses several limitations highlighted in a recent survey by [Schumann and Gómez, 2024], particularly those related to conflict detection in SBVR. To evaluate its effectiveness, we carried out an experimental analysis on synthetically generated datasets containing thousands of conflicts. The results demonstrate that our approach is well-suited for inconsistency checking in SBVR specification.

Finally, we developed CNLWizard, a tool designed to simplify the creation and customization of new controlled natural languages. By allowing users to define domain-specific CNLs in a more compact way, CNLWizard supports the broader adoption of KR systems through higher level languages.

Future work will aim to extend these approaches in several directions. First, we plan to further advance our research on scheduling problems in digital health by developing a general framework for their solution. This framework will exploit the similarities among the different scheduling problems, make usage of the LBBD approach to improve the performance.

Second, we aim to explore the use of the CNL approach in the field of explainability. Specifically, we will investigate methods for generating high-level natural language explanations of reasoning processes and solutions, enabling users to better understand and trust the outcomes of KR-based solutions. This step is crucial for ensuring the adoption and success of KR formalisms in real-world applications, where interpretability and user trust are essential.

References

Zahraa A Abdalkareem, Amiza Amir, Mohammed Azmi Al-Betar, Phaklen Ekhan, and Abdelaziz I Hammouri. Healthcare scheduling in optimization context: a review. *Health and Technology*, 11(3):445–469, 2021.

Elie Abi-Lahoud, Tom Butler, Donald Chapin, and John Hall. Interpreting regulations with SBVR. In *Proc. of International Rule Challenge, Special Track on Human Language Technology and RuleML DC*, volume 1004 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013. URL https://ceur-ws.org/Vol-1004/paper6.pdf.

Felicidad Aguado, Pedro Cabalar, Martín Diéguez, Gilberto Pérez, and Concepción Vidal. Temporal equilibrium logic: a survey. *J. Appl. Non Class. Logics*, 23(1-2):2–24, 2013. doi: 10.1080/11663081.2013.798985. URL https://doi.org/10.1080/11663081.2013.798985.

João Paulo Aires and Felipe Meneguzzi. Norm conflict identification using a convolutional neural network. In *Proc. of COIN and COINE*, volume 12298 of *LNCS*, pages 3–19. Springer, 2020.

João Paulo Aires, Juarez Monteiro, Roger Granada, and Felipe Meneguzzi. Norm conflict identification using vector space offsets. In *Proc. of IJCNN*, pages 1–8. IEEE, 2018.

Faezeh Akhavizadegan, Javad Ansarifar, and Fariborz Jolai. A novel approach to determine a tactical and operational decision for dynamic appointment scheduling at nuclear medical center. *Computers & Operations Research*, 78:267–277, 2017. ISSN 0305-0548.

Fares K. Alaboud and Andrew Coles. Personalized medication and activity planning in PDDL+. In J. Benton, Nir Lipovetzky, Eva Onaindia, David E. Smith, and Siddharth Srivastava, editors, *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2019, Berkeley, CA, USA, July 11-15, 2019*, pages 492–500. AAAI Press, 2019. URL https://ojs.aaai.org/index.php/ICAPS/article/view/3514.

O. M. Alade and A. O. Amusat. Solving nurse scheduling problem using constraint programming technique. *CoRR*, abs/1902.01193, 2019. URL http://arxiv.org/abs/1902.01193.

Lee-or Alon, Hana Weitman, Alexander Shleyfman, and Gal A. Kaminka. Planning to be healthy: Towards personalized medication planning. In Ulle Endriss, Francisco S. Melo, Kerstin Bach, Alberto José Bugarín Diz, Jose Maria Alonso-Moral, Senén Barro, and Fredrik Heintz, editors, *ECAI 2024 - 27th European Conference on Artificial Intelligence, 19-24 October 2024, Santiago de Compostela, Spain - Including 13th Conference on Prestigious Applications of Intelligent Systems (PAIS 2024)*, volume 392 of *Frontiers in Artificial Intelligence and Applications*, pages 4232–4239. IOS Press, 2024. doi: 10.3233/FAIA240996. URL https://doi.org/10.3233/FAIA240996.

Mario Alviano and Lorenzo Grillo. Answer set programming and large language models interaction with YAML: preliminary report. In *Proc. of CILC*, volume 3733 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2024. URL https://ceur-ws.org/Vol-3733/short2.pdf.

Mario Alviano, Francesco Calimeri, Carmine Dodaro, Davide Fuscà, Nicola Leone, Simona Perri, Francesco Ricca, Pierfrancesco Veltri, and Jessica Zangari. The ASP system DLV2. In *Proceedings of LPNMR*, volume 10377, pages 215–221. Springer, 2017a. doi: 10.1007/978-3-319-61660-5_19. URL https://doi.org/10.1007/978-3-319-61660-5_19.

Mario Alviano, Carmine Dodaro, and Marco Maratea. An advanced answer set programming encoding for nurse scheduling. In *AI*IA*, volume 10640 of *LNCS*, pages 468–482. Springer, 2017b.

Mario Alviano, Carmine Dodaro, and Marco Maratea. Nurse (re)scheduling via answer set programming. *Intelligenza Artificiale*, 12(2):109–124, 2018.

Mario Alviano, Giovanni Amendola, Carmine Dodaro, Nicola Leone, Marco Maratea, and Francesco Ricca. Evaluation of disjunctive programs in WASP. In Marcello Balduccini, Yuliya Lierler, and Stefan Woltran, editors, *LPNMR*, volume 11481 of *LNCS*, pages 241–255. Springer, 2019a.

Mario Alviano, Giovanni Amendola, Carmine Dodaro, Nicola Leone, Marco Maratea, and Francesco Ricca. Evaluation of disjunctive programs in WASP. In *LPNMR 2019*, volume 11481 of *LNCS*, pages 241–255. Springer, 2019b. doi: 10.1007/978-3-030-20528-7_18. URL https://doi.org/10.1007/978-3-030-20528-7_18.

Mario Alviano, Riccardo Bertolucci, Matteo Cardellini, Carmine Dodaro, Giuseppe Galatà, Muhammad Kamran Khan, Marco Maratea, Marco Mochi, Victoria Morozan, Ivan Porro, and Marco Schouten. Answer set programming in healthcare: Extended overview. In *IPS and RCRA 2020*, volume 2745 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020. URL http://ceur-ws.org/Vol-2745/paper7.pdf.

