Fair Shifts by the Rule: a Rule-based Compliance Methodology for Medical Rosters

Guido Governatori[1,2,*], Ilaria Angela Amantea[3], Marinella Quaranta[3,4], Marianna Molinari[3,4], Simone Vagnoni[4], Andrea Filippo Ferraris[4], Giuseppe Contissa[4], Monica Palmirani[4], Marco Busso[5] and Marco Grosso[5]

[1]College of Information and Communication Technology, Central Queensland University
[2]Artificial Intelligence and Cyber Futures Institute, Charles Sturt University
[3]Computer Science Department, University of Turin
[4]LaST-JD, Univeristy of Bologna
[5]Azienda Sanitaria Locale ASLTO3, Turin

Abstract

We present a rule-based system to check whether a roster of doctors in a healthcare setting complies with the work conditions mandated by the Italian National Contracts of Employment for medical doctors.

Keywords

Compliance, Defeasible Deontic Logic, E-Health, Healthcare, Scheduling,

1. Introduction

In healthcare, especially in hospitals, each human asset is fundamental. Each part of the medical and non-medical staff ensures the proper working of health services, and the time they spend working for health services affects the outcome of every medical care service.

A well-defined hospital scheduling, can increase productivity and leads to an efficient use of infrastructure and facilities [1]. Scheduling is needed in each medical department and for each type of worker (doctors, nurses, socio-sanitary workers, and administrative staff).

At least in Italy, the medical staff currently spends part of their working time on administrative tasks, both because of a lack of administrative staff and their expertise in medical logistics. Scheduling medical rosters is an example of an administrative task carried out by the medical staff (also, the Italian National Collective Labour Agreement in Healthcare regulating the working conditions of physicians in hospitals mandates this activity to chief physicians in charge of a medical departments). A well-fixed roster scheduling in a specific department is a long process and involves knowledge of medical procedures (their duration, the availability of medical staff, their specialities, the necessity of further assistance by other members of the hospital staff, the availability of rooms, etc.). The roster scheduling has to be carried out by the category chief, e.g. the chief physician of the department, the chief nurse of the department etc. This is for reason of responsibility, although it is a lengthy administrative task that takes time away from patient care for the department's most experienced personnel.

RuleML+RR'24: Companion Proceedings of the 8th International Joint Conference on Rules and Reasoning, September 16–22, 2024, Bucharest, Romania

*Corresponding author.
g.governatori@cqu.edu.au (G. Governatori); ilariaangela.amantea@unito.it (I. A. Amantea); marinella.quaranta@unito.it (M. Quaranta); marianna.molinari@unito.it (M. Molinari); simone.vagnoni3@unibo.it (S. Vagnoni); andrea.ferraris3@unibo.it (A. F. Ferraris); giuseppe.contissa@unibo.it (G. Contissa); monica.palmirani@unibo.it (M. Palmirani); marco.busso@aslto3.piemonte.it (M. Busso); marco.grosso@aslto3.piemonte.it (M. Grosso)
0000-0002-9878-2762 (G. Governatori); 0000-0003-1329-1858 (I. A. Amantea); 0000-0003-2691-0611 (M. Quaranta); 0009-0003-1832-8135 (M. Molinari); 0009-0008-8296-8299 (S. Vagnoni); 0009-0004-7487-5560 (A. F. Ferraris); 0000-0002-8511-1505 (G. Contissa); 0000-0002-8557-8084 (M. Palmirani); 0000-0001-8176-8641 (M. Busso); 0000-0002-7083-0929 (M. Grosso)

Talking with many physicians of different Italian hospitals, currently specific tools or advanced tools to accomplish scheduling are inadequate. They simply combine shifts and staff, but without taking into account the particular needs of the staff, needs of the medical procedures, needs of the structure, legal guidelines and are not versatile enough to meet imminent and not programmable well in advance needs as in hospital environment is required.

In conclusion, in most Italian hospitals, scheduling rosters is still a manual on-paper task, or sometimes with the additional support of basic programs like Excel.

Making shifts means taking into account many types of factors:

- Organizational. Such as: how much staff is available? With what specializations? How many services need to be covered? etc.
- Legal. Such as:
  - National law: professional requirements, physical restrictions (e.g. certified disabilities).
  - National labour law: e.g. how many (min and max) hours can people work, national holidays, permits, etc.
  - Professional guidelines: e.g. minimum specialised staff requiredfor a specific exam or service, etc.
  - Internal procedures necessities.
- Personal necessities: holiday requests, permits, health problems, physical restrictions, preference in the services provided, etc.
- Last minute occurrences: swap shifts, events that unexpectedly necessitate rescheduling a large number of shifts for a large amount of staff.

Based on our own personal experience (i.e., monthly creation of rosters) and anecdotal discussion with other chief physicians in other Italian hospitals, planning shifts is a very long task. During the planning, a plurality of requirements needs to be checked and, once the roster draft is completed, it has to be checked again from the perspective of equity and fairness. This double-check is needed for both a good quality working environment and a good quality service for patients.

In the healthcare field, new emerging technologies are often tested to improve care and general organization i.e. process mining [2], medical devices [3], tools for telemedicine [4] and now AI systems [5].

AI can improve several tasks related to care. AI could help schedule and suggest the best solutions in shift drafting, considering various features and variables. It would allow for human modification while guaranteeing equality of treatment for everyone.

