UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI MATEMATICA E INFORMATICA

DOTTORATO DI RICERCA
XXXIII CICLO

SETTORE SCIENTIFICO DISCIPLINARE INF/01 - INFORMATICA

TESI DI DOTTORATO

A LOGIC-BASED DECISION SUPPORT SYSTEM FOR THE DIAGNOSIS OF HEADACHE DISORDERS ACCORDING TO THE ICHD-3 INTERNATIONAL CLASSIFICATION

ROBERTA COSTABILE

Roberta Costabile

SUPERVISORE

Prof. Marco Manna

[illegible]

COORDINATORE

Ch.mo Prof. Gianluigi Greco

GIANLUIGI
GRECO
13.04.2022
09:05:39 UTC

Novembre 2017 - Marzo 2021

CONTENTS

ABSTRACT

Decision support systems play an important role in medical fields as they can augment clinicians to deal more efficiently and effectively with complex decision-making processes. In the diagnosis of headache disorders, however, existing approaches and tools are still not optimal. On the one hand, to support the diagnosis of this complex and vast spectrum of disorders, the International Headache Society released in 1988 the *International Classification of Headache Disorders* (ICHD), now in its 3rd edition: a 200 pages document classifying more than 300 different kinds of headaches, where each is identified via a collection of specific nontrivial diagnostic criteria. On the other hand, the high number of headache disorders and their complex criteria make the medical history process inaccurate and not exhaustive both for clinicians and existing automatic tools. To fill this gap, we present HEAD-ASP, a novel decision support system for the diagnosis of headache disorders. Through a REST Web Service, HEAD-ASP implements a dynamic questionnaire that complies with ICHD-3 by exploiting two logical modules to reach a complete diagnosis while trying to minimize the total number of questions being posed to patients. Finally, HEAD-ASP is freely available on-line and it is receiving very positive feedback from the group of neurologists that is testing it.

1

INTRODUCTION

1.1 Context and State-of-the-Art

The thesis work is a contribution to an experimental development and industrial research project, whose name is Alcmeone, funded by the Italian Ministry of Economic Development. The aim of the project is to provide an innovative organizational and management model, and an advanced technological platform of services for supporting the integrated clinical management of headache patients. This work, therefore, focuses on the management of headache disorders in the context of decision support systems. *Decision Support System* (DSS) have been conceived for providing the "information and analysis necessary for the decisions that must be made" [Don76]. After almost 50 years, DSSs are still evolving, and they play an important role in various application domains, in particular in the medical field, as they can help clinicians to deal more efficiently and effectively with complex decision-making processes such as diagnostics, disease management, and drug control [SPB+20]. Since the seventies, a lot has been done both in medicine and computer science to make DSS more and more robust and reliable. But in some specific fields, such as the diagnosis of headache disorders, existing approaches and tools are still not optimal. Headache disorders represent one of the most common and disabling conditions of the nervous system throughout the world [SHJ+07]. In particular, about 90% of all headaches are primary, namely, magnetic resonance imaging of the brain reveals no abnormality [Eva17].Many of them are severe enough to compromise, even very significantly, the quality of life of those who are affected.

The diagnostic evaluation of such disorders is notoriously one of the most difficult, since they are devoid of biochemical or neuroradiological markers. Therefore, their evaluation is mainly based on the description of symptoms by the patient, that is, on what in medicine is the collection, by the doctor, of information on the patient's clinical history. The elements to be taken into account are extremely varied, and the set of possible headache disorders, so far recognized by the medical community, is also very extended. To support the diagnosis of this complex and vast spectrum of disorders, in 1988, the *International Headache Society* [IHS18] released the first edition of the *International Classification of Headache Disorders* (ICHD), now in its 3rd edition: a 200-page document classifying, in a taxonomic way, more than 300 different kinds of headaches, and where each single form of headache is identified via a collection of specific nontrivial diagnostic criteria (see Figure 1.1). Due to the complexity and vastness of the considered domain, the medical history process may be inaccurate and not exhaustive, thus, it is of paramount importance, in this specific

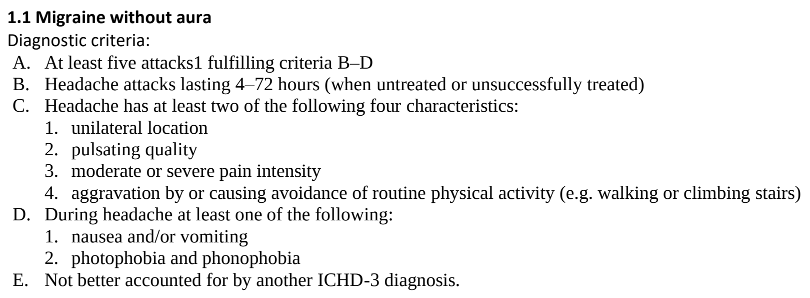

1.1 Migraine without aura
Diagnostic criteria:
A. At least five attacks1 fulfilling criteria B–D
B. Headache attacks lasting 4–72 hours (when untreated or unsuccessfully treated)
C. Headache has at least two of the following four characteristics:
 1. unilateral location
 2. pulsating quality
 3. moderate or severe pain intensity
 4. aggravation by or causing avoidance of routine physical activity (e.g. walking or climbing stairs)
D. During headache at least one of the following:
 1. nausea and/or vomiting
 2. photophobia and phonophobia
E. Not better accounted for by another ICHD-3 diagnosis.

Figure 1.1: Diagnostic criteria of an ICHD-3 diagnosis.

medical field, to support clinicians and specialists during the entire diagnostic phase, in order to improve disease management.

As said, a number of approaches in this domain have been already proposed in the literature. The most related ones are briefly discussed next. De Simone et al. (2004) developed *AIDA Cefalee*, a system consisting of a database for the storage of symptoms and diagnostic data of patients paired with a module that can suggest possible diagnosis but only when all symptoms have been acquired [DSMB04]. In particular, the database can be synchronized over the network allowing a continuous sharing of the patients' information and a cooperation between different research groups. The diagnostic tool has been validated experimentally but no details of the classification method are provided. Simìc et al. (2008) presented a novel tool that makes use of *rule-based fuzzy logic* but is limited to a few forms of disorders [SSSSI08]. The researchers showed the workflow of the basic rule-based fuzzy logic systems model in which the rules are expressed as a collection of if-then statements. In particular, the information can be extracted by the patients in the form of if-then statements and these rules can be modeled using a fuzzy logic system; once the rules are provided to the system, it can be viewed as an input-to-output mapping. Eslami et al. (2013) proposed a DSS implementing a dynamic questionnaire, but neither the system is publicly available nor the underlying classification method is described [EREHN+13]. In particular, the system provides questions related to headache disorders and, eventually, derives the most appropriate type of headache using simple human-like algorithmic logic; the accuracy of the diagnosis depends also on the accuracy of the patient's response. Dong et al. (2014) proposed a general architecture of a DSS based on the ICHD-3 classification [DYH+14]. In particular, the researchers' work is based on a 3-steps translation of the ICHD-3: first in terms of flow-charts, then in terms of an ontological model and, finally, in terms of rules. The system is described mostly from an architectural point-of-view and in-depth details of the translation and of the diagnostic process are not provided. Vandewiele et al. (2018) proposed a DSS based on machine learning which generates an interpretable predictive model from the collected data [VDBL+18]. In particular, the system consists of three modules: a mobile application that captures symptomatic data from patients; an automated diagnosis support module that generates an interpretable decision tree, based on data semantically annotated with expert knowledge; and a web application that helps the clinicians to interpret captured data and learned insights by means of visualizations. The diagnostic process is based on supervised machine learning models.

1.2 Motivation and Objectives

From the above overview, it should be already clear that none of the existing systems provides, at the same time, (i) the same level of accuracy required by ICHD-3, (ii) a solid, extensible and open knowledge representation model for fully and faithfully representing the ICHD-3 criteria, (iii) a dynamic questionnaire to support clinicians during the entire diagnostic phase, and (iv) an optimization strategy to minimize the number of questions posed to patients. Moreover, (v) none of the aforementioned systems is made available to be tested or used.

To make considerable steps forward in the diagnosis and management of headache disorders, the Italian Ministry of Economic Development appreciated and funded the research project *Alcmeone*, whose aim has been described in Section 1.1. In particular, concerning the headache diagnosis, the goal is to develop a *Decision Support System* (DSS) that meets the following five main project specifications.

1. strictly represent ICHD-3 information, structure and criteria;
2. focus on primary headaches, namely, on the first four chapters of the international classification;
3. implement an interactive questionnaire that rigorously guides both clinicians and patients during the medical history process;
4. reach a complete diagnostic picture of each patient by marking each primary headache diagnosis as compatible or not compatible;
5. keep reasonably low the number of questions posed to patients during the medical history process.

Driven by the lack of effective tools in the headache domain and by the project specifications, this thesis work has consisted of the development of HEAD-ASP, a novel DSS for the diagnosis of headache disorders, according to their classification provided within the most recent version of the ICHD.

Table 1.1 shows a comparative analysis of the tools existing in literature, including HEAD-ASP, with respect to some important parameters, highlighting the approach used for the development of each system. Regarding the type of questionnaire implemented, it is worth noting that *static* means the patient has to answer all possible questions in some order, whereas *dynamic* means the patient has to answers a subset of questions dynamically proposed by the system, thus, the length of the questionnaire may vary. Moreover, in the table are also shown the reference ICHD version, if the system has been verified and validated and if it is available online.

System	*Approach*	*Latest Release*	*Questionnaire Type*	*ICHD Version*	*Verification & Validation*	*Online Access*
AIDA CEFALEE	Flowcharts	2007	Static	ICHD-2	✓	✗
SIMIC ET AL.	Fuzzy Logic	2008	Dynamic	ICHD-2	✗	✗
ESLAMI ET AL.	Algorithmic Logic	2013	Dynamic	ICHD-2	✓	✗
DONG ET AL.	Logic Programming	2014	Static	ICHD-3	✓	✗
VANDEWIELE ET AL.	Supervised Machine Learning	2018	Static	ICHD-3	✓	✗
HEAD-ASP	Logic Programming (ASP)	2020	Dynamic	ICHD-3	✓ (still in progress)	✓

Table 1.1: Comparison between HEAD-ASP and the current state of the art systems.

1.3 Challenges and Contribution

During the development of the system, we faced three main technical challenges: designing a knowledge representation model being able to accommodate domain medical knowledge (often implicit in ICHD-3) together with a natural and formal encoding of the ICHD-3 diagnoses (among the most complex in the medical field); designing a standard methodology to encode ICHD-3 diagnoses into logical rules over the data model mentioned above; and designing an efficient and effective heuristics offering a good trade-off between the average number of questions and the time required to determine the next question.

After more than two years of work, HEAD-ASP does fulfill all the aforementioned desiderata and project specifications. From a technological viewpoint, the system consists of a REST Web service implementing a dynamic and interactive questionnaire that supports clinicians during the diagnostic phase. From a knowledge representation and reasoning perspective, the Web service encapsulates and manages two formal logical modules expressed in *Answer Set Programming* (ASP): the first one is a deductive module that faithfully encodes all primary headache diagnoses and criteria of ICHD-3, whereas the second one is an optimization module for minimizing the number of questions that are necessary to complete the diagnostic picture of patients.

Overall, I believe that HEAD-ASP fully meets the real needs in this domain. This has been made possible thanks to the adoption of a declarative knowledge representation formalism and to a close interaction between clinicians and computer scientists. Indeed, although it is still a research prototype, it is receiving very positive feedback from the group of neurologists that are testing and using it within *Alcmeone*. Some statistics on its effectiveness are reported in Section 6.2. The system is freely available on-line at `https://head-asp.github.io/ichd-dss/`.