Giovanni Amendola, Carmine Dodaro, Nicola Leone, and Francesco Ricca. On the application of answer set programming to the conference paper assignment problem. In *AI*IA*, volume 10037 of *Lecture Notes in Computer Science*, pages 164–178. Springer, 2016.

Carlos Ansótegui, Toni Pacheco, and Josep Pon. Pypblib, 2019. URL https://pypi.org/project/pypblib/.

Alessandro Armando, Claudio Castellini, Enrico Giunchiglia, Massimo Idini, and Marco Maratea. TSAT++: an open platform for satisfiability modulo theories. *Electronic Notes in Theoretical Computer Science*, 125(3):25–36, 2005.

Imran Sarwar Bajwa, Mark G. Lee, and Behzad Bordbar. SBVR business rules generation from natural language specification. In *AI for Business Agility*. AAAI, 2011. URL http://www.aaai.org/ocs/index.php/SSS/SSS11/paper/view/2378.

Marcello Balduccini. Industrial-size scheduling with ASP+CP. In James P. Delgrande and Wolfgang Faber, editors, *Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings*,

volume 6645 of *Lecture Notes in Computer Science*, pages 284–296. Springer, 2011. doi: 10.1007/978-3-642-20895-9_33. URL https://doi.org/10.1007/978-3-642-20895-9_33.

Chitta Baral and Juraj Dzifcak. 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, 2012.

Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability - Second Edition*, volume 336 of *FAIA*, pages 1267–1329. IOS Press, 2021. doi: 10.3233/FAIA201017. URL https://doi.org/10.3233/FAIA201017.

Riccardo Bertolucci, Alessio Capitanelli, Carmine Dodaro, Nicola Leone, Marco Maratea, Fulvio Mastrogiovanni, and Mauro Vallati. Manipulation of articulated objects using dual-arm robots via answer set programming. *Theory Pract. Log. Program.*, 21(3):372–401, 2021a.

Riccardo Bertolucci, Carmine Dodaro, Giuseppe Galatà, Marco Maratea, Ivan Porro, and Francesco Ricca. Explaining ASP-based operating room schedules. In Riccardo De Benedictis, Marco Maratea, Andrea Micheli, Enrico Scala, Ivan Serina, Mauro Vallati, and Alessandro Umbrico, editors, *Proceedings of the 9th Italian workshop on Planning and Scheduling (IPS'21) and the 28th International Workshop on "Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion" (RCRA'21) with CEUR-WS co-located with 20th International Conference of the Italian Association for Artificial Intelligence (AIxIA 2021), Milan, Italy (virtual), November 29th-30th, 2021*, volume 3065 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2021b. URL https://ceur-ws.org/Vol-3065/paper7_182.pdf.

Manuel A. Borroto Santana, Irfan Kareem, and Francesco Ricca. Towards automatic composition of ASP programs from natural language specifications. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, pages 6198–6206. ijcai.org, 2024. URL https://www.ijcai.org/proceedings/2024/685.

A. Braaksma, N. Kortbeek, G.F. Post, and F. Nollet. Integral multidisciplinary rehabilitation treatment planning. *Operations Research for Health Care*, 3(3):145–159, 2014. ISSN 2211-6923. doi: https://doi.org/10.1016/j.orhc.2014.02.001. URL https://www.sciencedirect.com/science/article/pii/S2211692314000058.

Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

Pierangela Bruno, Francesco Calimeri, Cinzia Marte, and Marco Manna. Combining deep learning and ASP-based models for the semantic segmentation of medical images. In Sotiris Moschoyiannis, Rafael Peñaloza, Jan Vanthienen, Ahmet Soylu, and Dumitru Roman, editors, *Proceedings of the 5th International Joint Conference on Rules and Reasoning (RuleML+RR 2021)*, volume 12851 of *Lecture Notes in Computer Science*, pages 95–110. Springer, 2021.

Pierangela Bruno, Francesco Calimeri, and Cinzia Marte. Dedudeep: An extensible framework for combining deep learning and asp-based models. In Georg Gottlob, Daniela

Inclezan, and Marco Maratea, editors, *Proceedings of the 16th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2022)*, volume 13416 of *Lecture Notes in Computer Science*, pages 505–510. Springer, 2022.

Pierangela Bruno, Carmine Dodaro, Giuseppe Galatà, Marco Maratea, and Marco Mochi. Improving asp-based ORS schedules through machine learning predictions. *Theory Pract. Log. Program.*, 25(4):558–578, 2025. doi: 10.1017/S1471068425100136. URL https://doi.org/10.1017/s1471068425100136.

Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Enhancing Disjunctive Datalog by Constraints. *IEEE Transactions on Knowledge and Data Engineering*, 12(5):845–860, 2000.

Pedro Cabalar, Roland Kaminski, Torsten Schaub, and Anna Schuhmann. Temporal answer set programming on finite traces. *TPLP*, 18(3-4):406–420, 2018. doi: 10.1017/S1471068418000297. URL https://doi.org/10.1017/S1471068418000297.

Pedro Cabalar, Roland Kaminski, Philip Morkisch, and Torsten Schaub. telingo = ASP + time. In *Proc. of LPNMR*, volume 11481 of *LNCS*, pages 256–269. Springer, 2019. doi: 10.1007/978-3-030-20528-7_19. URL https://doi.org/10.1007/978-3-030-20528-7_19.

Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Francesco Ricca, and Torsten Schaub. ASP-Core-2 Input Language Format, 2013. URL https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.01c.pdf.

Paola Cappanera, Marco Gavanelli, Maddalena Nonato, and Marco Roma. A decomposition approach to the clinical pathway deployment for chronic outpatients with comorbidities. In Lavinia Amorosi, Paolo Dell'Olmo, and Isabella Lari, editors, *Optimization in Artificial Intelligence and Data Sciences*, pages 213–226, Cham, 2022. Springer International Publishing. ISBN 978-3-030-95380-5.

Paola Cappanera, Marco Gavanelli, Maddalena Nonato, and Marco Roma. Logic-based Benders decomposition in answer set programming for chronic outpatients scheduling. *Theory and Practice of Logic Programming*, 23(4):848–864, 2023a. doi: 10.1017/S147106842300025X.

Paola Cappanera, Marco Gavanelli, Maddalena Nonato, and Marco Roma. Decomposition approaches for scheduling chronic outpatients' clinical pathways in answer set programming. *J. Log. Comput.*, 33(8):1851–1871, 2023b.