In this first paper, we first focus on the aim to provide a framework to automatically support the compliance checking of medical rosters in Italy. This is because the lack of legal compliance checking is the common missing requirement of the existing tools and because it is the fundamental requirement for the rosters, whether they are hand-made or automatically created. Even if the methodology and the future tool can be used in every country, for this first example, we will consider Italy and Italian regulations.

The compliance is checked according to the Italian National Collective Labour Agreement in Healthcare (CCNL)[1]. For reason of space, we will provide an example with just a pair of rules. From a broader perspective of optimization, the decrease in working time for an administrative task of a highly qualified resource in the medical field would also ensure the possibility of visiting more patients and, thus, the reduction of waiting times for medical services.

In the following section the background is presented. In Section 3, the architecture of the compliance checker is shown, and for each subsection, the details of every layer are highlighted. In Section 4, a summary and future work are presented.

[1]Contratto Collettivo Nazionale di Lavoro (CCNL) dell'area sanità, GU Serie Generale n.59 del 11-03-2024 in https://www.gazzettaufficiale.it/eli/id/2024/03/11/24A01226/sg

2. Background

In Italy, the chief of every professional category of the healthcare sector is in charge of the roster drafting. In our paper, we focus on the roster scheduling for medical doctors.

The chief physician is in charge of scheduling rosters because they know physicians' medical specialties, preferences, and necessities. The chief physician is also aware of restraints and requirements related to medical shifts according to the national legal framework.

From the collaborations with many main Italian hospitals, it emerged that the chief physician spends approximately 16 hours of work to finalize the monthly schedule, considering numerous variables that need to be coordinated. For example, every person on the night shift must take mandatory rest and must not be allocated to the next shift. Furthermore, night shifts must be equally distributed among the available physicians. Every activity must be assigned to somebody , when not resting. Also, staff rotate over the activities, attending different shifts and avoiding repeating the shifts they were allocated to for the previous days.

The CCNL (namely "Contratto Collettivo Nazionale di Lavoro dell'Area Sanità" in Italian) is the national legal framework for physicians' rosters. According to the Italian legal system, professional activities and remunerations are not established only by the law. They are established through collective negotiations conducted by trade unions of a specific professional sector. The CCNL is the pipeline for drafting legally compliant rosters as it defines the amount of working hours, how they should be allocated, rules for holidays and leave distributions, night shifts and non-stop shifts, etc.

The chief physician ensures compliance with these rules, but their work is affected by spending a significant number of hours per month on a non-medical activity. This purely administrative task of checking the legal compliance of the scheduled roster can and should be automated.

What we aim to propose is a methodology for checking the legal compliance of physician rosters according to the above-mentioned national regulation.

Starting from the assumption that being compliant means satisfying a set of measures to ensure that the (business) activities align to (legal) requirements they are subject to [6, 7]; the Defeasible Deontic Logic (DDL) [8] provides a conceptually sound, domain-neutral and computationally efficient representation of normative specifications. The reasoning mechanism of DDL has a (constructive) argumentation-like structure, and its language obeys the principles set in [9] for a rule language to be suitable for the representation of legal knowledge compatible with the LegalRuleML standard [10].

DDL has been successfully adopted to norms from a wide range of domains, including regulations [11], (business) contracts [12], autonomous agents and agent policies [13], building regulations [14], traffic rules [15], health-care policies [16] and the Australian spent conviction discipline [17]. Accordingly, DDL should be able to handle the legal requirements set for the work conditions for medical doctors by the CCNL. In what follows, we give a short overview of DDL, and we illustrate the language with some examples of rules corresponding to CCNL provisions.

In DDL, a norm is represented as an IF… THEN⋯ rule, where the IF part describes the conditions of applicability of the rule/norm, and the THEN part is the legal effect of the norm. There are two types of effects: a rule defines a term in the context of the set or norms (*constitutive rule*), or the rule mandates that a normative requirement –obligation, prohibition, permission–is in force if the conditions of applicability hold (*normative rule*, subdivided in *prescriptive rules* and *permissive rules*). Thus, we have expressions of the form

$$r\colon\ a_1, \dots, a_n \Rightarrow_X e$$

where r is the rule label, unique for each rule; $a_1, \dots, a_n$ are the conditions of the applicability (represented as propositions or deontic propositions); e, again a proposition, is the conclusion or effect of the rule; and X specifies the type of rule. If $X = C$ we have a constitutive rule, and the effect is e; otherwise, we have a normative rule, and the conclusion e is within the scope of a deontic operator. The operator depends on the value of X, where we have the obligation (or prohibition) of c (i.e., Oc) if $X = \mathrm{O}$ or that c is permitted (i.e., Pc) if $X = \mathrm{P}$. The language and the reasoning mechanism of DDL allow us to deal with a complete family of legal requirements [18], including violations and compensatory measures,

and a rich and natural treatment of exceptions [8]: when rules conflict we can specify that one of the rules defeats the other (i.e., $r_1 \succ r_2$, rule r_1 overrules rule r_2).

For example, the norm stipulating that the service offered by a doctor in stand-by called for duty is regarded as a medical emergency service can be formalised by the following constitutive rule:

$$Service(standBy), Shift(Doctor, standBy, Day, Week), Worked(Doctor, standBy, Day, Week), Emergency(standBy) \Rightarrow_C Worked(Doctor, Service, Day, Week)$$

This rule specifies that stand-by is one of the services; if a doctor is scheduled to be on stand-by for a particular shift and is called for duty, then the service counts as having worked in an emergency service (this has implications for the rate of pay and the future shifts to be allocated to the doctor).