1.4 Structure of the Thesis

The thesis is organized into three parts:

- In the first part we present an overall introduction of the main concepts that will be discussed in this work. In particular, in Chapter 2 the guidelines will be analyzed in detail identifying the essential aspects of the classification from a structural point of view. Since both the deductive module that encodes all diagnoses and criteria and the optimization module that implements the questionnaire are expressed in ASP, Chapter 3 will briefly introduce declarative languages focusing on *Answer Set Programming*. We will formalize its syntax and semantics and we will provide some example of known *Knowledge Representation and Reasoning* (KRR) problems solved using ASP.

- In the second part of this work we focus on the design and implementation of the HEAD-ASP system. More in detail, Chapter 4 will describe the designing of the knowledge representation model being able to accommodate domain medical knowledge and of the formal methodology to encode diagnoses into ASP logical rules; Chapter 5 will focus on the description of the logical decision module analyzing all the stages of the ASP encoding that implements the questionnaire. Eventually, Chapter 6 will present the HEAD-ASP system from a technological perspective, i.e., it will report about the system architecture and its implementation. Furthermore, it will present the results obtained during a testing phase run to verify and to investigate the performance of the approach.

- The thesis is closed with a third part, in which (i) we describe some of the most recent works concerning decision support systems for the diagnosis of headache that can be found in literature (Chapter 7), and (ii) we draw conclusions outlining possible future work (Chapter 8).

Part I

PRELIMINARY NOTIONS AND NOTATIONS

Roberta Costabile

2 THE ICHD CLASSIFICATION

In this chapter we analyze the guidelines in detail identifying the essential aspects of the classification from a structural perspective.

2.1 History and Objectives

The *International Classification of Headache Disorders* (ICHD) is a detailed hierarchical classification of all headache-related disorders published by the International Headache Society. It is considered the official classification of headaches by the World Health Organization, and, in 1992, was incorporated into the 10th edition of their International Classification of Diseases (ICD-10). Each class of headache contains explicit diagnostic criteria, meaning that the criteria include quantities rather than vague terms like several or usually, that are based on clinical and laboratory observations.

ICHD-3 is the document that contains the guidelines for the diagnosis of headaches. It describes the diagnostic criteria for all known headache disorders. The original document, written in English, has been translated into various languages. The Italian translation is currently only available for its third beta edition and has not yet been updated for ICHD-3. ICHD-3 was published as the first issue of Cephalalgia in 2018, exactly 30 years after the first edition of the International Classification of Headache Disorders, ICHD-I. This first version was based primarily on expert opinion, but still proved largely valid. ICHD-II, published in 2004, included a number of changes, partly resulting from new evidence and partly from revised expert opinions. The new scientific evidence has played an even more important role in the changes that led to ICHD-3 beta and all further changes included in ICHD-3 are based on that evidence. The classification of headaches will, therefore, in the future, be guided entirely by research. ICHD-3 reports a systematic classification with explicit diagnostic criteria for each entity of the disorder. It is very broad and is not intended to be learned by heart; even for the members of the classification committee it is difficult to remember all its contents. It is a document designed to be consulted several times and, in clinical practice, it is useful when the diagnosis is uncertain.

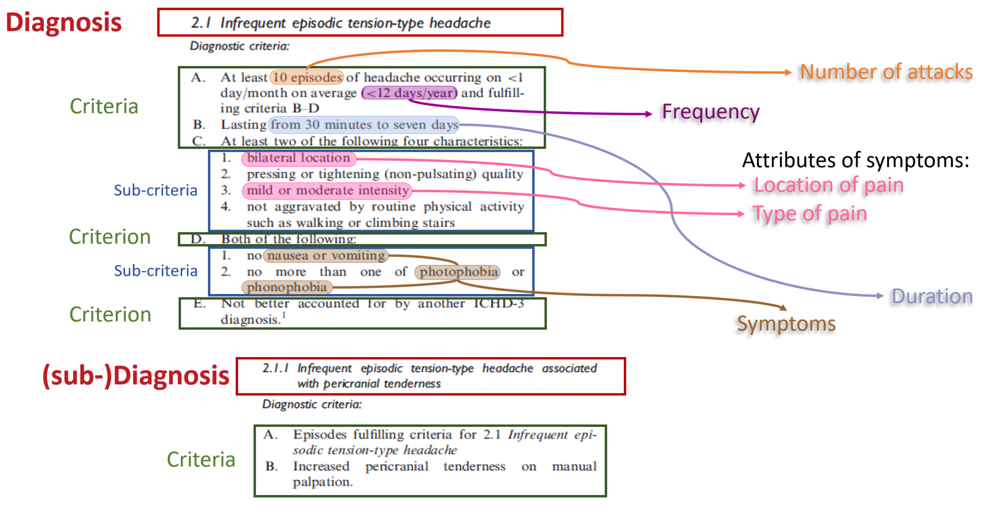
2.2 Structure and Content Organization

International Classification of Headache Disorders - 3rd Edition (ICHD-3) provides specific criteria, defined in natural language, for diagnosing known headaches. The possible diagnoses are organized in a hierarchical structure that expresses the existing relations between them. Each type of headache diagnosis includes its own sub-categories, that is, other more specific types of headache, which correspond to a higher level of detail.

The classification consists of 14 chapters grouped into 4 parts. This work focuses on primary headaches (part 1, chapters 1–4) according to the project specifications reported above. Anyway, the designed methodology can be definitely also applied to encode the rest of the diagnoses since they do not differ substantially in the structure.

Each chapter concerning primary headaches collects diagnoses related to a particular type of headache: *Migraine*, *Tension-type headache (TTH)*, *Trigeminal autonomic cephalalgias (TACs)* and *Other primary headache disorders*. In the following, we describe the structural aspects of the diagnoses, and then we report the main notions that underlie their content.

The *diagnosis* represents the fundamental structural unit of ICHD-3. Each diagnosis is identified by a set of *criteria* that appear within a list marked with letters ("A","B",...). Each criterion includes a series of requirements framed within the symptomatic state of the diagnosis the criterion refers to. A diagnosis is considered *compatible* if, considering the ailments the patient suffers from, the conditions expressed by all its criteria are met. A criterion can be presented in a monothetic or polythetic form. A criterion is considered monothetic when it identifies a set of specific requirements. The necessary condition for it to be validated is that all its requirements are met. A criterion is considered polythetic when it consists of requirements that appear in an enumerative list format within its own statement. To make such type of criterion as simple as possible, we assign the meaning of *sub-criterion* to each set of requirements identified by an element of this numbered list (thus, each sub-criterion is marked with a number). The necessary condition for the validation of a polythetic criterion is to verify a minimum number of sub-criteria on the basis of a fixed inclusion threshold, as shown in Figure 1.1 of Chapter 1. A systematic analytical phase was necessary to identify the main notions of a diagnosis (see Figure 2.1). At the basis of ICHD-3 there is the notion of *symptom*; it can be identified as a key notion because it is involved in the criteria of all the diagnoses of primary headaches. Throughout the systematic analysis of ICHD-3 contents, we extracted the *attributes* associated with the symptoms: (i) *location of pain* (unilateral, bilateral, etc.); (ii) *aggravating factors* that worsen the pain (such as the movement, etc.) and any *limitations* caused by pain (such as perform routine physical activities); and (iii) *type of pain* associated with headache (pulsating, intense, etc.). Furthermore, symptoms can be also characterized by: (i) *duration*, meant as the persistence over time of the pain caused by a symptom; (ii) *frequency* of pain attacks, i.e., the number of times the pain caused by a symptom occurs over a specified length of time (such as how many times a day, how many days a month the pain occurs) and, moreover, the (continuous) time interval in which a certain frequency lasts (as an example "headache occurs on 1-14 days/month on average for >3 months"); (iii) *number of attacks* that affect the patient; and (iv) information relating to the report of a particular *clinical exam* previously done by the patient.


2.1 Infrequent episodic tension-type headache

Diagnostic criteria:

A. At least 10 episodes of headache occurring on <1 day/month on average (<12 days/year) and fulfilling criteria B–D
B. Lasting from 30 minutes to seven days
C. At least two of the following four characteristics:
   1. bilateral location
   2. pressing or tightening (non-pulsating) quality
   3. mild or moderate intensity
   4. not aggravated by routine physical activity such as walking or climbing stairs
D. Both of the following:
   1. no nausea or vomiting
   2. no more than one of photophobia or phonophobia
E. Not better accounted for by another ICHD-3 diagnosis.[1]

2.1.1 Infrequent episodic tension-type headache associated with pericranial tenderness

Diagnostic criteria:

A. Episodes fulfilling criteria for 2.1 *Infrequent episodic tension-type headache*
B. Increased pericranial tenderness on manual palpation.

Figure 2.1: Identification of the notions which characterize the diagnostic criteria.

3 ANSWER SET PROGRAMMING

Logic Programming (LP) is a high level and human oriented programming paradigm, primary based on formal logic, to describe problems to computers without specifying how to solve them. Namely, LP's purpose is to make machines able to solve a problem by describing the problem themselves, lifting the programmers from the burden of defining how to solve it. In particular, in LP a single formalism has to suffices for both logic and computation, and that logic subsumes computation [KB83]. This aim is achieved by representing a given computational problem by means of a logic program whose intended models correspond to solutions, which can be found using a solver [VEK76, Llo12, Kow88, Lif99].

Around the 1950s, John McCarthy [LGP$^+$90] discussed how logic is particularly suited to be a full-fledged declarative programming paradigm, allowing to model problems in a natural and human-oriented fashion, and effectively represent knowledge representation and rational human reasoning. In the same years, a new computer science field was born: AI, and logic-based languages gained more and more importance and popularity. A breakthrough happened when Alain Colmerauer and its research group introduced Prolog [CR96] (from the French, PROgrammation en LOGique), the first logic programming language. However, it emerged that the first-order logic on which Prolog is based is not capable of modelling the commonsense human reasoning, which is non-monotonic: we as humans, starting from some premises, may rationally regret them whenever new information become available, while in first-order logic, logical consequences cannot be invalidated since the underlying reasoning is monotonic. Subsequently, new logic formalisms devoted to represent non-monotonic reasoning were introduced, such as Default Logic [Rei80], Autoepistemic Logic [Moo85] and Circumscription [McC80]. In the late '80s and early '90s, Michael Gelfond and Vladimir Lifschitz presented the logic formalism Answer Set Programming (ASP) [GL88, GL91] allowing to express non-monotonic reasoning in purely declarative fashion [BET11, EFLP00, EIK09a, GL91, MT99, Nie99]. ASP became widely used in AI and recognized as a powerful tool for KRR.

3.1 Syntax

Let $\mathscr{I}$ be a set of *identifiers*. An identifier is a non-empty string starting with some lowercase letter and containing only alphanumeric symbols and the symbol "_" (underscore).

Example 3.1.1. Examples of identifiers are: a, $a1\_B$, a_ID, $vertex$

3.1.1 Terms

A term is either a *constant*, a *variable*, an *arithmetic term* or a *functional term*. In particular, constants and variables can be considered as "basic terms", while arithmetic and functional terms are defined inductively as combinations of terms.

Definition 3.1.1 (Constant Term)**.** A *constant* is either a *symbolic constant*, if it is an identifier, a *string constant*, if it is a quoted string, or an *integer*.

Definition 3.1.2 (Variable Term)**.** A *variable* is a non-empty string starting with some uppercase letter and containing only alphanumeric symbols and the symbol "_" (underscore).