Matteo Cardellini, Paolo De Nardi, Carmine Dodaro, Giuseppe Galatà, Anna Giardini, Marco Maratea, and Ivan Porro. A two-phase ASP encoding for solving rehabilitation scheduling. In Sotiris Moschoyiannis, Rafael Peñaloza, Jan Vanthienen, Ahmet Soylu, and Dumitru Roman, editors, *Proceedings of the 5th International Joint Conference on Rules and Reasoning (RuleML+RR 2021)*, volume 12851 of *Lecture Notes in Computer Science*, pages 111–125. Springer, 2021.

Simone Caruso, Giuseppe Galatà, Marco Maratea, Marco Mochi, and Ivan Porro. An ASP-based approach to scheduling pre-operative assessment clinic. In Stefania Bandini, Francesca Gasparini, Viviana Mascardi, Matteo Palmonari, and Giuseppe Vizzari, editors,

Proc. of AIxIA 2021 - Advances in Artificial Intelligence - 20th International Conference of the Italian Association for Artificial Intelligence, Revised Selected Papers, volume 13196 of *Lecture Notes in Computer Science*, pages 671–688. Springer, 2021.

Simone Caruso, Giuseppe Galatà, Marco Maratea, Marco Mochi, and Ivan Porro. Scheduling pre-operative assessment clinic with answer set programming. *Journal of Logic and Computation*, 34(3):465–493, 04 2023. ISSN 0955-792X. doi: 10.1093/logcom/exad017. URL https://doi.org/10.1093/logcom/exad017.

Simone Caruso, Carmine Dodaro, Marco Maratea, Marco Mochi, and Francesco Riccio. CNL2ASP: converting controlled natural language sentences into ASP. *Theory Pract. Log. Program.*, 24(2):196–226, 2024. doi: 10.1017/S1471068423000388. URL https://doi.org/10.1017/s1471068423000388.

Simone Caruso, Carmine Dodaro, Marco Maratea, and Marco Mochi. Grammar of the CNL. Available at https://github.com/dodaro/cnl2asp/blob/main/src/cnl2asp/grammar.lark., 2025a.

Simone Caruso, Carmine Dodaro, Marco Maratea, and Alice Tarzariol. Github repository of CNLWizard. MIT Licence, available at https://github.com/dodaro/CNLWizard, 2025b. URL https://github.com/dodaro/CNLWizard.

Marcello Ceci, Firas Al Khalil, and Leona O'Brien. Making sense of regulations with SBVR. In *Supplementary Proceedings of the RuleML 2016 Challenge, Doctoral Consortium and Industry Track*, volume 1620 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016. URL https://ceur-ws.org/Vol-1620/paper7.pdf.

Pavan Kumar Chittimalli and Kritika Anand. Domain-independent method of detecting inconsistencies in sbvr-based business rules. In *Proc. of ForMABS@ASE*, pages 9–16. ACM, 2016.

Noam Chomsky. On certain formal properties of grammars. *Inf. Control.*, 2(2):137–167, 1959. doi: 10.1016/S0019-9958(59)90362-6. URL https://doi.org/10.1016/S0019-9958(59)90362-6.

Peter Clark, Philip Harrison, Thomas Jenkins, John A. Thompson, and Richard H. Wojcik. Acquiring and using world knowledge using a restricted subset of english. In *Proceedings of FLAIRS*, pages 506–511. AAAI Press, 2005. URL http://www.aaai.org/Library/FLAIRS/2005/flairs05-083.php.

A. Condotta and N.V. Shakhlevich. Scheduling patient appointments via multilevel template: A case study in chemotherapy. *Operations Research for Health Care*, 3(3):129–144, 2014. ISSN 2211-6923. doi: https://doi.org/10.1016/j.orhc.2014.02.002. URL https://www.sciencedirect.com/science/article/pii/S221169231400006X.

Domenico Conforti, Francesca Guerriero, and Rosita Guido. Optimization models for radiotherapy patient scheduling. *4or*, 6:263–278, 2008.

Erica Coppolillo, Francesco Calimeri, Giuseppe Manco, Simona Perri, and Francesco Ricca. LLASP: fine-tuning large language models for answer set programming. In *Proc. of KR*, 2024.

Darin J. Correll, Angela M. Bader, Melissa W. Hull, Cindy Hsu, Lawrence C. Tsen, and David L. Hepner. Value of Preoperative Clinic Visits in Identifying Issues with Potential Impact on Operating Room Efficiency. *Anesthesiology*, 105(6):1254–1259, 12 2006. ISSN 0003-3022. doi: 10.1097/00000542-200612000-00026. URL https://doi.org/10.1097/00000542-200612000-00026.

Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 854–860. IJCAI/AAAI, 2013. URL http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997.

Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Z3: an efficient SMT solver. In *Proc. of TACAS*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Satisfiability modulo theories: introduction and applications. *Commun. ACM*, 54(9):69–77, 2011.

Michael Dingess and Miroslaw Truszczynski. Automated aggregator - rewriting with the counting aggregate. In *Proceedings of ICLP Technical Communications*, volume 325 of *EPTCS*, pages 96–109, 2020.

Carmine Dodaro and Marco Maratea. Nurse scheduling via answer set programming. In *LPNMR*, volume 10377 of *LNCS*, pages 301–307. Springer, 2017.

Carmine Dodaro, Giuseppe Galatà, Marco Maratea, and Ivan Porro. Operating room scheduling via answer set programming. In *AI*IA*, volume 11298 of *LNCS*, pages 445–459. Springer, 2018.

Carmine Dodaro, Giuseppe Galatà, Muhammad Kamran Khan, Marco Maratea, and Ivan Porro. An ASP-based solution for operating room scheduling with beds management. In Paul Fodor, Marco Montali, Diego Calvanese, and Dumitru Roman, editors, *Proceedings of the Third International Joint Conference on Rules and Reasoning (RuleML+RR 2019)*, volume 11784 of *Lecture Notes in Computer Science*, pages 67–81. Springer, 2019a.

Carmine Dodaro, Giuseppe Galatà, Marco Maratea, and Ivan Porro. An ASP-based framework for operating room scheduling. *Intelligenza Artificiale*, 13(1):63–77, 2019b.

Carmine Dodaro, Giuseppe Galatà, Andrea Grioni, Marco Maratea, Marco Mochi, and Ivan Porro. An ASP-based solution to the chemotherapy treatment scheduling problem. *Theory and Practice of Logic Programming*, 21(6):835–851, 2021.