On the other hand, an example of a prescriptive rule is the following

$$Director(Doctor), Emergency(onCall) \Rightarrow_O \neg Shift(Doctor, onCall, Day, Week)$$

encoding section 29.2 that forbids directors of medical units to serve as "doctor-on-call". However, the prohibition is overridden by the permissive rule

$$Director(Doctor), \neg CoveredServices(Day, Week), Emergency(onCall) \Rightarrow_P Shift(Doctor, onCall, Day, Week)$$

that allows unit directors to be a doctor-on-call if the personnel available is insufficient to cover the required services.

The reasoning mechanism is based on a three-phase argumentation-like structure: (1) in the first phase, we identify an argument (either a fact or an applicable rule) for the conclusion we want to assert. (2) In the second phase, we consider possible counter-arguments (i.e., rules for the opposite conclusion). (3) We must rebut the counter-arguments in the third and final phase. We have two ways. (i) undercut: we show that the rule is not applicable, as some premises do not hold; (ii) defeat: we use the superiority relation to show that a counter-argument is weaker than an applicable rule for the conclusion we want to prove.

For the full details about the logic, its properties, algorithms, motivations, and applications, we refer the readers to [19, 8].

3. Legal Rostering Architecture

We are going to discuss the architecture of the rostering compliance checking tool, shown in Fig. 1. The aim of the tools is, given a doctor roster and the conditions mandated in the CCNL, to determine if the allocation of the doctors to the various shifts in the roster conforms with the legal requirements set by the CCNL.

For implementing our medical roster compliance checking system, we adopted CLINGO[2]. CLINGO is a modern and efficient implementation of the Answer Set Programming (ASP) [20], and it allows for the integration with other programming languages (Python and Lua) for pre-, post- and interleaving processing. For our purposes, we opted for a multi-phase process that integrates Python for data (pre)processing, LegalRuleML for representing norms stated in the CCNL in a machine-processable format, and an implementation of Defeasible Deontic Logic in ASP for logical modelling of the norms.

Here, we describe each phase.

3.1. LegalRuleML

As we referred to above, we need a formal representation of the norms established in the CCNL. We manually translated the relevant section of CCNL in LegalRuleML. LegalRuleML [21, 22] is an XML-based standard for the representation of norms and legal knowledge. LegalRuleML is technology-neutral

[2] https://potassco.org.

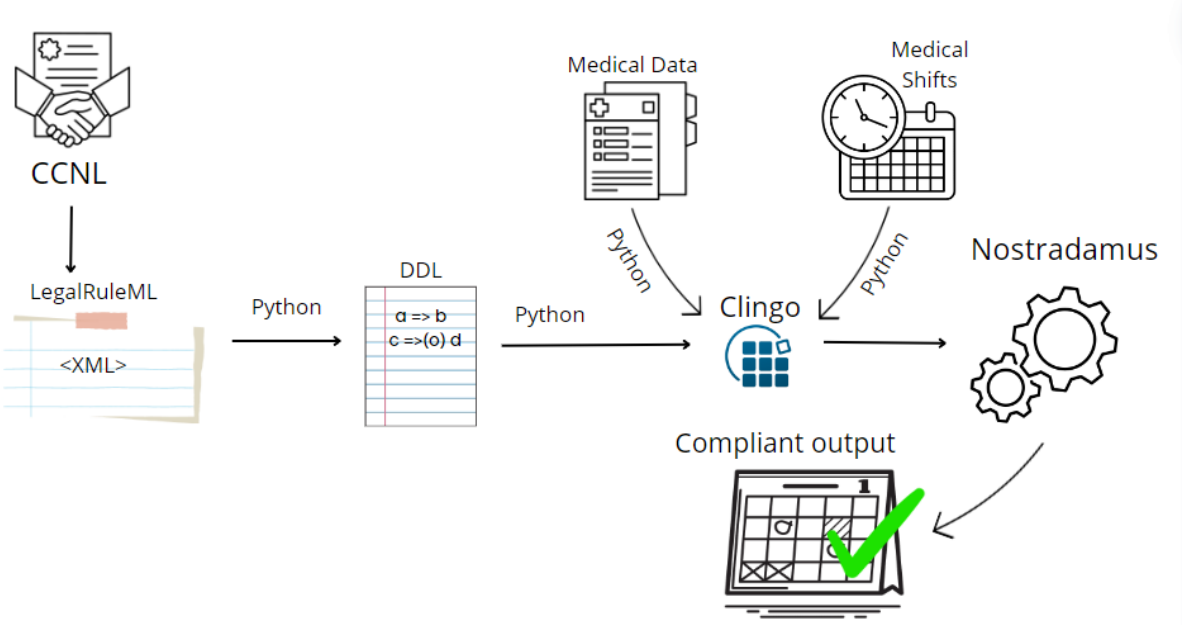

Figure 1: Architecture of the rostering compliance checking tool.

in the sense that the representation does not depend on any specific language or system to formalise and reason with the norms, and the LegalRuleML encoding can be translated to a target language and implementation. At the same time, the standard offers features such as distinction between normative and prescriptive rules, defeasible rules, override relationships over rules, deontic operators, makes it suitable to encode norms.