Furthermore a special variable, namely *anonymous variable*, is represented by the symbol "_" (underscore). This syntactic shortcut is intended to indicate a *fresh* variable, that does not appear elsewhere in the context in which it is located.

Definition 3.1.3 (Arithmetic Term)**.** An *arithmetic term* has the form $\text{-}(t)$ or $(t_1 \diamond t_2)$ for terms t_1 and t_2 with $\diamond \in \{+, -, *, /\}$. Parentheses can optionally be omitted, and standard operator precedences apply.

Definition 3.1.4 (Functional Term)**.** A *functional term* has the form $f(t_1, \dots, t_n)$, where f is an identifier, known as *functor*, $t_1, \dots, t_n$ are terms and $n > 0$.

Example 3.1.2. Examples of terms are:

– Constants: a, x, "*http://google.com*", 0, 123

– Variables: X, X_134, $X2$, $Color$

– Arithmetic terms: $\text{-}X$, $X+Y$, $2*(\text{-}5)$, $X+ab$, $X/3$

– Functional terms: $f(X)$, $father(aristotle)$, $g(2*5,$"*abc*"$)$

A term is *ground* (i.e., variable-free) if it does not contain any variable. For, instance in Example 3.1.2 all the constants, the arithmetic term $2*(\text{-}5)$ and the functional terms $father(aristotle)$ and $g(2*5,$"*abc*"$)$ are ground.

3.1.2 Atoms and Literals

Definition 3.1.5 (Predicate)**.** Given an identifier p and an integer n with $n \geq 0$, the expression p/n represents a *predicate*. p is said *predicate symbol* and n represents the associated arity.

Example 3.1.3. Examples of predicates are: $a/2$, $p/3$, $predicate\_3/1$, $true/0$.

In the following, when no ambiguities arise we denote a predicate p/n simply as p.

Definition 3.1.6 (Predicate Atom)**.** A *predicate atom* has the form $p(t_1,\dots,t_n)$, where $n \geq 0$, p/n is a predicate with predicate name p and arity n and $t_1,\dots,t_n$ are terms; if $n = 0$, parenthesis are omitted and the notation p is used.

Definition 3.1.7 (Classical Atom)**.** A *classical atom* is either $-a$ or a where a is a predicate atom and $-$ denotes the *strong negation* symbol.

Definition 3.1.8 (Built-in Atom)**.** A *built-in atom* has the form $t_1 \rhd t_2$ where t_1, t_2 are terms and $\rhd \in \{<, <=, =, <>, != , >, >=\}$.

Definition 3.1.9 (Naf-Literal)**.** A *naf-literal* can either be a built-in atom or have form a or *not* a where a is a classical atom, and *not* is the *negation as failure* symbol.

Example 3.1.4. Some examples are shown below.

- Predicate Atoms: $edge(X,Y)$, $atom(f(a,b),c)$, $true$
- Classical Atoms: $edge(X,Y)$, $atom(f(a,b),c)$, $true$, *-true*
- Built-in Atoms: $father(aristotle) = nicomachus$, $X! = Y$, $X * 2 = Y$
- Naf-Literals: $father(aristotle) = nicomachus$, $X! = Y$, $X * 2 = Y$, $edge(X,Y)$, *-atom*$(f(a,b),c)$, *true*, *-true*, *not -true*, *not true*

In addition to the type of atoms above illustrated, *aggregate atoms* have been introduced to permit aggregation operations on multi-sets of terms by means of concise expressions.

Definition 3.1.10 (Aggregate Element)**.** An *aggregate element* is composed as: $t_1,\dots,t_m : l_1,\dots,l_n$, where $t_1,\dots,t_m$ are terms $l_1,\dots,l_n$ are naf-literals for $n \geq 0$, $m \geq 0$.

Definition 3.1.11 (Aggregate Atom)**.** An *aggregate atom* has the form:

$$af\{e_1,\dots,e_n\} \rhd t$$

where:

- $af \in \{\#count, \#sum, \#max, \#min\}$
- $e_1,\dots,e_n$ are aggregate elements for $n \geq 0$
- $\rhd \in \{<, <=, =, <>, != , >, >=\}$
- t is a term

Definition 3.1.12 (Aggregate Literal)**.** An *aggregate literal* is either a or *not* a where a is an aggregate atom.

Example 3.1.5. For instance, the following are aggregate literals: *not* $\#max\{X,Y : age(X,Y)\} < 20$, $\#sum\{X,Y : age(X,Y)\} = s(S)$, $\#count\{1 : a(1)\} > 3$. Moreover, the latter two literals are also aggregate atoms.

An atom is *ground* if it does not contain any variable. A literal is ground if its atom is ground. In Examples 3.1.4 and 3.1.5 $father(aristotle) = nicomachus$, -$atom(f(a,b),c)$, $true$, -$true$, *not* -$true$, *not* $true$, #$count\{1 : a(1)\} > 3$ are ground.

In the following we will refer to classical, built-in and aggregate atoms as *atoms*. Similarly, we will indicate naf and aggregate literals as *literals*. A literal is *negative* if the *not* symbol is present, otherwise it is *positive*.

3.1.3 Rules, Constraints, Queries and Programs

After defining the basic constructs, we now describe the main components of an ASP logic program.

Definition 3.1.13 (Rule)**.** A *rule* r has the following form:

$$a_1 \mid \ldots \mid a_n \text{ :- } b_1, \ldots, b_m.$$

where:

- $a_1, \ldots, a_n$ are classical atoms
- $b_1, \ldots, b_m$ are literals
- $n \geq 0, m \geq 0$

The *disjunction* $a_1 \mid \ldots \mid a_n$ is the *head* of r, while the *conjunction* $b_1, \ldots, b_m$ is the *body* of r. We denote by $H(r)$ the set $\{a_1, \ldots, a_n\}$ of the head atoms, and by $B(r)$ the set $\{b_1, \ldots, b_m\}$ of the body literals. $B^+(r)$ denotes the set of literals occurring positively in $B(r)$; while $B^-(r)$ is the set of negative literals in $B(r)$. A rule having precisely one head literal (i.e., $n = 1$) is said to be a *normal rule*; if $n > 1$ the rule is *disjunctive*.

Example 3.1.6. Examples of rules are:

$$hasUmbrella(X) \mid doesNotHaveUmbrella(X) \text{ :- } person(X).$$

$$isRaining \mid \text{-}isRaining \text{ :- } cloudyWeather.$$

Definition 3.1.14 (Fact)**.** A rule r is a *fact* with $B(r) = \emptyset$, $|H(r)| = 1$ and $H(r) = \{a\}$ where a is a classical ground atom.

Example 3.1.7. Examples of facts are:

$$cloudyWeather.\ \text{-}isRaining.\ person(alice).\ person(bob).$$

the :- sign is usually omitted.

In the following, as it is common, we will adopt the notation reported next to represent in a compact way a set of facts: $p(m_{1_1}..m_{1_2}, \ldots, m_{n_1}..m_{n_2})$. where p/n is a predicate of arity n, and m_{i_j} with $i \in \{1, \dots n\}$ and $j \in \{1, 2\}$ are terms. For instance, $a(1..2, f(3..4))$. defines the facts: $a(1, f(3))$. $a(2, f(3))$. $a(1, f(4))$. $a(2, f(4))$.

A predicate p/n is referred to as an *EDB* predicate if, for each rule r in which p/n appears in $H(r)$, r is a fact; all others predicates are referred to as *IDB* predicates. The set of facts in which *EDB* predicates occur, is called *Extensional Database (EDB)*, the set of all other rules is the *Intensional Database (IDB)*.

Definition 3.1.15 (Strong (or Integrity) Constraint)**.** A *strong constraint s* is a rule with $|H(s)| = \emptyset$.

Definition 3.1.16 (Weak Constraint)**.** A *weak constraint c* is a special type of rule, of the form:

$$:\sim\ b_1, \dots, b_m.\ [w@l, t_1, \dots, t_n]$$

where:

- $n \geq 0$, $m \geq 0$
- $b_1, \dots, b_m$ are literals
- $w, l, t_1, \dots, t_n$ are terms; w and l are referred to, respectively, as *weight* and *level* for c; if $l = 0$, the expression @0 can be omitted.

Basically, a weak constraint is like a strong one, where the implication symbol :- is replaced by :∼. The informal meaning of a weak constraint :∼ B. is "try to falsify B," or "B should preferably be false".

For a weak constraint c we will indicate as *weak specification*, denoted $W(c)$, the part within the square brackets.

Example 3.1.8. Examples of constraints are:

$$:-\ isRaining, not\ isWetStreet.$$
$$:\sim\ isRaining, person(X), not\ hasUmbrella(X).\ [1]$$

A rule r is *ground* if all the atoms in $H(r)$ are ground and all the literals in $B(r)$ are ground. A strong constraint s is ground if all the literals in $B(s)$ are ground. A weak constraint c is ground if all the literals in $B(c)$ are ground, and all the terms in its weak specification $W(c)$ are ground.

In Example 3.1.6 the rule *isRaining* | *-isRaining* :- *cloudyWeather*. is ground, as well as the two constraints in Example 3.1.8.

For a literal l, let $var(l)$ be the set of variables appearing in l; if l is ground $var(l) = \emptyset$. For a conjunction of literals C, $var(C)$ denotes the set of variables occurring in the literals in C; similarly, for a disjunction of atoms D, $var(D)$ denotes the set of variables in the atoms in D. Inductively, for a rule r, $var(r) = var(H(r)) \cup var(B(r))$; for a strong constraint s, $var(r) = var(B(s))$; for a weak constraint c, $var(c) = var(B(c)) \cup var(W(c))$.

Given a rule or weak constraint r, a variable X is *global* if it appears outside of an aggregate element in r; we denote as $var_g(r)$ the set of global variables in r. Given an aggregate element e in a rule or weak constraint r, $var_l(e) = var(e) \setminus var(r)$ denotes the set of *local* variables of e, i.e., the set of variables appearing only in e, while the set of *global* variables of e contains variables appearing in both r and e, i.e., $var_g(e) = var(r) \cap var(e)$. Suppose that r contains the aggregate elements $E = \{e_1, \ldots, e_n\}$, then $var(r)$ can be also defined as $var(r) = var_g(r) \cup \bigcup_{i=1}^{n} \{var_l(e_i) | e_i \in E\}$; if $n = 0$ and thus $E = \emptyset$, i.e., r does not contain any aggregate element, then $var(r) = var_g(r)$.

Example 3.1.9. As an example, given the following rule r:

$$a(X) \text{ :- } b(X),\ not\ c(X), \#sum\{Y : d(X,Y); Z : f(Z)\}.$$

we can observe that $var(r) = \{X,Y,Z\}$, $var_g(r) = \{X\}$, $var_l(Y : d(X,Y)) = \{Y\}$, $var_l(Z : f(Z)) = \{Z\}$.

In the following, we will denote rules and constraints (weak or strong) simply as *rules*.

Definition 3.1.17 (Query)**.** A *query* has the form: a? where a is a classical atom.

Example 3.1.10. Examples of queries are: *-isRaining*?, $hasUmbrella(X)$?.

A query is *ground* if its atom is ground. In Example 3.1.10 *-isRaining*? is ground.

Definition 3.1.18 (Program)**.** A *program* is a finite set of rules, possibly accompanied by a single query.

A program is *ground* if all its rule, constraints, and the possible query are ground. A program containing disjunctive rules is *disjunctive*, otherwise it is *non disjunctive*.