Carmine Dodaro, Giuseppe Galatà, Muhammad Kamran Khan, Marco Maratea, and Ivan Porro. Operating room (re)scheduling with bed management via ASP. *Theory Pract. Log. Program.*, 22(2):229–253, 2022.

Carmine Dodaro, Giuseppe Galatà, Martin Gebser, Marco Maratea, Cinzia Marte, Marco Mochi, and Marco Scanu. Operating room scheduling via answer set programming: Improved encoding and test on real data. *Journal of Logic and Computation*, 34(8): 1556–1579, 2024a.

Carmine Dodaro, Giuseppe Mazzotta, and Francesco Ricca. Blending grounding and compilation for efficient ASP solving. In *KR*, 2024b.

G. M. Edward, S. F. Das, S. G. Elkhuizen, P. J. M. Bakker, J. A. M. Hontelez, M. W. Hollmann, B. Preckel, and L. C. Lemaire. Simulation to analyse planning difficulties at the preoperative assessment clinic. *BJA: British Journal of Anaesthesia*, 100(2):195–202, 02 2008a. ISSN 0007-0912.

GM Edward, SF Das, SG Elkhuizen, PJM Bakker, JAM Hontelez, MW Hollmann, B Preckel, and LC Lemaire. Simulation to analyse planning difficulties at the preoperative assessment clinic. *British journal of anaesthesia*, 100(2):195–202, 2008b.

Esra Erdem and Volkan Patoglu. Applications of ASP in robotics. *Künstliche Intell.*, 32(2-3): 143–149, 2018.

Esra Erdem and Reyyan Yeniterzi. Transforming controlled natural language biomedical queries into answer set programs. In *Proceedings of the BioNLP Workshop*, pages 117–124. Association for Computational Linguistics, 2009.

Esra Erdem, Michael Gelfond, and Nicola Leone. Applications of answer set programming. *AI Magazine*, 37(3):53–68, 2016.

Esra Erdem, Muge Fidan, David F. Manlove, and Patrick Prosser. A general framework for stable roommates problems using answer set programming. *Theory Pract. Log. Program.*, 20(6):911–925, 2020.

Michael Esser and Peter Struss. Obtaining models for test generation from natural-language-like functional specifications. In *Proc. of DX*, pages 75–82, 2007.

Wolfgang Faber, Gerald Pfeifer, and Nicola Leone. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, 175(1):278–298, 2011.

Andreas A. Falkner, Gerhard Friedrich, Konstantin Schekotihin, Richard Taupe, and Erich Christian Teppan. Industrial applications of answer set programming. *Künstliche Intelligenz*, 32(2-3):165–176, 2018.

Jorge Fandinno and Claudia Schulz. Answering the "why" in answer set programming - A survey of explanation approaches. *Theory Pract. Log. Program.*, 19(2):114–203, 2019. doi: 10.1017/S1471068418000534.

Juan Fdez-Olivares, Eva Onaindia, Luis A. Castillo, Jaume Jordán, and Juan A. Cózar. Personalized conciliation of clinical guidelines for comorbid patients through multi-agent planning. *Artif. Intell. Medicine*, 96:167–186, 2019. doi: 10.1016/J.ARTMED.2018.11.003. URL https://doi.org/10.1016/j.artmed.2018.11.003.

Marla B. Ferschl, Avery Tung, BobbieJean Sweitzer, Dezheng Huo, and David B. Glick. Preoperative Clinic Visits Reduce Operating Room Cancellations and Delays. *Anesthesiology*, 103(4):855–859, 10 2005. ISSN 0003-3022.

Norbert E. Fuchs. Knowledge representation and reasoning in (controlled) natural language. In *Proceedings of ICCS*, volume 3596 of *LNCS*, pages 51–51. Springer, 2005. doi: 10.1007/11524564_3. URL https://doi.org/10.1007/11524564_3.

Norbert E. Fuchs and Rolf Schwitter. Specifying logic programs in controlled natural language. *CoRR*, abs/cmp-lg/9507009, 1995. URL http://arxiv.org/abs/cmp-lg/9507009.

Luca Di Gaspero and Tommaso Urli. A CP/LNS approach for multi-day homecare scheduling problems. In Maria J. Blesa, Christian Blum, and Stefan Voß, editors, *Hybrid Metaheuristics - 9th International Workshop, HM 2014, Hamburg, Germany, June 11-13, 2014. Proceedings*, volume 8457 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2014. doi: 10.1007/978-3-319-07644-7_1. URL https://doi.org/10.1007/978-3-319-07644-7_1.

Marco Gavanelli, Maddalena Nonato, and Andrea Peano. An ASP approach for the valves positioning optimization in a water distribution system. *Journal of Logic and Computation*, 25(6):1351–1369, 2015.

Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187:52–89, 2012.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Philipp Wanko. Theory solving made easy with clingo 5. In *ICLP (Technical Communications)*, volume 52 of *OASICS*, pages 2:1–2:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

Martin Gebser, Nicola Leone, Marco Maratea, Simona Perri, Francesco Ricca, and Torsten Schaub. Evaluation techniques and systems for answer set programming: a survey. In Jérôme Lang, editor, *IJCAI*, pages 5450–5456. ijcai.org, 2018a.

Martin Gebser, Philipp Obermeier, Torsten Schaub, Michel Ratsch-Heitmann, and Mario Runge. Routing driverless transport vehicles in car assembly with answer set programming. *Theory and Practice of Logic Programming*, 18(3-4):520–534, 2018b. doi: 10.1017/S1471068418000182. URL https://doi.org/10.1017/S1471068418000182.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot ASP solving with clingo. *Theory Pract. Log. Program.*, 19(1):27–82, 2019.

Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference and Symposium , Seattle, Washington, August 15-19, 1988 (2 Volumes)*, pages 1070–1080. MIT Press, 1988. ISBN 0-262-61056-6.

Giovanni Grasso, Nicola Leone, Marco Manna, and Francesco Ricca. ASP at work: Spin-off and applications of the DLV system. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, volume 6565 of *LNCS*, pages 432–451. Springer, 2011.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021. URL https://www.gurobi.com.

Miriam J. P. Harnett, Darin J. Correll, Shelley Hurwitz, Angela M. Bader, and David L. Hepner. Improving Efficiency and Patient Satisfaction in a Tertiary Teaching Hospital Preoperative Clinic. *Anesthesiology*, 112(1):66–72, 01 2010. ISSN 0003-3022.