For instance, we show how to encode (part of) Section 27 establishing the limit of hours a doctor can work in a week (or, to be more precise, the maximum number of hours the doctor can be allocated to when a roster is prepared).

```
<lrml:PrescriptiveStatement key=":art27_1">
    <lrml:hasStrength><lrml:Defeasible/></lrml:hasStrength>
    <ruleml:Rule key=":art_27_1_Doctor_Week" closure="universal">
        <ruleml:then>
            <lrml:Obligation>
                <ruleml:Atom>
                    <ruleml:Rel iri="#Weekly_limit"/>
                    <ruleml:Var iri="#Doctor"/>
                    <ruleml:Var iri="#Week"/>
                </ruleml:Atom>
            </lrml:Obligation>
        </rulem:then>
    <rulemL:Rule>
</lrml:PrescriptiveStatement>
```

In LegalRuleML, a rule is a statement; where a statement can be either prescriptive or constitutive (with the same meaning discussed in the previous section). Similarly, a rule has an if-then structure (inherited from RuleML, so they are in the `ruleml` namespace). The "if" and "then" parts of a rule contain logical expressions (with Boolean operators), where the basic element is an `Atom`. An atom can be declared with a `Rel` (a predicate or a proposition). For a predicate, it is possible to specify its arguments. An argument can be either a variable (`Var`) or a constant `Con`. The `iri` points to the definition of a predicate/proposition and the specific variables and constants. LegalRuleML extends

RuleML with deontic operators (`Obligation`, `Permission`, and `Prohibition`); see [21] for the full description of the LegalRuleML standard.

Accordingly, the rule above states that normally, the scheduled hours for a doctor cannot exceed the weekly limits; thus, we have the predicate `Weekly_limit` that takes a doctor and a week as its argument. However, CCNL also states that the limit can be exceeded if there are some specific service requirements; the following LegalRuleML statement captures this.

```
<lrml:PrescriptiveStatement key=":art27_2">
    <lrml:hasStrength><lrml:Defeasible/></lrml:hasStrength>
    <ruleml:Rule key=":art27_2_Doctor_Week" closure="universal">
        <ruleml:if>
            <ruleml:Atom>
                <ruleml:Rel iri="#Service_Requirement"/>
                <rulem:Var iri="#Week"/>
            </ruleml:Atom>
        </ruleml:if>
        <ruleml:then>
            <lrml:Permission>
                <ruleml:not>
                    <ruleml:Atom>
                        <ruleml:Rel iri="#Weekly_limit"/>
                        <ruleml:Var iri="#Doctor"/>
                        <ruleml:Var iri="#Week"/>
                    </ruleml:Atom>
                </ruleml:not>
            </lrml:Permission>
        </rulem:then>
    </rulemL:Rule>
</lrml:PrescriptiveStatement>
```

Notice that the first statement sets an obligation in the conclusion (`then` element of the rule) while the second enforce a permission as its conclusion. Given that an obligation and the permission of the opposite conflict, we have to specify which rule takes precedence when both are applicable, this is done by

```
<lrml:Overrides over="#art27_2"  under="#art27_1"/>
```

specifying that the rule for the permission is stronger than the rule for the obligation (and thus, the permission is an exception to the obligation).

The next part is writing statements (rules) to determine the weekly limits. We can use the following constitutive rules:

```
<lrml:ConstitutiveStatement key=":art27_1_c_0">
    <lrml:hasStrength><lrml:Defeasible/></lrml:hasStrength>
    <ruleml:Rule key=":art_27_1_c_0_Doctor_Week" closure="universal">
        <ruleml:then>
            <ruleml:Atom>
                <ruleml:Rel iri="#Weekly_limit"/>
                <ruleml:Var iri="#Doctor"/>
                <ruleml:Var iri="#Week"/>
            </ruleml:Atom>
        </rulem:then>
    <rulemL:Rule>
</lrml:ConstitutivStatement>
```

This rule establishes by default that the number of hours per week for a doctor is within the limit, then the next rule specifies that if the worked hours are in excess of 38 hours, then we are over the limit (the limit is not satisfied)

```
<lrml:ConstitutiveStatement key=":art27_1_c_1">
    <lrml:hasStrength><lrml:Defeasible/></lrml:hasStrength>
    <ruleml:Rule key=":art_27_1_c_1_Doctor_Week" closure="universal">
        <ruleml:if>
            <ruleml:Atom>
              <rulem:Rel>GreaterThan</ruleml:Rel>
                <ruleml:Fun iri="#Weekly_hours/>
                <ruleml:Var iri="#Doctor"/>
                <ruleml:Var iri="#Week"/>
             <ruleml:Con>38</ruleml:Con>
            </ruleml:Atom>
        </ruleml:if>
        <ruleml:then>
            <ruleml:not>
                <ruleml:Atom>
                    <ruleml:Rel iri="#Weekly_limit"/>
                    <ruleml:Var iri="#Doctor"/>
                    <ruleml:Var iri="#Week"/>
                </ruleml:Atom>
            </ruleml:not>
        </rulem:then>
    <rulemL:Rule>
</lrml:ConstitutivStatement>
```

and similarly to what we have done with the prescriptive rules, the second rule overrides the first one.