Example 3.1.11. The following constitutes a *disjunctive* program:

$$hasUmbrella(X) \mid doesNotHaveUmbrella(X) \text{:- } person(X).$$
$$isRaining \mid \text{-}isRaining \text{:- } cloudyWeather.$$
$$\text{:- } isRaining, not\ isWetStreet.$$
$$\text{:}\sim\ isRaining, person(X), not\ hasUmbrella(X).\ [1]$$
$$cloudyWeather.\ \text{-}isRaining.$$
$$person(alice).\ person(bob).$$
$$hasUmbrella(bob)?$$

Programs are also classified according to their structural properties, such as dependencies among predicates [CCIL08].

Definition 3.1.19. (Dependency Graph) The *Dependency Graph* of P is a directed graph $G_P = \langle N, E \rangle$, where N is the set of IDB predicates of P, and E contains an edge $(p/n, q/m)$ if there is a rule r in P such that q/m occurs in the head of r and p/n occurs in a classical atom of $B(r)$ or in a classical atom within an aggregate literal of $B(r)$.

The graph G_P induces a partition of P into subprograms (also called *modules*). For each strongly connected component (SCC)[1] C of G_P (a set of predicates), the set of rules defining the predicates in C is called *module* of C and is denoted by M_C. A rule r occurring in a module M_C (i.e., containing in its head some predicate $q/m \in C$) is said to be *recursive* if there is a predicate $p/n \in C$ in the positive body of r; otherwise, r is said to be an *exit rule*. Moreover, we say that p/n and q/m are *recursive* predicates. A program containing at least a recursive rule is said *recursive*.

Definition 3.1.20. (Component Graph) The *Component Graph* of a program P is a directed labelled graph $G^c_P = \langle N, E, lab \rangle$, where N is the set of strongly connected components of G_P, and E contains:

– an edge (B,A) with $lab((B,A)) =$"+", if there is a rule r in P such that $a \in A$ occurs in the head of r and $b \in B$ occurs in a classical atom of $B(r)$ or in a classical atom within an aggregate literal of $B(r)$;

– an edge (B,A), with $lab((B,A)) =$"-", if there is a rule r in P such that $a \in A$ occurs in the head of r and $b \in B$ occurs in a negative naf-literal of $B(r)$ or in a negative naf-literal within an aggregate literal of $B(r)$, and there is no edge e' in E, with $lab(e') =$"+".

A predicate p/n is *stratified* [ABW88] with respect to negation if it does not occur in cycles in G^c_P involving negative dependencies (i.e., edges labelled with "-"), otherwise p/n is said *unstratified*. Consequently, a program P is *stratified* with respect to negation if every predicate appearing in it is stratified, or equivalently, if no cycles in G^c_P involve negative dependencies, otherwise P is said *unstratified*. A predicate is *solved* if: (i) p/n is defined solely by non-disjunctive rules (i.e., all rules with p/n in the head are non-disjunctive), and (ii) q does not depend (even transitively) on any unstratified predicate or disjunctive predicate (i.e., a predicate defined by a disjunctive rule).

The *Component Graph* induces a partial ordering among the SCCs of the *Dependency Graph* as follows. For any pair of nodes A, B of G^c_P, *A positively precedes B* in G^c_P (denoted $A \prec_+ B$) if there is a path in G^c_P from A to B in which all arcs are labeled with "$+$"; *A negatively precedes B* (denoted $A \prec_- B$), if there is a path in G^c_P from A to B in which at least one arc is labeled with "$-$". This ordering induces admissible component sequences $C_1, \dots, C_n$ of SCCs of G_P such that for each $i < j$

– $C_j \not\prec_+ C_i$;

– if $C_j \leftarrow C_i$ then there is a cycle in G^c_P from C_i to C_j (i.e., either $C_i \prec_+ C_j$ or $C_i \leftarrow C_j$).

Several sequences exist in general.

1 We briefly recall that a strongly connected component of a directed graph is a maximal subset of the vertices, such that every vertex is reachable from every other vertex.

Example 3.1.12. As an example let us consider the following program P_1:

$$\begin{aligned}
r_1 &: a(X) \text{:-}\ b(X), not\ c(X).\\
r_2 &: b(Y) \text{:-}\ a(Y), Y = X + 1, f(X).\\
r_3 &: c(X) \text{:-}\ d(X), not\ a(X).\\
r_4 &: d(X) \text{:-}\ f(X), not\ g(X).
\end{aligned}$$

The dependency and component graphs are illustrated in Figure 3.1. In the dependency graph G_{P_1}, there are three components: (1) a first component C_1 is formed by predicate $a/1$ and $b/1$, (2) the predicate $c/1$ forms another component C_2, and (3) a third component C_3 is composed by the predicate $d/1$. Hence, $M_{C_1} = \{r_1, r_2\}$, $M_{C_2} = \{r_3\}$, $M_{C_3} = \{r_4\}$. Moreover, the rules r_1 and r_2 are recursive, thus P_1 is recursive. Finally, in the component graph $G^c_{P_1}$ there is a cycle involving components $\{a/1, b/1\}$ and $\{c/1\}$, and so P_1 is *unstratified* under negation.

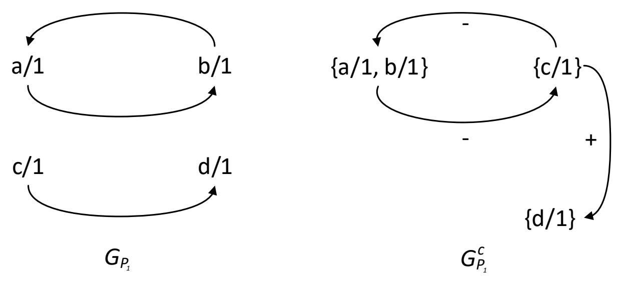


Figure 3.1: Dependency and Component Graphs.

3.2 Semantics

The semantics of an ASP program is given by the set of its *answer sets*. Each *answer set* corresponds to a solution for the encoded problem. Notably, ASP is a fully declarative paradigm: the order in which the program is composed by rules, constraints and query, as well as the order of literals and atoms in the rules bodies and heads, have no effect on the semantics.

Furthermore, answer sets are defined for ground programs only. However, for every non-ground program, a semantically equivalent ground program can be defined. The process of producing such a ground program is referred to as *instantiation* or *grounding*. Essentially, for each rule of a non–ground program its variables are considered universally quantified and ranging over the set of ground terms defined by the program Herbrand Universe. Intuitively, variables are just an abstraction to represent ground terms.

In the following, we formalize the semantics of ASP-Core-2, obtained by inheriting the semantics proposed in [GL91] as a generalization of *stable model* semantics [GL88], extended to aggregates according to [FLP04, FPL11].

3.2.1 Theoretical Instantiation

Let P be an ASP program.

Definition 3.2.1 (Herbrand Universe)**.** The *Universe of Herbrand* of P, U_P, is the set of all integers and ground terms constructible from constants and functors appearing in P. In case no constant appears in P an arbitrary constant c is added to U_P.

Definition 3.2.2 (Herbrand Base)**.** The *Base of Herbrand* of P, B_P, is the set of all ground classical atoms obtainable by combining predicate names appearing in P with terms from U_P as arguments.

Example 3.2.1. As running example, let us consider the program P_1:

$$b(1).\ b(2).\ c(1).$$
$$a(X) \text{ :- } b(X),\ not\ c(X*1).$$
$$d(Y) \text{ :- } \#count\{X:\ a(X)\}\ =\ Y.$$

then $U_{P_1} = \{1,2\}$ and $B_{P_1} = \{a(1),a(2),b(1),b(2),c(1),c(2),d(1),d(2)\}$.

Definition 3.2.3 (Substitution)**.** Given a Herbrand Universe U_P of a program P and a set of variables V, a *substitution* is total function $\sigma : V \mapsto U_P$ that maps each variable in V to an element in U_P. For some object O occurring in P (term, atom, aggregate atom, literal, rule, weak constraint, query, etc.), we denote by $O\sigma$ the object obtained by replacing each occurrence of a variable $v \in var(O)$ by $\sigma(v)$ in O. σ is well-formed if the arithmetic evaluation, performed in the standard way, of each arithmetic sub-term t in $O\sigma$ is well-defined.

In the following, we will denote a substitution σ also as the set $\{X = c \,|\, \sigma(X) = c\}$.

Definition 3.2.4 (Global and Local Substitutions)**.** Given a rule or weak constraint r in P a substitution is *global* if it involves variables in $var_g(r)$; for an aggregate element e in r, a substitution is *local* if it involves variables in $var_l(e)$.

We remark that for terms, classical atoms, naf-literals and queries a substitution is implicitly global, due to the absence of aggregate elements. In the following for the above mentioned constructs we will indicate substitutions for them as *global* substitutions.

Example 3.2.2. Consider the rule r_1 from P_1 and the (global) substitution $\sigma_1 = \{X = 1\}$, then $r_1\sigma_1 = a(1) \text{ :- } b(1),\ not\ c(1*1)$. Note that σ_1 is well-formed, while for instance, supposing that U_P contained also the symbolic constant abc, then a substitution $\sigma_2 = \{X = abc\}$ would not be well-formed.

Now, consider the rule r_2 from P_1 and the global substitution $\sigma_3 = \{Y = 1\}$, then $r_2\sigma_3 = d(1)$:- $\#count\{X :\ a(X)\} \ = \ 1$. If instead, we consider the local substitution $\sigma_4 = \{X = 1\}$, then $r_2\sigma_4 = d(Y)$:- $\#count\{1 :\ a(1)\} \ = \ Y$.

The instantiation of an aggregate element e is obtained by considering well-formed local substitutions for e; formally, the instantiation of e consists of the following set of ground aggregate elements:

$$inst(e) = \{e\sigma | \sigma \text{ is a well-formed local substitution for } e\}$$

Inductively, the instantiation of a series of aggregate elements $\{e_1, \ldots, e_n\}$ is provided by the set of aggregate elements reported below:

$$inst(\{e_1, \ldots, e_n\}) = \bigcup_{i=1}^{n} \{e_i\sigma | \sigma \text{ is a well-formed local substitution for } e_i\}$$

A *ground instance* of a term, classical atom, naf-literal, a rule, weak constraint, or query o is obtained in two steps: (i), a well-formed global substitution σ for o is applied to o; (ii), for every aggregate atom $af\{e_1, \ldots, e_n\} \rhd t$ in $r\sigma$ its aggregate elements $\{e_1, \ldots, e_n\}$ are replaced by $inst(\{e_1, \ldots, e_n\})$.

Example 3.2.3. Consider the aggregate element $e = \{X :\ a(X)\}$ of rule r_2 from P_1, then the instantiation $inst(e)$ of e consists of $inst(e) = \{1 : a(1); 2 : a(2)\}$.

At this point, a ground instance of r_2 is obtained by applying the substitution $\sigma_3 = \{Y = 1\}$, and replacing e with $inst(e)$: $d(1)$:- $\#count\{1 : a(1); 2 : a(2)\} \ = \ 1$.

The arithmetic evaluation of a ground instance g of some term, classical atom, naf-literal, rule, weak constraint or query is obtained by replacing any maximal arithmetic subterm appearing in g by its integer value, which is calculated in the standard way.

The ground instantiation of a program P, denoted by $grnd(P)$, is the set of arithmetically evaluated ground instances of rules, strong and weak constraints in P.