Mahmoud Heshmat and Amr B. Eltawil. Solving operational problems in outpatient chemotherapy clinics using mathematical programming and simulation. *Ann. Oper. Res.*, 298(1):289–306, 2021. doi: 10.1007/S10479-019-03500-Y. URL https://doi.org/10.1007/s10479-019-03500-y.

Alexander Holt and Ewan Klein. A semantically-derived subset of english for hardware verification. In *Proc. of ACL*, pages 451–456. ACL, 1999. doi: 10.3115/1034678.1034747. URL https://aclanthology.org/P99-1058/.

John N. Hooker and Willem Jan van Hoeve. Constraint programming and operations research. *Constraints An Int. J.*, 23(2):172–195, 2018. doi: 10.1007/S10601-017-9280-3. URL https://doi.org/10.1007/s10601-017-9280-3.

John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979. ISBN 0-201-02988-X.

Shaobin Huang, Jingyun Sun, and Rongsheng Li. Neuralconflict: Using neural networks to identify norm conflicts in normative documents. *Expert Syst. J. Knowl. Eng.*, 41(6), 2024.

Anali Huggins, David Claudio, and Eduardo Pérez. Improving resource utilization in a cancer clinic: An optimization model. In *IIE Annual Conference and Expo 2014*, 01 2014.

Alexey Ignatiev, António Morgado, and João Marques-Silva. RC2: an efficient maxsat solver. *J. Satisf. Boolean Model. Comput.*, 11(1):53–64, 2019. doi: 10.3233/SAT190116. URL https://doi.org/10.3233/SAT190116.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020. URL https://arxiv.org/abs/2001.08361.

Jaroslav Karpovic, Gintare Krisciuniene, Linas Ablonskis, and Lina Nemuraite. The comprehensive mapping of semantics of business vocabulary and business rules (SBVR) to OWL 2 ontologies. *Inf. Technol. Control.*, 43(3):289–302, 2014.

Jaroslav Karpovic, Linas Ablonskis, Lina Nemuraite, and Bronius Paradauskas. Experimental investigation of transformations from SBVR business vocabularies and business rules to owl 2 ontologies. *Inf. Technol. Control.*, 45(2):195–207, 2016.

Tobias Kuhn. A survey and classification of controlled natural languages. *Comput. Linguistics*, 40(1):121–170, 2014. doi: 10.1162/COLI_a_00168. URL https://doi.org/10.1162/COLI_a_00168.

Paolo Landa, Roberto Aringhieri, Patrick Soriano, Elena Tànfani, and Angela Testi. A hybrid optimization algorithm for surgeries scheduling. *Operations Research for Health Care*, 8: 103–114, 2016.

Lark. Lark repository. URL: https://github.com/lark-parser/lark.

Nicola Leone and Francesco Ricca. Answer set programming: A tour from the basics to advanced development tools and industrial applications. In Wolfgang Faber and Adrian Paschke, editors, *Reasoning Web. Web Logic Rules - 11th International Summer School 2015, Berlin, Germany, July 31 - August 4, 2015, Tutorial Lectures*, volume 9203 of *Lecture Notes in Computer Science*, pages 308–326. Springer, 2015. doi: 10.1007/978-3-319-21768-0_10. URL https://doi.org/10.1007/978-3-319-21768-0_10.

Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.

Vladimir Lifschitz. *Answer Set Programming*. Springer, 2019. ISBN 978-3-030-24657-0. doi: 10.1007/978-3-030-24658-7. URL https://doi.org/10.1007/978-3-030-24658-7.

Vladimir Lifschitz. Translating definitions into the language of logic programming: A case study. In *Proceedings of the ICLP Workshops*, volume 3193 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2022. URL https://ceur-ws.org/Vol-3193/short1ASPOCP.pdf.

Nurulhuda A. Manaf, Andreas Antoniades, and Sotiris Moschoyiannis. SBVR2Alloy: An SBVR to Alloy Compiler. In *Proc. of SOCA*, pages 73–80. IEEE Computer Society, 2017.

Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Open-wbo: A modular maxsat solver,. In *SAT 2014*, volume 8561 of *LNCS*, pages 438–445. Springer, 2014. doi: 10.1007/978-3-319-09284-3_33. URL https://doi.org/10.1007/978-3-319-09284-3_33.

Joren Marynissen and Erik Demeulemeester. Literature review on multi-appointment scheduling problems in hospitals. *European Journal of Operational Research*, 272(2):407–419, 2019. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2018.03.001. URL https://www.sciencedirect.com/science/article/pii/S0377221718302108.

Martin Michalowski, Szymon Wilk, Wojtek Michalowski, and Marc Carrier. Mitplan: A planning approach to mitigating concurrently applied clinical practice guidelines. *Artif. Intell. Medicine*, 112:102002, 2021. doi: 10.1016/J.ARTMED.2020.102002. URL https://doi.org/10.1016/j.artmed.2020.102002.

Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. Recent advances in natural language processing via large pre-trained language models: A survey. *ACM Comput. Surv.*, 56(2):30:1–30:40, 2024. doi: 10.1145/3605943. URL https://doi.org/10.1145/3605943.

Shervin Minaee, Tomás Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large language models: A survey. *CoRR*, abs/2402.06196, 2024. doi: 10.48550/ARXIV.2402.06196. URL https://doi.org/10.48550/arXiv.2402.06196.

Sayandeep Mitra, Kritika Anand, and Pavan Kumar Chittimalli. Identifying anomalies in sbvr-based business rules using directed graphs and smt-libv2. In *Proc. of ICEIS*, pages 215–222. SciTePress, 2018.

António Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-Guided MaxSAT with Soft Cardinality Constraints. In *CP 2014*, pages 564–573, Lyon, France, September 2014. Springer.

Vu Nguyen, Sophia Deeds-Rubin, Thomas Tan, and Barry Boehm. A sloc counting standard. In *Cocomo ii forum*, volume 2007, pages 1–16. Citeseer, 2007.

Olivier Roussel and Vasco Manquinho. Input/Output Format and Solver Requirements for the Competitions of Pseudo-Boolean Solvers, 2012. URL https://www.cril.univ-artois.fr/PB12/format.pdf.

OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023. doi: 10.48550/ARXIV.2303.08774. URL https://doi.org/10.48550/arXiv.2303.08774.

Umut Öztok and Esra Erdem. Generating explanations for complex biomedical queries. In *Proceedings of AAAI*. AAAI Press, 2011. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3519.