3.2. From LegalRuleML to DDL and ASP

We have implemented a parser in Python that takes the rules in LegalRuleML and transforms them in DDL and in the format required by the DDL implementation in ASP [23].[3] For example, the rules of the previous section are translated in DDL as

$$Art27.1: \ \Rightarrow_{\mathrm{O}} WeeklyLimit(Doctor, Week)$$
$$Art27.2: \ ServiceRequirement(Week) \Rightarrow_{\mathrm{P}} \neg WeeklyLimit(Doctor, Week)$$
$$Art27.1.c0: \ \Rightarrow_{C} WeeklyLimit(Doctor, Week)$$
$$Art27.1.c1: \ WeeklyHours(Doctor, Week) \geq 38 \Rightarrow_{C} \neg WeeklyLimit(Doctor, Week)$$
$$Art27.2 > Art27.1$$
$$Art27.1.c2 > Art27.1.c1$$

In a situation (week) where there are not service requirement, we conclude that the condition

$$\mathrm{O}\, WeeklyLimit(Doctor, Week)$$

holds, thus, there is obligation to schedule an appropriate number of hours. Then we check if the number of worked hours exceeds 38 or not. If it does, then rule *Art27.1.c1* is applicable, and we conclude $\neg WeeklyLimit(Doctor, Week)$, from which we conclude that there is a violation and that the roster table might not be compliant.

[3]The DDL ASP implementation is avalable at https://github.com/gvdgdo/Defeasible-Deontic-Logic.

For the ASP implementation, given that CLINGO does not support explicit lists, we split each rule in two ASP clauses: one for the label and the conclusion of the rule, the second for its condition of applicability. Thus for a rule

$$r\colon a_1, \ldots, a_1 \Rightarrow_X e$$

we create the clauses

```
<type>Rule(r,e).
applicable(r,e) :- defeasible(a_1), ... , defeasaible(a_n).
```

An important aspect to consider for the formal representation of the rule is that the conclusions of the rules are predicates with arguments, and thus, it is possible to have different values for the variables. Thus, we need to incorporate the argument (more precisely, the primary key of the rules, in the rule label). Moreover, ASP requires the rules to be safe. This means that all variables in the head of the rule must appear in the body of the rule [24]. To accommodate the first requirement we use the `key` attribute to indicate what variables are part of the primary key of the rule. The Python parser from LegalRuleML/DDL to the ASP implementation of DDL examines the variables appearing in the rules, uses the information about the types of the variables to include appropriate predicates in the body of the rules (thus the instances of `doctor(Doctor)` and `week(Week)` in the ASP clauses below); for the full details of how to encode rules in our DDL ASP implementation, see [23].

Accordingly, the rules above we are translated as follow,

```
prescriptiveRule(art27_1(Doctor,Week),weeklyLimit(Doctor,Week)) :-
    doctor(Doctor), week(Week).
applicable(art27_1(Doctor,Week),weeklyLimit(Doctor,Week)) :-
    doctor(Doctor), week(Week).
permissiveRule(art27_2(Doctor,Week),non(weeklyLimit(Doctor,Week))) :-
    doctor(Doctor),  week(Week).
applicable(art27_2(Doctor,Week),non(weeklyLimit(Doctor,Week))) :-
    defeasible(ServiceRequirement(Week)), doctor(Doctor).
superior(art27_2(Doctor,Week),art27_1(Doctor,Week)).
...
constitutiveRule(art27_1_c2(Doctor,Week),non(weeklyLimit(Doctor,Week)) :-
    #sum { Hours,D,W : shift(D,S,W,Hours) } > 38, doctor(D), week(W).
```

An important aspect to consider for the formal representation of the rule is that the conclusions of the rules are predicates with arguments, and thus, it is possible to have different values for the variables. Thus, we need to incorporate the argument (more precisely, the primary key of the rules, in the rule label). Moreover, ASP requires the rules to be safe. This means that all variables in the head of the rule must appear in the body of the rule [24]. To accommodate the first requirement we use the `key` attribute to indicate what variables are part of the primary key of the rule. The Python parser from LegalRuleML/DDL to the ASP implementation of DDL examines the variables appearing in the rules, uses the information about the types of the variables to include appropriate predicates in the body of the rules (thus the instances of `doctor(Doctor)` and `week(Week)` in the ASP clauses above).

There are several reasons for implementing the tools using the ASP implementation of DDL. The first is that it will handle the grounding of the rules, and then it offers an efficient computation of the extension of the program. The second is that we can use the built-in aggregate functions. Thus the function *WeeklyHours*(*Doctor*, *Week*) is implemented by the CLINGO aggregate function

```
#sum { Hours,D,W : shift(D,S,W,Hours) } > 38, doctor(D), week(W).
```

3.3. Data Preprocessing with Python

The next step is to extract the raw roster data from CSV files into a structured format suitable for ASP processing, focusing on standardization, relevant information extraction, and preparation for logical modelling.

Steps:

Data Extraction: A custom Python script systematically parses large-scale CSV files to extract critical scheduling information. This includes:

- Staff identifiers (e.g., doctor names or unique IDs)
- Service types (e.g., emergency, surgery, outpatient)
- Shift timings (start and end times)
- Dates of scheduled shifts

This foundational data forms the basis for all subsequent scheduling analyses and manipulations.

Data Transformation: The extracted data undergoes a transformation process:

- Raw date strings are converted to Python `datetime` objects
- Shift times are normalized to a 24-hour format
- Week numbers are calculated for each shift date
- Holidays and special dates are identified and flagged

This standardization enables precise temporal calculations crucial for compliance checking.

Data Standardization: To ensure consistency, we implement a comprehensive mapping schema that:

- Normalizes service names (e.g., ”ER” and ”Emergency Room” are mapped to a single identifier)
- Standardizes role descriptions across different departments
- Unifies any inconsistent terminologies present in the raw data

This step is vital for ensuring that the logical modeling phase operates on a consistent dataset.