Example 3.2.4. Eventually, let us consider P_1, $grnd(P_1)$ consists of:

$b(1).\ b(2).\ c(1).$
$a(1)$:- $b(1),\ not\ c(1).$
$a(2)$:- $b(2),\ not\ c(2).$
$d(1)$:- $\#count\{1 : a(1); 2 : a(2)\} \ = \ 1.$
$d(2)$:- $\#count\{1 : a(1); 2 : a(2)\} \ = \ 2.$

Note that the substitution $\{X = 1, X = 2\}$ has been applied to r_1, and the arithmetic terms $(1 * 1)$ and $(2 * 1)$ have been evaluated respectively to 1 and 2.

Remark 3.2.1. The instantiation of a program is idempotent: for each program P, $ground(P) = ground(ground(P))$.

3.2.2 Interpretations

Once that a ground program is obtained, the truth values of atoms, literals, rules, constraints etc. is properly defined according to interpretations.

Definition 3.2.5 (Herbrand Interpretation)**.** A *(Herbrand) interpretation* I for P is a *consistent* subset of B_P; to this end, for each predicate atom $a \in B_P$, $\{a, -a\} \nsubseteq I$ must hold.

Literals can be either true or false w.r.t. an interpretation. To illustrate how their truth values are determined, as a preliminary step, we need to define a proper total order $\preceq$ on terms in U_P. Several orderings may be defined, in ASP-Core-2 the one reported next has been adopted.

Let t and u be two arithmetically evaluated ground terms, then:

- $t \preceq u$ for integers t and u if $t \leq u$,
- $t \preceq u$ if t is an integer and u is a symbolic constant,
- $t \preceq u$ for symbolic constants t and u with t lexicographically smaller or equal to u,
- $t \preceq u$ if t is a symbolic constant and u is a string constant,
- $t \preceq u$ for string constants t and u with t lexicographically smaller or equal to u,
- $t \preceq u$ if t is a string constant and u is a functional term,
- $t \preceq u$ for functional terms $t = f(t_1, \ldots, t_n)$ and $u = g(u_1, \ldots, u_n)$ if either:
  - $m < n$ or,
  - $m = n$ and $g \npreceq f$ (f is lexicographically smaller than g) or,
  - $m = n$, $f \preceq g$ and, for any $1 \leq j \leq m$ such that $t_j \npreceq u_j$, there is some $1 \leq i < j$ such that $t_i \npreceq u_i$ (i.e., the tuple of terms of t is smaller than or equal to the arguments of u).

At this point, we are ready to properly define literals satisfaction. Let $I \subseteq B_P$ be a consistent interpretation for P.

The satisfaction of built-in atoms can be easily defined according to the total order $\preceq$, in the intuitive way, as they represent comparisons among terms. A ground classical atom $a \in B_P$ is *true* w.r.t. I if $a \in I$. A positive ground naf-literal a is *true* w.r.t. I if a is a classical or built-in atom that is true w.r.t. I; otherwise, a is false w.r.t. I. A negative ground naf-literal *not* a is true (or false) w.r.t. I if a is false (or true) w.r.t. I.

Given a ground aggregate atom $af\{e_1, \ldots, e_n\} \rhd t$, in order to correctly evaluate its semantics according to its aggregate function, the expression $af\{e_1, \ldots, e_n\}$ has to be mapped to a term, say u. Indeed, aggregate functions can be seen as mappings from set of terms to a term. Let T be the set of terms in $\{e_1, \ldots, e_n\}$, then:

– if $ag = \#count$, then $u = |T|$;

– if $ag = \#sum$, then $u = \sum_{t_i \in T} t_i$ is an integer;

– if $ag = \#max$, then $u = max\{t_i | t_i \in T\}$

– if $ag = \#min$, then $u = min\{t_i | t_i \in T\}$

Essentially, *#count* depends on the cardinality of the set of terms T, *#sum* is evaluated as the sum of the integers in T, while *#max* and *#min* functions strictly rely on the total order $\preceq$ on terms in U_P. In case $T = \emptyset$, the following convention is adopted: $\#max\{\emptyset\} \preceq u$ and $u \preceq \#min\{\emptyset\}$ for every term $u \in U_P$.

Fixed an interpretation some aggregate elements may not contribute to the semantics of an aggregate atom. Intuitively, an interpretation can filter out some aggregate elements according to their truth values w.r.t. the interpretation itself. More formally, the interpretation I maps a collection E of aggregate elements to the following set of tuples of ground terms:

$$\begin{aligned} eval(E,I) = \{ & (t_1,\dots,t_n) | \{t_1,\dots,t_n : l_1,\dots,l_m\} \text{ occurs in } E \text{ and} \\ & \{l_1,\dots,l_m\} \text{ are true w.r.t. } I\} \end{aligned}$$

Let $a = af\{e_1,\dots,e_n\} \rhd t$ be an aggregate atom, a is true (or false) w.r.t. I if $af\{eval(e_1,\dots,e_n,I)\} \rhd t$. A positive aggregate literal a is true (or false) w.r.t. I if a is an aggregate atom that is true (or false) w.r.t. I. A negative aggregate literal *not* a is true (or false) w.r.t. I if a is false (or true) w.r.t. I.

Let r be a ground rule in $grnd(P)$. The head of r is *true* w.r.t. I if $H(r) \cap I \neq \emptyset$. The body of r is *true* w.r.t. I if all body literals of r are true w.r.t. I (i.e., $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$) and is *false* w.r.t. I otherwise. The rule r is *satisfied* (or *true*) w.r.t. I if its head is true w.r.t. I or its body is false w.r.t. I.

Example 3.2.5. Let

$$I_1 = \{b(1),b(2),c(1),a(1),d(1),d(2)\}$$

be an interpretation for $grnd(P_1)$, then:

3.2.3 Answer Sets

Definition 3.2.6 (Model)**.** A *model* for P is an interpretation M for P such that every rule $r \in grnd(P)$ is true w.r.t. M.

Definition 3.2.7 (Minimal Model)**.** A model M for P is *minimal* if no model N for P exists such that N is a proper subset of M. The set of all minimal models for P is denoted by $\text{MM}(P)$.

$b(1).\ b(2).\ c(1).$	are satisfied w.r.t. I_1.
$a(1)$:- $b(1)$, *not* $c(1)$.	is not satisfied because *not* $c(1)$ is false w.r.t. I_1.
$a(2)$:- $b(2)$, *not* $c(2)$.	is not satisfied because $a(2)$ is false w.r.t. I_1.
$d(1)$:- $\#count\{1:a(1);2:a(2)\} = 1.$	is satisfied w.r.t. I_1 since $eval(\{1 : a(1);2 : a(2)\},I_1) = \{1 : a(1)\}$, and $\#count\{1:a(1)\} = 1$.
$d(2)$:- $\#count\{1:a(1);2:a(2)\} = 2$	is not satisfied w.r.t. I_1 because of the evaluation reported above.

$b(1).\ b(2).\ c(1).$	are satisfied w.r.t. I_2.
$a(1)$:- $b(1)$, *not* $c(1)$.	is satisfied w.r.t. I_2 because both the body and the head are false.
$a(2)$:- $b(2)$, *not* $c(2)$.	is satisfied w.r.t. I_2.
$d(1)$:- $\#count\{1:a(1);2:a(2)\} = 1.$	is satisfied w.r.t. I_2 since $eval(\{1 : a(1);2 : a(2)\},I_2) = \{2 : a(2)\}$, and $\#count\{2:a(2)\} = 1$.
$d(2)$:- $\#count\{1:a(1);2:a(2)\} = 2$	is satisfied w.r.t. I_2 because of the evaluation reported above, thus both the body and the head are false.

Example 3.2.6. The interpretation I_1 is not a model for P_1, while the interpretation

$$I_2 = \{b(1),b(2),c(1),a(2),d(1)\}$$

is a model for P_1:

Moreover, I_2 is also a minimal model for P_1.

Definition 3.2.8 (Reduct)**.** Given a ground program P and an interpretation I, the *reduct* of P w.r.t. I is the subset P^I of P, which is obtained from P by deleting rules in which a body literal is false w.r.t. I.

It is worthwhile noting that the above definition of reduct, proposed in [FLP04], simplifies the original definition of Gelfond-Lifschitz (GL) transform [GL91], but is fully equivalent to the GL transform for the definition of answer sets [FLP04].

Definition 3.2.9 (Answer Set)**.** [Prz91, GL91] Let I be an interpretation for a program P. I is an *answer set* for P if $I \in MM(P^I)$ (i.e., I is a minimal model for the program P^I). The set of all answer sets for P is denoted by $AS(P)$.

Example 3.2.7. Let us consider $grnd(P_1)$ and I_2, then the *reduct* $grnd(P)^{I_2}$ is:

$$\begin{aligned}
&b(1).\, b(2).\, c(1).\\
&a(2) \text{ :- } b(2),\ not\ c(2).\\
&d(1) \text{ :- } \#count\{2 : a(2)\} \ = \ 1.
\end{aligned}$$

I_2 is a minimal model for $grnd(P_1)^{I_2}$ since no proper subset of I_2 exists such that is a model for it, and thus I_2 an *answer set* for P_1.

In particular, since P_1 is a *not disjunctive* and *stratified* program, I_2 is the unique answer set for P_1, i.e., $AS(P_1) = \{I_2\}$. This type of program admits a unique answer set, which corresponds to its *perfect model* [EIK09b].

Furthermore, we distinguish *coherent* and *incoherent* programs: *coherent* programs admit at least one answer set, while *incoherent* programs have no answer sets.

Equivalence of logic programs is a fundamental property also from a practical perspective. For instance, it is common that one looks for an equivalent version Π' of a logic program Π which can be possibly evaluated more efficiently.

Definition 3.2.10 (Equivalent Logic Programs)**.** [GL88] A logic program Π_1 is said to be *equivalent* to a logic program Π_2 in the sense of the answer set semantics if Π_1 and Π_2 have the same answer sets.

Definition 3.2.11 (Strongly Equivalent Logic Programs)**.** [LPV01] A program Π_1 is said to be *strongly equivalent* to a program Π_2 if for every logic program Π, $\Pi_1 \cup \Pi$ has the same answer sets as $\Pi_2 \cup \Pi$.

In case of weak constraints, answer sets need to be further examined, and classified as *optimal* or not. Intuitively, strong constraints represent conditions that *must* be satisfied, while *weak constraints*, introduced originally in [LPF+06b, BLR00], indicate conditions that *should* be satisfied; their semantics involves minimizing the number of violations, thus allowing to easily encode optimization problems.

Optimal answer sets of P are selected among $AS(P)$, according to the following schema. Let I be an interpretation, then:

$$\begin{aligned}
weak(P,I) = \{ & (w@l, t_1, \dots, t_m)\\
& :\sim b_1, \dots, b_n [w@l, t_1, \dots, t_m] \text{ occurs in } grnd(P)\\
& \text{and } b_1, \dots, b_n \text{ are true w.r.t. } I\}
\end{aligned}$$

For any integer l, let

$$P_l^I = \sum_{(w@l,t_1,\dots,t_m)\in weak(P,I), w \text{ is an integer}} w$$

denotes the sum of integers w over tuples with $w@l$ in $weak(P,I)$.

In other words, for each weak constraints satisfied by I in $grnd(P)$ we sum the weights per level: these numbers represent a sort of penalty paid by I: the lower they are, the higher is the possibility for I, if it represents an answer set, to be optimal.

More formally, we define the notion of *domination* among answer sets as follows. Given an answer set $A \in AS(P)$, it is said *dominated* by another answer set A' if there is some integer l such that $P_l^{A'} < P_l^A$ and $P_{l'}^{A'} < P_{l'}^A$ for all integers $l' > l$. An answer set $A \in AS(P)$ is optimal if there is no $A' \in AS(P)$ such that A is dominated by A'. In general, a coherent program may have one or more optimal answer sets.