Yücel Öztürkoğlu. A different approach to nurse scheduling problem: Lagrangian relaxation. *Alphanumeric Journal*, 8(2):237–248, 2020.

Alessandro Dal Palù, Agostino Dovier, Andrea Formisano, and Enrico Pontelli. ASP applications in bio-informatics: A short tour. *Künstliche Intell.*, 32(2-3):157–164, 2018.

Hee-Sun Park, Sung-Hoon Kim, Myoung-Rye Bong, Dae-Kee Choi, Wook-Jong Kim, Seung-Woo Ku, Young Jin Ro, and In-Cheol Choi. Optimization of the operating room scheduling process for improving efficiency in a tertiary hospital. *Journal of Medical Systems*, 44(9):171, 2020.

Laurent Perron, Frédéric Didier, and Steven Gay. The CP-SAT-LP solver (invited talk). In *Proc. of CP*, volume 280 of *LIPIcs*, pages 3:1–3:2. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi: 10.4230/LIPICS.CP.2023.3. URL https://doi.org/10.4230/LIPIcs.CP.2023.3.

Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977. doi: 10.1109/SFCS.1977.32. URL https://doi.org/10.1109/SFCS.1977.32.

Eduardo Pérez, Lewis Ntaimo, Wilbert E. Wilhelm, Carla Bailey, and Peter McCormack. Patient and resource scheduling of multi-step medical procedures in nuclear medicine. *IIE Transactions on Healthcare Systems Engineering*, 1(3):168–184, 2011. doi: 10.1080/19488300.2011.617718.

Mohaimenul Azam Khan Raiaan, Md. Saddam Hossain Mukta, Kaniz Fatema, Nur Mohammad Fahad, Sadman Sakib, Most Marufatul Jannat Mim, Jubaer Ahmad, Mohammed Eunus Ali, and Sami Azam. A review on large language models: Architectures, applications, taxonomies, open issues and challenges. *IEEE Access*, 12:26839–26874, 2024. doi: 10.1109/ACCESS.2024.3365742. URL https://doi.org/10.1109/ACCESS.2024.3365742.

Francesco Ricca, Giovanni Grasso, Mario Alviano, Marco Manna, Vincenzino Lio, Salvatore Iiritano, and Nicola Leone. Team-building with answer set programming in the Gioia-Tauro seaport. *Theory and Practice of Logic Programming*, 12(3):361–381, 2012.

Heather Riley and Mohan Sridharan. Integrating non-monotonic logical reasoning and inductive learning with deep learning for explainable visual question answering. *Frontiers Robotics AI*, 6:125, 2019. doi: 10.3389/FROBT.2019.00125. URL https://doi.org/10.3389/frobt.2019.00125.

Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006. ISBN 978-0-444-52726-4. URL https://www.sciencedirect.com/science/bookseries/15746526/2.

Kristin Y. Rozier and Moshe Y. Vardi. LTL satisfiability checking. *Int. J. Softw. Tools Technol. Transf.*, 12(2):123–137, 2010. doi: 10.1007/S10009-010-0140-3. URL https://doi.org/10.1007/s10009-010-0140-3.

Paul Saikko, Jeremias Berg, and Matti Järvisalo. LMHS: A SAT-IP hybrid maxsat solver. In *SAT 2016*, volume 9710 of *LNCS*, pages 539–546. Springer, 2016. doi: 10.1007/978-3-319-40970-2_34. URL https://doi.org/10.1007/978-3-319-40970-2_34.

Katja Schimmelpfeng, Stefan Helber, and Steffen Kasper. Decision support for rehabilitation hospital scheduling. *OR spectrum*, 34:461–489, 2012.

Gerrit Schumann and Jorge Marx Gómez. Detection of conflicts, contradictions and inconsistencies in regulatory documents: A literature review. In *Proc. of IDSTA*, pages 81–88. IEEE, 2024.

Rolf Schwitter. English as a formal specification language. In *13th International Workshop on Database and Expert Systems Applications (DEXA 2002), 2-6 September 2002, Aix-en-Provence, France*, pages 228–232. IEEE Computer Society, 2002. doi: 10.1109/DEXA.2002.1045903. URL https://doi.org/10.1109/DEXA.2002.1045903.

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

Rolf Schwitter. Specifying events and their effects in controlled natural language. *Procedia - Social and Behavioral Sciences*, 27:12–21, 2011. ISSN 1877-0428. doi: https://doi.org/10.1016/j.sbspro.2011.10.578. URL https://www.sciencedirect.com/science/article/pii/S1877042811024050. Computational Linguistics and Related Fields.

Rolf Schwitter. Specifying and verbalising answer set programs in controlled natural language. *Theory Pract. Log. Program.*, 18(3-4):691–705, 2018. doi: 10.1017/S1471068418000327.

Rolf Schwitter, Bernhard Hamburger, and Norbert E. Fuchs. Attempto: Specifications in controlled natural language. In *Proceedings of the Workshop on Logische Programmierung*, pages 151–160, 1995.

Amit P. Sheth and Kaushik Roy. Neurosymbolic value-inspired artificial intelligence (why, what, and how). *IEEE Intell. Syst.*, 39(1):5–11, 2024. doi: 10.1109/MIS.2023.3344353. URL https://doi.org/10.1109/MIS.2023.3344353.

Dmitry Solomakhin, Enrico Franconi, and Alessandro Mosca. Logic-based reasoning support for SBVR. *Fundam. Informaticae*, 124(4):543–560, 2013.

Cameron Stark, Anne Gent, and Linda Kirkland. Improving patient flow in pre-operative assessment. *BMJ Open Quality*, 4(1), 2015. doi: 10.1136/bmjquality.u201341.w1226.

Hassan Tariq, Rafeeq Ahmed, Salil Kulkarni, Sana Hanif, Omesh Toolsie, Hafsa Abbas, and Sridhar Chilimuri. Development, functioning, and effectiveness of a preoperative risk assessment clinic. *Health Services Insights*, 2016:1, 10 2016. doi: 10.4137/HSI.S40540.

Tim Teitelbaum and Thomas Reps. The cornell program synthesizer: a syntax-directed programming environment. *Commun. ACM*, 24(9):563–573, September 1981. ISSN 0001-0782. doi: 10.1145/358746.358755. URL https://doi.org/10.1145/358746.358755.

The Business Rules Group. Defining business rules $\sim$ what are they really?, 2000. URL http://www.businessrulesgroup.org/first_paper/BRG-whatisBR_3ed.pdf.