According to the procedure we obtain records with the folowing format

```
doctor(sencla).
...
service(emergency_room).
...
duration(emergegy_room,12).
...
shift(sencla,emergency_room,1,52).
...
```

denoting, respectively, that `sencla` is a doctor, the `emergency-room` is one of the services offered by the hospital, and its duration is 12 hours, and that doctor `sencla` is scheduled to work the emergency room shift on day 1 of week 52.

Another important feature is that it is possible to integrate CLINGO with Python to create external functions and run CLINGO in conjunction with other Python scripts. We have implemented several external functions, in particular for temporal operators (difference of hours, determining what day of the week is a particular date and so on) The integration of these external Python functions significantly enhances the capabilities of our ASP-based roster compliance checking system. By providing complex time calculations, shift analysis, and calendar operations, these functions enable the ASP solver to handle intricate scheduling constraints that would be difficult or impossible to express directly in ASP. This approach combines the declarative power of ASP with the computational flexibility of Python, resulting in a robust and versatile system capable of addressing the complex requirements of medical staff scheduling while ensuring compliance with labour regulations and organizational policies.

3.4. Compliance Checking

The final step of the process is to execute the program. We have tested the implementation with real shift data (1 month) from a hospital. The data included 25 doctors and 17 services offered by the hospital. The compliance tool was able to analyse the data and report compliance violations (this was not unexpected), and it took a few seconds on a laptop computer to generate the model. Most of the execution time was dedicated to grounding the theory. The DDL implementation requires specifying the domain of the application; thus we have to create ASP clauses like

```
atom(shitf(Doctor,Service,Day,Week)):-
    doctor(Doctor), service(Service), day(Day), week(Week).
```

to provide appropriate grounding. Now, types `doctor`, `service`, `day` and `week` have, respectively, 25, 17, 31, and 52 distinct values. Accordingly, during the grounding, over 650,000 ASP atoms are generated. Clearly, most of them are useless, and we plan to use more clever Python scripts to minimise the number of atoms that need to be created and to improve the response time. Notice, however, that on a laptop computer, the grounding for a year of data takes between 2 and 3 seconds, and the evaluation of the full program (with rule encoding the CCNL, between 10 and 15 seconds). In addition, a roster must be created every month; thus we can restrict the grounding and evaluation to the data for one month, and such a computation takes approximately less than 2 seconds. For the monthly computation, a few additional predicates must be created, and the result of their computation stored and updated every month (this comports a negligible overhead for each single monthly computation, but with a significant saving over the computation for an entire year).

At the same time, to test the correctness of the rules and implementation, we created a few instances of non-compliant shifts. The tool reported non-compliance issues. Notice that the ASP DDL implementation [23] has as built-in `violation` predicates, and it is easy to use them to build customised violation predicates to report specific noncompliance issues, for example:

```
violazione(Doctor,Week_limit,Max,Worked) :-
    violation(art27_1(Doctor,Week),week_limit(Doctor,Week),1),
    defeasible(max_hours(Max,Doctor,Week)),
    worked_hours(Worked,Doctor,Week).
```

4. Summary and Future Work

The issue of rostering medical staff (and other domain) has been the focus of research in operational research and constraints programming for a long time (see among other [25, 26, 27, 28, 29, 30, 31]). Despite the success in research, based on our personal experience, there seems to be little uptake in practice, with a few attempts on automated rostering in recent years [32, 33, 34]. Most of the models are too complex for hospital staff to use or do not provide useful features for physicians, and are extremely hard to customise [35]. In addition, we are not aware of approaches taking into consideration the relevant legislative framework governing the work conditions.

This study presents a comprehensive framework for automating the scheduling of medical staff using a combination of Python for data preprocessing and Answer Set Programming (ASP) with CLINGO for logical modeling. The system is designed to address the intricate challenges of scheduling in healthcare settings by integrating advanced computational tools and methodologies to enhance efficiency, compliance, and fairness in shift allocation.

The implementation demonstrates that the use of Python to standardize and prepare data, coupled with the declarative nature of ASP for rule-based logic modeling, effectively creates a robust and scalable solution. This approach not only streamlines the scheduling process but also ensures that all operational, legal, and personal constraints are systematically addressed and adhered to. Moreover, the integration of external functions allows for real-time adjustments and validations, further enhancing the system's adaptability to dynamic operational requirements.

The outcomes of this study indicate significant improvements in scheduling efficiency and compliance with legal standards. By automating the rostering process, healthcare facilities can potentially reduce administrative overhead, minimize human error, and better allocate resources to direct patient care. This shift in resource management could lead to reduced waiting times for patients and improved job satisfaction among staff due to fairer and more transparent shift distributions.

Accordingly, we plan to extend our follow the approach by Dodaro and coworkers [30, 31], and to use ASP as a constraint solver to generate legally valid roster incorporating the ASP encoding of the CCNL requirements.