Example 3.2.8. Let us consider the following ground program P_2:

$c(1).\ c(2).$
$a(1) \mid b(1) \text{:-}\ c(1).$
$a(2) \mid b(2) \text{:-}\ c(2).$
$:\sim a(1).\ [1@1]$
$:\sim b(1).\ [1@2]$
$:\sim a(2).\ [2@1]$
$:\sim b(2).\ [2@2]$

The set $AS(P_2)$ consists of:

$as_1:\ c(1).\ c(2).\ a(1).\ b(2)$
$as_2:\ c(1).\ c(2).\ a(1).\ a(2)$
$as_3:\ c(1).\ c(2).\ b(1).\ b(2)$
$as_4:\ c(1).\ c(2).\ b(1).\ a(2)$

For an answer set a, we will represent as $< \{w_1, l_1\}, \ldots, \{w_n, l_n\} >$ the sum of weights $w_1, \ldots, w_n$ for $l_1, \ldots, l_n$, $n \geq 0$. Now, for as_1 we obtain $< \{1,1\}, \{2,2\} >$, for as_2 $< \{3,1\}, \{0,2\} >$, for as_3 $< \{0,1\}, \{3,2\} >$ and finally for as_4 $< \{2,1\}, \{1,2\} >$. Hence, P_2 admit a unique optimal answer set, namely as_2, since it is not dominated by any other answer sets.

3.3 THE GUESS-CHECK-OPTIMIZE TECHNIQUE

The *Guess/Check/Optimize* (GCO) paradigm is a declarative programming methodology which allows to encode complex queries and, more generally, search problems in a simple and highly declarative fashion; even some optimization problems of rather high computational complexity can be declaratively encoded by using this methodology.

One of the most important element that distinguishes ASP from other *Logic Programming* languages is the implementation of the so called "*Guess & Check*" paradigm which is, sometimes, also called "*Generate & Test*" methodology [Lif02, EIK09c]. The idea behind the *Guess & Check* paradigm is to proceed as follows:

– the *guess* part uses nondeterminism that comes with unstratified negation, or equally well with disjunction in rule heads, to create candidate solutions to a problem (program part $G \subseteq P$), whereas

- the *check* part, with further rules and/or constraints, filters the solution candidate in such a way that they are admissible for the current program instance (program part $C \subseteq P$). This part may also involve auxiliary predicates, if needed.

More in details, the part G defines the search space, and the part C prunes illegal branches.

$$\left.inS(X) \mid outS(X) \text{ :- } node(X).\right\} Guess$$

$$\left.\text{:- } edge(X,Y),\ not\ inS(X),\ not\ inS(Y).\right\} Check$$

An extension and refinement of the *Guess & Check* paradigm is the *Guess/Check/Optimize* (GCO) methodology, which adds one more step to the traditional ones (the optimization part) which can be described as follows:

- the *optimization* part $O \subseteq P$ of the program allows to express a quantitative cost evaluation of solutions by using weak constraints.

It implicitly defines an objective function $f : AS(G \cup C \cup F_I) \rightarrow N$ mapping the answer sets of $G \cup C \cup F_I$ to natural numbers, where F_I represents a set of facts that specify an instance I of some problem $\mathscr{P}$. The semantics of $G \cup C \cup F_I \cup O$ optimizes f by filtering those answer sets having the minimum value; this way, the optimal (least cost) solutions are computed [LPF+06a].

$$\left.\text{:}\sim\ outS(X).\ [1@1,X]\right\} Optimize$$

Sometimes, an additional *define* part is used to model auxiliary predicates and their relationships with other parts of the logic programs. This part usually consists of rules [Lif08].

For further and deeper details about GCO methodology, the reader can refer to [Lif02, LPF+06a, EIK09c]

3.4 Applications and Examples

3.4.1 Three-colorability

As first example, consider the well-known NP-complete problem *Three-colorability*:

> Given an undirected graph $G = (V, E)$, assign to each vertex one of three colors such that adjacent vertices always have distinct colors.

Firstly, we can start defining a problem instance via facts. In this case, we need to model an undirected graph $G = (V, E)$:

– vertices can be encoded as facts of the form $vertex(x)$;

– for edges, facts of type $edge(x,y)$ can be used to encode that is there an edge between the vertices x and y.

The next step consists of encoding the actual problem into an ASP *GCO* program P_{3col}. In the guessing part we define the search space, thus we assign to each vertex exactly one color among the three available, say red, green and blue. In the checking part we ensure that no two connected vertices are associated with the same color. For this problem, we do not need an optimize part, indeed, there are not preferences among solutions to be expressed.

```
% Guessing Part (1 rule)
color(X,red) | color(X,green) | color(X,blue) :- vertex(X).

% Checking Part (1 rule)
:- edge(X,Y), color(X,C), color(Y,C).
```

Notably, thanks to the declarative capability of ASP, when designing the encoding the focus is on how to *model* the problem at hand, rather than on how to actually *solve* it.

By coupling P_{3col} with a set of facts F for *vertex* and *edge*, if the program $P_{3_{col}} \cup F$ is coherent, than each answer set represents an admissible solution. For instance, suppose that

$$F = \{vertex(1), vertex(2), vertex(3), edge(1,2), edge(1,3), edge(2,3)\}$$

then the input graph is complete (i.e. every pair of distinct vertices is connected by an edge) and the answer set of $P_{3_{col}} \cup F$ are:

$$
\begin{array}{llll}
as_1: & color(1,blue). & color(2,red). & color(3,green).\\
as_2: & color(1,blue). & color(2,green). & color(3,red).\\
as_3: & color(1,red). & color(2,blue). & color(3,green).\\
as_4: & color(1,red). & color(2,green). & color(3,blue).\\
as_5: & color(1,green). & color(2,red). & color(3,blue).\\
as_6: & color(1,green). & color(2,blue). & color(3,red).
\end{array}
$$

3.4.2 Hamiltonian Path

As next example, let us consider a classical NP-complete problem in graph theory, namely *Hamiltonian Path*:

> Given a directed graph $G = (V,E)$ and a vertex $v \in V$ of this graph, does there exist a path in G starting from v and passing through each vertex in V exactly once?

Similarly to Example 3.4.1 the graph $G = (V,E)$ can be specified by means of facts over predicates $vertex/1$ and $edge/2$. Moreover, we need to define the starting vertex v: it can be specified by the predicate $start/1$. Consequently, the following program P_{hp} encodes a solution to the problem:

```
% Guessing Part (3 rules)
{inPath(X,Y)}:- start(X),edge(X,Y).
{inPath(X,Y)}:- reached(X),edge(X,Y).
reached(X) :- inPath(Y,X).

% Checking Part (3 rules)
:- inPath(X,Y),inPath(X,Y1),Y <> Y1.
:- inPath(X,Y),inPath(X1,Y),X <> X1.
:- vertex(X),not reached(X),not start(X).
```

In the guessing part, the two choice rules guess a subset S of the edges to be in the path, while the rest of the program checks whether S constitutes a Hamiltonian Path. Here, an auxiliary predicate reached is used, which is associated with the guessed predicate *inPath* using the third rule. Note that reached is completely determined by the guess for *inPath*, and no further guessing is needed. In turn, through the second rule, the predicate *reached* influences the guess of *inPath*, which is made somehow inductively: initially, a guess on an edge leaving the starting vertex is made by the first rule, followed by repeated guesses of edges leaving from reached vertices by the second rule, until all reached vertices have been handled.

In the checking part, the first two constraints ensure that the set of edges S selected by *inPath* meets the following requirements, which every Hamiltonian Path must satisfy: (i) there must not be two edges starting at the same vertex, and (ii) there must not be two edges ending in the same vertex. The third constraint enforces that all vertices in the graph are reached from the starting vertex in the subgraph induced by S.

It is easy to see that any set of edges S which satisfies all three constraints must contain the edges of a path $v_0, v_1, \ldots, v_k$ in G that starts at vertex $v_0 = a$, and passes through distinct vertices until no further vertex is left, or it arrives at the starting vertex a again. In the latter case, this means that the path is in fact a Hamiltonian Cycle (from which a Hamiltonian path can be immediately computed, by dropping the last edge).

Thus, given a set of facts F for $vertex$, $edge$, and $start$, the program $P_{hp} \cup F$ has an answer set if and only if the corresponding graph has a Hamiltonian Path.

3.4.3 k-Clique

In this example we focus on the relation between *stratification* negation and *disjunction*. To this end, let us consider the following NP –complete problem in graph theory, referred to as *k–Clique*:

> Given an undirected graph $G = (V, E)$ and an integer k, a $k - clique$ of G is a complete subgraph of G with k vertices.

As in previous examples, a graph can be encoded by means of facts representing vertices and edges and a fact of form $k(n)$ can be used to represent the integer k, while the problem can be encoded in the following $P_{k-clique}$ program:

% Guessing Part (2 rules)

$clique(X)$:- $vertex(X), not\ nonClique(X)$.

$nonClique(X)$:- $vertex(X), not\ clique(X)$.

% Checking Part (2 rules)

:- $\#count\{X : clique(X)\}! = N, k(N)$.

:- $clique(X), clique(Y), X! = Y, not\ edge(X,Y), not\ edge(Y,X)$.

The guessing part generates potential cliques. Notably, in these two guessing rules there is a negative dependency between the predicates $clique/1$ and $nonClique/1$, thus $P_{k-clique}$ is unstratified under negation. Interestingly, we are using unstratified negation to simulate disjunction: by replacing the two rules with the rule

$$clique(X) \mid nonClique(X) \text{:- } vertex(X).$$

we obtain a semantically equivalent program. Indeed, even if only in some cases, unstratified negation can, somehow, model disjunction. More in detail, unless the polynomial hierarchy collapses, this correspondence holds for NP problems, as in general disjunction allows to express problems up to $\Sigma_2^P = \mathrm{NP}^{\mathrm{NP}}$. In the checking part, the first constraint verifies that only k-cliques are considered and the second one ensures that vertices belonging to a clique are connected. Given an instance I fixing k to k_1, if $P_{k-clique} \cup I$ has no answer set, then no k_1-clique exists, otherwise each returned answer set will represent a clique of the desired size k_1.

As final remark, for the guessing part we could instead use a *choice* rule as in the following $Q_{k-clique}$ program:

% Guessing Part (1 rule)

$N = \{clique(X) : vertex(X)\}$:- $k(N)$.

% Checking Part (1 rule)

:- $clique(X), clique(Y), X! = Y, not\ edge(X,Y), not\ edge(Y,X)$.

By doing so, the first constraint can be removed, since the requirement on the clique size can be directly specified in the choice rule.

3.4.4 Maximal Clique

So far, we considered problems without an optimizing part. As example of complete *GCO* program let us consider a NP-hard problem, namely *Maximal Clique*. It can be considered as a variant of the *k-clique* problem:

> Given an undirected graph $G = (V, E)$ and an integer k, determine a clique C of maximal size in G, i.e. for each other clique C' in G, the number of vertices in C must be larger than or equal to the number of nodes in C'.

Readopting the same instance representation of Example 3.4.3, with few modifications to the program $P_{k-clique}$ we obtain the following encoding $P_{max-clique}$:

```
% Guessing Part (2 rules)
clique(X) :- vertex(X), not nonClique(X).
nonClique(X) :- vertex(X), not clique(X).

% Checking Part (1 rule)
:- clique(X), clique(Y), X != Y, not edge(X,Y), not edge(Y,X).

% Optimizing Part (1 weak constraint)
:~ nonClique(X). [1,X]
```

The guessing part remains unchanged, while in the checking part we removed the constraint on the cliques size. Indeed, in this case we want to find cliques with the maximal size, which is unknown, thus a weak constraint allows us to express this requirement. Each answer set will pay a penalty at level 0 according to how many vertices are not included in the clique it represents, hence the optimal answer set(s) will be the ones with the cliques of the biggest possible size.