The Object Management Group. Semantics Of Business Vocabulary And Business Rules. Version: 1.5, 2019.

Mark Wallace. Practical applications of constraint programming. *Constraints An Int. J.*, 1(1/2):139–168, 1996. doi: 10.1007/BF00143881. URL https://doi.org/10.1007/BF00143881.

Zhongsheng Wang, Jiamou Liu, Qiming Bao, Hongfei Rong, and Jingfeng Zhang. Chatlogic: Integrating logic programming with large language models for multi-step reasoning. In *Proc. of IJCNN*, pages 1–8. IEEE, 2024. doi: 10.1109/IJCNN60899.2024.10650138. URL https://doi.org/10.1109/IJCNN60899.2024.10650138.

C. L. Woodrum, M. Wisniewski, D. J. Triulzi, J. H. Waters, L. H. Alarcon, and M. H. Yazer. The effects of a data driven maximum surgical blood ordering schedule on preoperative blood ordering practices. *Hematology*, 22(9):571–577, 2017.

Qian Xiao, Li Luo, Shu-zhen Zhao, Xiaobin Ran, and Ya-bing Feng. Online appointment scheduling for a nuclear medicine department in a chinese hospital. *Comput. Math. Methods Medicine*, 2018:5148215:1–5148215:13, 2018.

Xiaocheng Yang, Bingsen Chen, and Yik-Cheung Tam. Arithmetic reasoning with LLM: prolog generation & permutation. *CoRR*, abs/2405.17893, 2024. doi: 10.48550/ARXIV.2405.17893. URL https://doi.org/10.48550/arXiv.2405.17893.

Zhun Yang, Adam Ishay, and Joohyung Lee. Coupling large language models with logic programming for robust and general reasoning from text. In *ACL*, pages 5186–5219. Association for Computational Linguistics, 2023.

Liangmeng Yu, Huiyue Zhou, Jiamei Li, and Xiaoling Yu. Shift work sleep disorder in nurses: a concept analysis. *BMC nursing*, 24(1):18, 2025.

Jian Zhang, Mahjoub Dridi, and Abdellah El Moudni. A stochastic shortest-path MDP model with dead ends for operating rooms planning. In *ICAC*, pages 1–6. IEEE, 2017.

Zheng Zhang and Xiaolan Xie. Simulation-based optimization for surgery appointment scheduling of multiple operating rooms. *IIE Transactions*, 47(9):998–1012, 2015.

Appendix A

CNL2ASP generated encodings

A.1 Encoding generated by our CNL tool for reald-world use cases

A.1.1 Nurse Scheduling Problem

CNL specifications reported in Section 5.4.1 are automatically converted as ASP rules by our tool. The generated encoding is the following:

```
nurse(1..numberOfNurses).
day(1..365).
shift(1,"morning",7).
shift(2,"afternoon",7).
shift(3,"night",10).
shift(4,"specrest",0).
shift(5,"rest",0).
shift(6,"vacation",0).
1 <= {work_in(NURSE,DAY,SHIFT):shift(_,SHIFT,_)} <= 1 :-
    nurse(NURSE), day(DAY).
:- day(DAY), #count{NURSE: work_in(NURSE,DAY,"morning")} >
    maxNurseMorning.
:- day(DAY), #count{NURSE: work_in(NURSE,DAY,"afternoon")} >
    maxNurseAfternoon.
:- day(DAY), #count{NURSE: work_in(NURSE,DAY,"night")} >
    maxNurseNight.
```

```
13 :- day(DAY), #count{NURSE: work_in(NURSE,DAY,"morning")} <
   minNurseMorning.
14 :- day(DAY), #count{NURSE: work_in(NURSE,DAY,"afternoon")} <
   minNurseAfternoon.
15 :- day(DAY), #count{NURSE: work_in(NURSE,DAY,"night")} <
   minNurseNight.
16 :- nurse(NURSE), #sum{HOURS,DAY: work_in(NURSE,DAY,SHIFT),
   shift(_,SHIFT,HOURS)} > 1692.
17 :- nurse(NURSE), #sum{HOURS,DAY: work_in(NURSE,DAY,SHIFT),
   shift(_,SHIFT,HOURS)} < 1687.
18 :- nurse(NURSE), #count{DAY: work_in(NURSE,DAY,"vacation")} != 30.
19 :- shift(SO1,SHIFT1,_), shift(SO2,SHIFT2,_),
   work_in(NURSE,DAY,SHIFT2), work_in(NURSE,DAY+1,SHIFT1), SO1 < SO2,
   SO2 >= 1, SO2 <= 3, SO1 >= 1, SO1 <= 6.
20 :- nurse(NURSE), day(D1), D1 <= 352, #count{D2:
   work_in(NURSE,D2,"rest"), D2 >= D1, D2 <= D1+13} < 2.
21 :- day(D1), #count{D2: work_in(NURSE,D2,"night"), D2 >= D1, D2 <=
   D1+1} = 2, not work_in(NURSE,D1+2,"specrest"), nurse(NURSE).
22 :- work_in(NURSE,DAY,"specrest"), day(DAY), #count{SHIFT:
   work_in(NURSE,SHIFT,"night"), SHIFT >= DAY-2, SHIFT <= DAY-1} != 2.
23 :- nurse(NURSE), #count{DAY: work_in(NURSE,DAY,"morning")} > 82.
24 :- nurse(NURSE), #count{DAY: work_in(NURSE,DAY,"afternoon")} > 82.
25 :- nurse(NURSE), #count{DAY: work_in(NURSE,DAY,"night")} > 61.
26 :- nurse(NURSE), #count{DAY: work_in(NURSE,DAY,"morning")} < 74.
27 :- nurse(NURSE), #count{DAY: work_in(NURSE,DAY,"afternoon")} < 74.
28 :- nurse(NURSE), #count{DAY: work_in(NURSE,DAY,"night")} < 58.
29 :~ nurse(NURSE), #count{DAY: work_in(NURSE,DAY,"morning")} = TOTDAYS,
   TOTDAYS >= 74, TOTDAYS <= 82, RES = |78 - TOTDAYS|. [RES@3, NURSE]
30 :~ nurse(NURSE), #count{DAY: work_in(NURSE,DAY,"afternoon")} =
   TOTDAYS, TOTDAYS >= 74, TOTDAYS <= 82, RES = |78 - TOTDAYS|.
   [RES@3, NURSE]
31 :~ nurse(NURSE), #count{DAY: work_in(NURSE,DAY,"night")} = TOTDAYS,
   TOTDAYS >= 58, TOTDAYS <= 61, RES = |60 - TOTDAYS|. [RES@3, NURSE]
```