Future Directions: While the current system provides a solid foundation for staff scheduling, several avenues for future research and development emerge:

- **Integration with Hospital Information Systems:** Future iterations could explore deeper integration with existing hospital information systems to automate data flow and further reduce manual interventions (e.g., by automatically recruiting staff with their qualifications, rest, illness, recruitment etc.; or by linking with the patient booking system).
- **Advanced Machine Learning Techniques:** Implementing machine learning could offer predictive insights into staffing needs, taking into account historical data, trends, and even predicting future demand spikes.
- **User Interface Development:** Developing a user-friendly interface for schedule management could enhance accessibility and usability for administrative staff, allowing for easier adjustments and real-time feedback (e.g., in the event of machinery maintenance, it may be feasible to open temporary rooms with different hours than usual).
- **Expansion to Other Roles:** Extending the framework to include other hospital staff categories, such as nursing and administrative personnel, could help in achieving a more integrated and holistic scheduling system across the entire institution.
- **Cross-Facility Scheduling Solutions:** For healthcare networks, developing solutions that optimize staff allocation across multiple facilities could further enhance efficiency and resource utilization (e.g., in Italy, health territorial facilities its made by one main hospital, maybe one or more minor hospitals and fixed or temporary hubs scattered on the territory. The staff group is just one and the rosters must take into account the demands of patients on the territory and different buildings).

In conclusion, the deployment of AI and logical programming in healthcare scheduling represents a significant advancement in the management of medical staff. The successful implementation of this system sets a precedent for further research into automated solutions that are not only compliant with regulatory frameworks but also responsive to the complex dynamics of healthcare environments.

As a future work, we want to upgrade this solution with an AI tool able to automatically create compliant rosters.

In a view of high level optimization hospital processes and cascading beneficial effects [36], the aim of this research is to provide a tool able to release the highly qualified medical staff from a purely administrative task. In this way, these changes will allow highly qualified medical staff to spend more time with patients and reduce waiting lists. Finally, even if the presented paper shows an Italian example, therefore, with Italian legislation, the framework is usable for all existing hospitals taking into account the legislation or needs of the other country.

Acknowledgments

This study was funded in the context of the European Digital Innovation Hub (EDIH) for the Healthcare Digital and AI support and innovation (Circular Health European Digital Innovation Hub - CHEDIH) (www.chedih.eu) in Piedmont Region, Italy.

References

[1] R. L. Burdett, E. Kozan, An integrated approach for scheduling health care activities in a hospital, European Journal of Operational Research 264 (2018) 756–773.

[2] J. Munoz-Gama, N. Martin, C. Fernandez-Llatas, O. A. Johnson, M. Sepúlveda, E. Helm, V. Galvez-Yanjari, E. Rojas, A. Martinez-Millana, D. Aloini, et al., Process mining for healthcare: Characteristics and challenges, Journal of Biomedical Informatics 127 (2022) 103994.

[3] I. A. Amantea, E. Sulis, G. Boella, F. De Marchi, L. Mazzini, F. Alloatti, A. Bolioli, Adopting assistive technologies in healthcare processes: a chatbot for patients with amyotrophic lateral sclerosis, in: Italian forum of ambient assisted living, Springer, 2020, pp. 163–174.

[4] I. A. Amantea, E. Sulis, G. Boella, R. Marinello, M. Grosso, A. Crespo, A modeling framework for an innovative e-health service: the hospital at home, in: Simulation and Modeling Methodologies, Technologies and Applications: 10th International Conference, SIMULTECH 2020 Lieusaint-Paris, France, July 8-10, 2020 Revised Selected Papers 10, Springer, 2022, pp. 111–132.

[5] F. De Marchi, I. A. Amantea, M. Serioli, E. Sulis, G. Boella, F. Alloatti, A. Bolioli, S. Riso, R. Cantello, L. Mazzini, E-health solutions for amyotrophic lateral sclerosis patients: A chatbot for dietary monitoring, Journal of the Neurological Sciences 429 (2021).

[6] M. Hashmi, G. Governatori, H.-P. Lam, M. T. Wynn, Are we done with business process compliance: State-of-the-art and challenges ahead, Knowledge and Information Systems 57 (2018) 79–133.

[7] H. Groesfema, N. van Beest, G. Governatori, On the use of the conformance and compliance keywords during verification of business processes, in: BPM Forum 2022, volume 458 of *LNBPI*, Springer, 2022, pp. 21–37.

[8] G. Governatori, A. Rotolo, G. Sartor, Logic and the law: Philosophical foundations, deontics, and defeasible reasoning, in: D. M. Gabbay, J. Horty, X. Parent, R. van der Meyden, L. van der Torre (Eds.), Handbook of Deontic Logic and Normative Reasoning, volume 2, College Publications, London, 2021, pp. 655–760.

[9] T. F. Gordon, G. Governatori, A. Rotolo, Rules and norms: Requirements for rule interchange languages in the legal domain, Number 5858 in LNCS, Springer, Heidelberg, 2009, pp. 282–296.

[10] H. Lam, M. Hashmi, Enabling reasoning with LegalRuleML, Theory Practice of Logic Programming 19 (2019) 1–26.

[11] G. Antoniou, D. Billington, G. Governatori, M. J. Maher, On the modeling and analysis of regulations, in: Proceedings of the Australian Conference Information Systems, 1999, pp. 20–29.

[12] G. Governatori, Representing business contracts in RuleML, International Journal of Cooperative Information Systems 14 (2005) 181–216.

[13] M. Dastani, G. Governatori, A. Rotolo, L. van der Torre, Programming cognitive agents in defeasible logic, in: G. Sutcliffe, A. Voronkov (Eds.), LPAR 2005, volume 3835 of *LNAI*, Springer, Heidelberg, 2005, pp. 621–636.