Part II

THE HEAD-ASP SYSTEM

4 KNOWLEDGE BASE OF THE SYSTEM

During the development of the system, one of the main technical challenges we faced up, was the designing of a knowledge representation model being able to accommodate domain medical knowledge, in order to provide a natural and formal encoding of the ICHD diagnoses and criteria.

4.1 Relational Schema and Assertions

The model is based on a core relational schema consisting of 18 predicates of three different types, both intensional and extensional. Three of these predicates are intensional (type 3) and are used to derive diagnoses, criteria and subcriteria of a specific patient; the remaining ones are extensional. Among the latter, six of them (type 1) are used to represent key notions of ICHD-3, such as all possible diagnoses, symptoms and their attributes; the remaining ones (type 2) are used to encode patient history, such as specific symptoms and the number of their attacks. Hence, all instances of predicates of type 1 and type 3, as well as all rules, are always present in $\mathscr{P}$. Conversely, instances of predicates of type 2 vary with patients.

Type 1. As said, this group of predicates models key notions of ICHD-3. We first represent all possible diagnoses, symptoms and attributes via the following binary predicates: *ichdDiagnosis(Id, Name)*, *ichdSymptom(Id, Name)* and *ichdAttribute(Id, Name)*, where, in all the cases, the first term is an identifier and the second one its name. For example, *ichdDiagnosis(d.1.1, "migraine without aura")* represents the diagnosis "1.1. Migraine without aura", *ichdSymptom(s4, "headache")* represents the symptom "headache" identified by code *s4*, and *ichdAttribute(loc2, "bilateral location")* represents the attribute "bilateral location" with its identification code *loc2*. Moreover, we also make explicit some properties of attributes that are only implicit in ICHD-3. First, there are cases in which the presence of an attribute associated to a specific symptom excludes the possibility to have some other attribute for the same symptom. We use the predicate *mutuallyExclusive(Id_attr_1, Id_attr_2)* to model such kind of information. For example, if a headache is characterized by a unilateral location, then it cannot be characterized, at the same time, by a bilateral location. Hence, we may have *mutuallyExclusive(loc1, loc2)* to express the mutual exclusion between the attribute "unilateral location" identified by code *loc1* and the attribute "bilateral location" identified by code *loc2*. Clearly, these two identifiers occur in two facts of the form: *ichdAttribute(loc1, "unilateral location")* and *ichdAttribute(loc2, "bilateral location")*. Similarly, we also have *sameAs(Id_attr_1, Id_attr_2)* to specify that, if an attribute is associated with a specific symptom, then, at the same time, also the attribute semantically equivalent to the first one should be associated with the same symptom. This is very important because ICHD-3 often uses different synonymous terms in different diagnostic criteria; without modeling such similarities our encoding would not completely reflect the intended ICHD-3 meaning. For example, we include in $\mathscr{P}$ *sameAs(int2, int10)*, *ichdAttribute(int2, "strong intensity")*, and *ichdAttribute(int10, "acute intensity")*. Finally, we represent the dependence between diagnoses, symptoms and attributes via the predicate *isA(Id_-1,Id_2)*. An example concerning the symptoms is the atom *isA(s18, s54)*, where *s18* and *s54* are provided by *ichdSymptom(s18, "diplopia")* and *ichdSymptom(s54, "visual symptom")*.

TYPE 2. In this group we have predicates used to encode patient history. We start with *symptom(Id_sym)* and *symAttribute(Id_sym, Id_attr)*, modelling the fact that a patient has a specific symptom and the fact that his/her symptoms have some peculiarity modeled via what we called attributes (e.g., symptom location, pain type); for example, *symptom(s4)* and *symAttribute(s4, loc2)* represent that the patient reported the symptom "headache" with the attribute "bilateral location"; indeed, *ichdSymptom(s4, "headache")* and *ichdAttribute(loc2, "bilateral location")* hold. We also use the predicates *minAttacks(Id_sym, Value)* and *maxAttacks(Id_sym, Value)* to specify that the actual number of attacks of a patient's symptom falls in a certain range; for example *minAttacks(s4, 5)* (resp. *maxAttacks(s4, 10)*) indicates that the patient reported at least "5" (resp. at most "10") attacks associated with the symptom identified by code *s4*. Similarly, we use the predicates *minDuration(Id_sym, Value)* and *maxDuration(Id_sym, Value)* to specify the duration of the pain caused by a certain symptom. As an example, *minDuration(s4, 240)* (resp. *maxDuration(s4, 4320)*) specify that the patient reported that the symptom *s4* lasts at least *240* (resp. at most *4320*) minutes. In the same way, frequency of pain attacks that are associated to a certain symptom is modeled by means of the predicates *minDaysPerMonth(Id_sym, Value)* and *maxDaysPerMonth(Id_sym, Value)*. It is important to underline that the identifiers of type 2 predicates are those collected in type 1 predicates, therefore, the values of type 2 depend on what has been specified in type 1. Finally, we use the predicate *reportedCriterion(Description)* to model some portions of text for which it was not possible and convenient to identify a decomposed modeling. Indeed, they express peculiar assertions only of certain diagnoses, occurring infrequently within the domain or, when reported, do not present syntactic or semantic variations. The only term of this predicate is the textual description of the statement. For example, *reportedCriterion ("At least one first or second degree family member has had attacks that meet the criteria of hemiplegic migraine")* expresses the presence of the specified statement.

TYPE 3. Predicates in this group have the following signature: *diagnosis(Id)*, *criterion(Id_diag, Letter)*, *subCriterion(Id_diag, Letter_crit, Number)*. They model the identifiers of diagnoses, criteria and subcriteria that can be derived for a specific patient. As an example, if the atom *diagnosis(d.1.1)* can be derived from $\mathscr{P}$, this means that the ICHD-3 diagnosis “1.1. Migraine without aura” is compatible with patient history. Likewise, the derivation of the atom *criterion(d.1.1,“A”)* means that criterion “A” of “1.1. Migraine without aura” is compatible with patient history, namely, the patient reported “at least five attacks fulfilling criteria B-D”. Finally, the derivation of the atom *subCriterion(d.1.1,“C”,1)* means that subcriterion “1” of criterion “C” of “1.1. Migraine without aura” is compatible with patient history, namely, the patient reported that headache has the characteristic “unilateral location”.

4.2 Methodology for Diagnoses Encoding

In what follows, we encode ICHD-3 diagnoses and criteria via an ASP program $\mathscr{P}$ simply using (stratified) negation, aggregates and strong negation. As shown in the previous subsection, the diagnoses are organized in a hierarchical structure which expresses the specialization-generalization relations existing between them (Figure 4.1). Intuitively, if a more specific diagnosis is compatible, then it is possible to infer a more generic diagnosis (the former being a sub-type of the latter); conversely, if a higher level diagnosis is not compatible, all its related specializations are invalidated. The following rules, among the others, express the implications inferable from the relations between diagnoses.

$$r_1 : diagnosis(Id\_sup) \leftarrow diagnosis(Id),\ isA(Id, Id\_sup).$$
$$r_2 : -diagnosis(Id) \leftarrow -diagnosis(Id\_sup), isA(Id, Id\_sup).$$

The two rules shown below define the conditions that must be met so that the diagnosis “1.1. Migraine without aura” can be compatible or not compatible (Figure 1.1). Intuitively, rule r_3 derives the diagnosis as true (compatible) if all its criteria “(A-D)” are true. Similarly, rule r_4 expresses that the diagnosis is certainly false (not compatible) if at least one of its criteria is certainly false. Note that to use the strong negation is quite convenient to model such scenarios in which we need to distinguish criteria and diagnoses that are definitely confirmed from those definitely disconfirmed and from those still undefined, without any need in adding new predicates and further constants. This is also reflected to the predicates encoding patient’s symptomatology.

$$\begin{aligned} r_3 : diagnosis(Id) \leftarrow\ & ichdDiagnosis(Id, \text{“}migraine\ without\ aura\text{”}),\\ & criterion(Id,\ \text{“}A\text{”}),\ criterion(Id,\ \text{“}B\text{”}),\ criterion(Id,\ \text{“}C\text{”}),\ criterion(Id,\ \text{“}D\text{”}).\\ r_4 : -diagnosis(Id) \leftarrow\ & ichdDiagnosis(Id, \text{“}migraine\ without\ aura\text{”}), -criterion(Id, \_).\end{aligned}$$

Each criterion (resp. subcriterion) of ICHD-3 is encoded through rules whose head is constituted by the predicate *criterion* (resp. *subcriterion*), while the body models effectively the semantics of the statement of the criterion (resp. sub-criterion), on the basis of its informal description in the international classification. As an example, consider criterion “B" of diagnosis “1.1”: “Headache attacks lasting 4-72 hr”. The following rules model such monothetic criterion. Declaratively, rule r_5 states that if the symptom “headache” is true and its duration respects the specified time interval, then *criterion(d.1.1,"B")* will be derived as true. Rules r_6, r_7, r_8 express the conditions according to which the criterion is certainly false.

r_5 : *criterion*(*Id*, *"B"*) ← *ichdDiagnosis*(*Id*, *"migraine without aura"*),
ichdSymptom(*Id_sym*, *"headache"*), *symptom*(*Id_sym*),
minDuration(*Id_sym*,240), *maxDuration*(*Id_sym*,4320).
r_6 : −*criterion*(*Id*, *"B"*) ← *ichdDiagnosis*(*Id*, *"migraine without aura"*),
ichdSymptom(*Id_sym*, *"headache"*), *symptom*(*Id_sym*),
−*minDuration*(*Id_sym*,240).
r_7 : −*criterion*(*Id*, *"B"*) ← *ichdDiagnosis*(*Id*, *"migraine without aura"*),
ichdSymptom(*Id_sym*, *"headache"*), *symptom*(*Id_sym*),
−*maxDuration*(*Id_sym*,4320).
r_8 : −*criterion*(*Id*, *"B"*) ← *ichdDiagnosis*(*Id*, *"migraine without aura"*),
ichdSymptom(*Id_sym*, *"headache"*), −*symptom*(*Id_sym*).

Consider criterion "C" of diagnosis "1.1": "Headache has at least two of the following four characteristics: (*i*) unilateral location; (*ii*) pulsating quality; (*iii*) moderate or severe pain intensity; (*iv*) aggravation by or causing avoidance of routine physical activity (eg, walking or climbing stairs)". The following portion of the program corresponds to the encoding of such polythetic criterion.

r_9 : *criterion*(*Id*, *"C"*) ← *ichdDiagnosis*(*Id*, *"migraine without aura"*),
#count{*X* : *subCriterion*(*Id*, *"C"*, *X*)} >= 2.
r_{10} : −*criterion*(*Id*, *"C"*) ← *ichdDiagnosis*(*Id*, *"migraine without aura"*),
#count{*X* : −*subCriterion*(*Id*, *"C"*, *X*)} >= 3.
r_{11} : *subCriterion*(*Id*, *"C"*, 1) ← *ichdDiagnosis*(*Id*, *"migraine without aura"*),
ichdSymptom(*Id_sym*, *"headache"*), *symptom*(*Id_sym*),
symAttribute(*Id_sym*,*Id_attr*), *ichdAttribute*(*Id_attr*, *"unilateral location"*).
r_{12} : −*subCriterion*(*Id*, *"C"*, 1) ← *ichdDiagnosis*(*Id*, *"migraine without aura"*),
ichdSymptom(*Id_sym*, *"headache"*), −*symptom*(*Id_sym*).
r_{13} : −*subCriterion*(*Id*, *"C"*, 1) ← *ichdDiagnosis*(*Id*, *"migraine without aura"*),
ichdSymptom(*Id_sym*, *"headache"*), *symptom*(*Id_sym*),
−*symAttribute*(*Id_sym*,*Id_attr*), *ichdAttribute*(*Id_attr*, *"unilateral location"*).