A.1.2 Manipulation of Articulated Objects Using Dual-Arm Robots

CNL specifications reported in Section 5.4.2 are automatically converted as ASP rules by our tool. The generated encoding is the following:

```
time(1). time(2). time(3). time(4). time(5). time(6). time(7).
    time(8). time(9). time(10).
link(J2,J1) :- link(J1,J2).
0 <= {rotation(J1,J2,A,AI,time(T)):joint(J1), joint(J2), angle(A),
    link(J1,J2), position(joint(J1),angle(AI),time(T))} <= 1 :-
    time(T), T > 0.
:- rotation(_,_,_,_,time(T)), T >= timemax.
:- rotation(J1,J2,_,_,time(_)), J1 <= J2.
:- rotation(_,_,A,AI,time(_)), (A)\360 == (AI)\360.
:- rotation(_,_,A,AI,time(_)), (A)\360 > 0, (AI)\360 > (A)\360,
    (AI)\360 != (A + granularity)\360.
:- rotation(_,_,A,AI,time(_)), (AI)\360 > 0, (A)\360 > (AI)\360,
    (A)\360 != (AI + granularity)\360.
:- rotation(_,_,A,0,time(_)), (A)\360 != 360 - granularity.
:- rotation(_,_,0,AI,time(_)), (AI)\360 != 360 - granularity.
1 <= {position(joint(J),angle(A),time(T)):angle(A)} <= 1 :- joint(J),
    time(T).
:- position(joint(J),angle(A1),time(T)),
    position(joint(J),angle(A2),time(T+1)), not
    rotation(_,_,_,_,time(T)), T <= timemax, (A1)\360 != (A2)\360.
:- position(joint(J1),angle(A1),time(T)),
    rotation(J1,_,A2,_,time(T-1)), (A1)\360 != (A2)\360.
:- time(T), position(joint(J1),angle(AN),time(T+1)),
    rotation(J2,_,A,AP,time(T)),
    position(joint(J1),angle(AC),time(T)), J1 > J2, (AN)\360 !=
    (|AC+(A-AP)+360|)\360.
:- position(joint(J1),angle(A1),time(T)),
    position(joint(J1),angle(A2),time(T+1)),
    rotation(J2,_,_,_,time(T)), J2 > J1, T <= timemax, (A1)\360 !=
    (A2)\360.
```

```
16 :- goal(joint(J),angle(A1)),
      position(joint(J),angle(A2),time(timemax)), (A1)\360 != (A2)\360.
```

We want to emphasize here that each angle is automatically followed by `\ 360` which represents the module operation to ensure that an angle is always between 0 and 359 degrees.

A.1.3 Chemotherapy Treatment Scheduling Problem

CNL specifications reported in Section 5.4.3 are automatically converted as ASP rules by our tool. The generated encoding is the following:

```
1 timeslot(1,"07:30"). timeslot(2,"07:40"). timeslot(3,"07:50").
2 timeslot(4,"08:00"). timeslot(5,"08:10"). timeslot(6,"08:20").
3 timeslot(7,"08:30"). timeslot(8,"08:40"). timeslot(9,"08:50").
4 timeslot(10,"09:00"). timeslot(11,"09:10"). timeslot(12,"09:20").
5 timeslot(13,"09:30"). timeslot(14,"09:40"). timeslot(15,"09:50").
6 timeslot(16,"10:00"). timeslot(17,"10:10"). timeslot(18,"10:20").
7 timeslot(19,"10:30"). timeslot(20,"10:40"). timeslot(21,"10:50").
8 timeslot(22,"11:00"). timeslot(23,"11:10"). timeslot(24,"11:20").
9 timeslot(25,"11:30"). timeslot(26,"11:40"). timeslot(27,"11:50").
10 timeslot(28,"12:00"). timeslot(29,"12:10"). timeslot(30,"12:20").
11 timeslot(31,"12:30"). timeslot(32,"12:40"). timeslot(33,"12:50").
12 timeslot(34,"13:00"). timeslot(35,"13:10"). timeslot(36,"13:20").
13 day(1,"01/01/2022").
14 day(2,"02/01/2022").
15 day(3,"03/01/2022").
16 day(4,"04/01/2022").
17 day(5,"05/01/2022").
18 day(6,"06/01/2022").
19 day(7,"07/01/2022").
20 1 <= {assignment(registration(patient(R),0), day(D), timeslot(TS)) :
      day(D,_), timeslot(TS,_) } <= 1 :-
      registration(patient(R),0,_,_,_,_,_).
21 1 <= {assignment(registration(patient(P),OR),day(D+W),timeslot(T)) :
      timeslot(T,_)} <= 1 :- registration(patient(P),OR,W,_,_,_,_),
      assignment(registration(patient(P),OR-1), day(D), timeslot(_)),
      day(D+W,_).
```

```
22 1 <= {position_in(patient(P),S) : seat(S,_)} <= 1 :- patient(P,_),
       assignment(registration(patient(P),_), day(D), timeslot(T)),
       registration(patient(P),_,_,_,_,_,_,PH4), PH4 > 0.
23 position_in(patient(P),S,timeslot(T), day(D)) :-
       _generated_support(patient(P),S,timeslot(T),day(D)),
       position_in(patient(P),S).
24 _generated_support(patient(P),S,timeslot(T..T+PH4-1),day(D)) :-
       position_in(patient(P),S), patient(P,_),
       assignment(registration(patient(P),_), day(D), timeslot(T)),
       registration(patient(P),_,_,_,_,_,_,PH4).
25 :- registration(patient(P),OR,_,PH1,PH2,PH3,_),
       assignment(registration(patient(P),OR),day(_),timeslot(T)), T <=
       PH1 + PH2 + PH3.
26 :- day(D,_), timeslot(TS,_), seat(S,_), #count{P:
       position_in(patient(P),S,timeslot(TS),day(D))} >= 2.
27 :- registration(patient(P),OR,_,_,_,_,_,PH4),
       assignment(registration(patient(P),OR),day(_),timeslot(TS)), PH4 >
       50, TS < 24.
28 :~ seat(S,T), patient(P,T),
       position_in(patient(P),S,timeslot(_),day(_)). [-1@3, P]
```