[14] J. Dimyadi, G. Governatori, R. Amor, Evaluating legaldocml and legalruleml as a candidate standard for sharing normative information in the aec/fm domain, in: F. Bosché, I. Brilakis, R. Sacks (Eds.), LC3 2017: Volume I Proceedings of the Joint Conference on Computing in Construction (JC3), 2017, pp. 639–646.

[15] H. Bhuiyan, G. Governatori, A. Rakotonirainy, A. Bond, Traffic rules compliance checking of automated vehicle maneuvers, Artificial Intelligence and Law 32 (2024) 1–56.

[16] I. A. Amantea, L. Robaldo, E. Sulis, G. Governatori, G. Boella, Business process modelling in healthcare and compliance management: a logical framework, Journal of Applied Logics—IfCoLog Journal of Logics and their Applications 9 (2022).

[17] G. Governatori, P. Casanovas, L. de Koker, On the formal representation of the australian spent conviction scheme, in: V. Gutiérrez Basulto, T. Kliegr, A. Soylu, M. Giese, D. Roman (Eds.), Rules and Reasoning, volume 12173 of *LNCS*, Springer International Publishing, Cham, 2020, pp. 177–185.

[18] M. Hashmi, G. Governatori, M. T. Wynn, Normative requirements for regulatory compliance: An abstract formal framework, Information Systems Frontiers 18 (2016) 429–455.

[19] G. Governatori, F. Olivieri, A. Rotolo, S. Scannapieco, Computing strong and weak permis-

sions in defeasible logic, Journal of Philosophical Logic 42 (2013) 799–829. doi:10.1007/s10992-013-9295-1.
[20] G. Brewka, T. Eiter, M. Truszczyński, Answer set programming at a glance, Communications of the ACM 54 (2011) 92–103. doi:10.1145/2043174.2043195.
[21] M. Palmirani, G. Governatori, T. Athan, H. Boley, A. Paschke, A. Wyner, LegalRuleML Core Specification Version 1.0, OASIS Committee Specification 2, OASIS, 2020. URL: https://docs.oasis-open.org/legalruleml/legalruleml-core-spec/v1.0/cs02/legalruleml-core-spec-v1.0-cs02.html.
[22] T. Athan, H. Boley, G. Governatori, M. Palmirani, A. Paschke, A. Wyner, Oasis legalruleml, in: E. Francesconi, B. Verheij (Eds.), Proceedings of the Fourteenth International Conference on Artificial Intelligence and Law, ACM, New York, 2013, pp. 3–12. doi:10.1145/2514601.2514603.
[23] G. Governatori, An ASP implementation of defeasible deontic logic, Künstliche Intelligenz (2024). doi:10.1007/s13218-024-00854-9.
[24] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Answer Set Solving in Practice, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2012. doi:10.1007/978-3-031-01561-8.
[25] L. Gierl, B. Pollwein, G. Heyde, H. Kurt, Knowledge-based scheduling of duty rosters for physicians, Medical Informatics 18 (1993) 355–366.
[26] A. Caprara, F. Focacci, E. Lamma, P. Mello, M. Milano, P. Toth, D. Vigo, Integrating constraint logic programming and operations research techniques for the crew rostering problem, Software – Practice and Experience 28 (1998) 49–76.
[27] S. Rosenberg, Programming medical rosters in Prolog, Medical Informatics 13 (1988) 187–198.
[28] G. B. Lamont, H. Haddad, G. A. Papadopoulos, B. Panda, H. Li, A. Lim, B. Rodrigues, A hybrid AI approach for nurse rostering problem, Proceedings of the 2003 ACM symposium on Applied computing (2003) 730–735.
[29] F. Morgado, R. L. Saldanha, J. Roussado, L. Albino, E. Morgado, J. P. Martins, Using AI Local Search to Improve an OR Optimizer, Proceedings of the AAAI Conference on Artificial Intelligence 26 (2012) 2237–2244.
[30] C. Dodaro, G. Galatà, M. Maratea, I. Porro, An ASP-based framework for operating room scheduling, Intelligenza Artificiale 13 (2019) 63–77.
[31] C. Dodaro, G. Galatà, M. K. Khan, M. Maratea, I. Porro, Operating Room (Re)Scheduling with Bed Management via ASP, Theory and Practice of Logic Programming 22 (2021) 229–253.
[32] C. N. Gross, J. O. Brunner, M. Blobner, Hospital physicians can't get no long-term satisfaction – an indicator for fairness in preference fulfillment on duty schedules, Health Care Management Science 22 (2019) 691–708.
[33] K. O'Callahan, S. Sitters, M. Petersen, 'You make the call': Improving radiology staff scheduling with AI-generated self-rostering in a medical imaging department, Radiography 30 (2024) 862–868. doi:10.1016/j.radi.2024.03.014.
[34] S. Agrawal, J. Turner, Using AI to build the foundation of better capacity management, Management in Healthcare: A Peer-Reviewed Journal 7 (2023) 194.
[35] M. Tuffaha, M. R. Perello-Marin, Adoption Factors of Artificial intelligence in Human Resources Management, Future of Business Administration 1 (2022) 1–12. doi:10.33422/fba.v1i1.140.
[36] I. Amantea., M. Quaranta., Eco-sustainability and efficiency of healthcare complex systems, in: Proceedings of the 14th International Conference on Simulation and Modeling Methodologies, Technologies and Applications - SIMULTECH, INSTICC, SciTePress, 2024, pp. 423–430. doi:10.5220/0012857400003758.