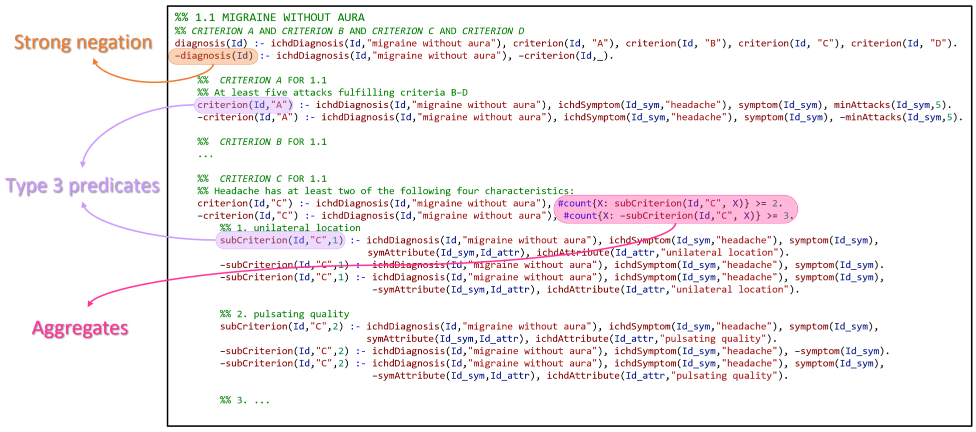
In this case, for a more natural modeling, it was convenient to use the aggregation construct *#count*. In particular, the rule r_9 aggregates the satisfied subcriteria and counts them (*#count*) in order to check if the resulting value matches the number required by the criterion itself.

```
1. Migraine
    1.1 Migraine without aura
    1.2 Migraine with aura
        1.2.1 Migraine with typical aura
            1.2.1.1 Typical aura with headache
            1.2.1.2 Typical aura without headache
        1.2.2 Migraine with brainstem aura
        1.2.3 Hemiplegic migraine
            1.2.3.1 Familial hemiplegic migraine (FHM)
                1.2.3.1.1 Familial hemiplegic migraine type 1 (FHM1)
                1.2.3.1.2 Familial hemiplegic migraine type 2 (FHM2)
                1.2.3.1.3 Familial hemiplegic migraine type 3 (FHM3)
                1.2.3.1.4 Familial hemiplegic migraine, other loci
            1.2.3.2 Sporadic hemiplegic migraine (SHM)
        1.2.4 Retinal migraine
    ...
```

Figure 4.1: Some ICHD-3 diagnoses.

4.3 Examples of Encoded Diagnoses

The designing of the formal methodology to encode each single diagnostic criterion into ASP logical rules, on the basis of the mentioned data model, can be observed in Figure 4.2. It shows the inference rules of the diagnosis "migraine without aura", in which we can notice the different encoding methodology adopted in order to model semantically different criteria. It can be noted that the rules that express the conditions according to which the predicate in the head is certainly false can be automatically generated by denying, one at a time, the predicates of type 2. Moreover, it is worth noting that predicates in the head of the rules are of type 3.



```
%% 1.1 MIGRAINE WITHOUT AURA
%% CRITERION A AND CRITERION B AND CRITERION C AND CRITERION D
diagnosis(Id) :- ichdDiagnosis(Id,"migraine without aura"), criterion(Id, "A"), criterion(Id, "B"), criterion(Id, "C"), criterion(Id, "D").
-diagnosis(Id) :- ichdDiagnosis(Id,"migraine without aura"), -criterion(Id,_).

    %%  CRITERION A FOR 1.1
    %% At least five attacks fulfilling criteria B-D
    criterion(Id,"A") :- ichdDiagnosis(Id,"migraine without aura"), ichdSymptom(Id_sym,"headache"), symptom(Id_sym), minAttacks(Id_sym,5).
    -criterion(Id,"A") :- ichdDiagnosis(Id,"migraine without aura"), ichdSymptom(Id_sym,"headache"), symptom(Id_sym), -minAttacks(Id_sym,5).

    %%  CRITERION B FOR 1.1
    ...

    %%  CRITERION C FOR 1.1
    %% Headache has at least two of the following four characteristics:
    criterion(Id,"C") :- ichdDiagnosis(Id,"migraine without aura"), #count{X: subCriterion(Id,"C", X)} >= 2.
    -criterion(Id,"C") :- ichdDiagnosis(Id,"migraine without aura"), #count{X: -subCriterion(Id,"C", X)} >= 3.
        %% 1. unilateral location
        subCriterion(Id,"C",1) :- ichdDiagnosis(Id,"migraine without aura"), ichdSymptom(Id_sym,"headache"), symptom(Id_sym),
                                  symAttribute(Id_sym,Id_attr), ichdAttribute(Id_attr,"unilateral location").
        -subCriterion(Id,"C",1) :- ichdDiagnosis(Id,"migraine without aura"), ichdSymptom(Id_sym,"headache"), symptom(Id_sym).
        -subCriterion(Id,"C",1) :- ichdDiagnosis(Id,"migraine without aura"), ichdSymptom(Id_sym,"headache"), symptom(Id_sym),
                                   -symAttribute(Id_sym,Id_attr), ichdAttribute(Id_attr,"unilateral location").

        %% 2. pulsating quality
        subCriterion(Id,"C",2) :- ichdDiagnosis(Id,"migraine without aura"), ichdSymptom(Id_sym,"headache"), symptom(Id_sym),
                                  symAttribute(Id_sym,Id_attr), ichdAttribute(Id_attr,"pulsating quality").
        -subCriterion(Id,"C",2) :- ichdDiagnosis(Id,"migraine without aura"), ichdSymptom(Id_sym,"headache"), -symptom(Id_sym).
        -subCriterion(Id,"C",2) :- ichdDiagnosis(Id,"migraine without aura"), ichdSymptom(Id_sym,"headache"), symptom(Id_sym),
                                   -symAttribute(Id_sym,Id_attr), ichdAttribute(Id_attr,"pulsating quality").

        %% 3. ...
```

Figure 4.2: An example of encoded diagnosis.

5 THE LOGICAL DECISION MODULE

As already mentioned, HEAD-ASP consists of two logical modules expressed through the Answer Set Programming formalism; the first one, discussed in the Chapter 4, is a deductive module which faithfully encodes all primary headache diagnoses and criteria of ICHD-3, whereas the second one is a logical module which optimizes the diagnostic process. This optimization has been achieved thanks to the development of a system implementing a questionnaire that: (i) rigorously guides both clinicians and patients during the medical history process; (ii) automatically adapts to patients; and (iii) reaches, in a reasonable amount of time, a complete diagnostic picture by inferring every diagnosis as either *compatible* or *not compatible*. The logic module that implements the interactive questionnaire is discussed in detail in the following sections. From a general point of view, the operating principle of the questionnaire is based on an algorithmic process implemented in the system.

The algorithm takes as input the ASP encoding of ICHD-3, as described in the previous sections, and the ASP encoding that implements the questionnaire logic, as described in the next paragraphs. The output is composed of the set of *compatible* diagnoses and the set of *not compatible* diagnoses according to the history of answers provided by the patient. The history of answers is collected during the process. Initially, the algorithm retrieves the set of all possible encoded diagnoses. At each iteration, first the algorithm computes the current diagnostic status, i.e., the current set of *compatible* diagnoses and the current set of *not compatible* diagnoses, by running the ICHD-3 encoding together with the history of answers provided by the patient. At this point, the algorithm checks whether there are still diagnoses that are *not determined*: in the negative case it finishes, and in the positive case it computes the next question to be posed to the patient. There are $2n$ *answer sets*, where n is the number of questions selected at the current step (further details later in this section): given a selected question, there is exactly one *answer set* where it is asked and the answer is affirmative, and exactly one *answer set* where it is asked and the answer is negative. Within the process, the algorithm computes, heuristically, the best question to pose to the patient, according to the worst-case minimization strategy discussed below. For each question, the algorithm finds the number of determined diagnoses if answered affirmatively and the number of determined diagnoses if answered negatively in the worst-case scenario, i.e., the selected question is the question having the maximum score. As a result of this process, the selected best question is posed to the patient and their answer is added to the history of answers.

In the next sections we analyze the portions of the ASP encoding used to implement the questionnaire.

5.1 ASP Encoding

In the following, we enumerate and give an informal semantics of the predicates defined in the ASP program that implements the questionnaire.

Based on the notions that appear in the diagnostic criteria classified in the previous section and that will be the subject of the questions to be asked, we identified a set of topics that allow us to split the set of potential questions into groups. In the encoding that implements the questionnaire we listed such topics in the form of instances of the predicate *topic(T, D)*, where, the variable *T* is instantiated by the constants that identify the topics, and *D* is either instantiated by the constant *dependent* or *independent* to indicate that a topic is related to a symptom or not. An example of a ground instance of the predicate *topic* is *topic(duration, dependent)*, which represents that *duration* is a dependent notion (e.g., the duration of headache or nausea makes sense, while duration alone does not). On the other hand, a symptom is a notion that is not dependent on another concept and we express it by the ground instance *topic(symptom, independent)*. The instances of the predicate *criterionDependsOn(Id_diag, Letter, X, Y, Topic)* express the dependence of a diagnostic criterion *Letter* of a diagnosis *Id_diag*, on the elements *X* and *Y* that characterize the topic *Topic* and that will constitute the possible subject of a question. For example, we use *criterionDependsOn (d.2.1, "D", s33, "nausea", symptom)* to denote that the criterion "D" of the diagnosis "d.2.1" depends on the symptom "nausea", whose identifier is *s33*. Each possible subject of the question is collected in the ground instances of the predicate *possible* through the following rule:

$$r_{14} : possible(X,Y,Topic) \leftarrow criterionDependsOn(Id\_diag, Letter, X, Y, Topic).$$

The predicate *possible* is then used to generate the predicate *ask(X, Y, Topic)* in the portion of the disjunctive program in which we evaluate asking potential questions. To the independent type topics correspond instances of the predicate *possible* in which *X* and *Y* are the characterizing elements, identified and formalized in the previous section. As an illustration, the characterizing elements of the topic *exam* are its name and the relative report, so, an example of its instantiation is *possible("gene CACNA1A", "presence of mutation", exam)*. In the case of independent type topic, it is necessary to represent the name of the symptom the topic refers to. Therefore, the variable *X* is instantiated by the name of the symptoms and the variable *Y* by the elements that characterize the topic itself. An example for the topic *duration* is the atom *possible("headache", 5, duration)* where, "5" is one of the values, belonging to the considered domain, expressed in minutes.

5.1.1 Pruning the Search Space

The initial portion of the ASP encoding that implements the questionnaire is based on the pruning of the search space, i.e., from the set of all possible questions to be asked to the patient, the "relevant" questions are selected. We define "relevant" an element present in a criterion relating to at least one diagnosis not yet determined and relating to a diagnosis "child" (according to predicate *isA*) whose diagnosis "father" has been confirmed.

$$\begin{aligned}r_{15}: & \; relevant(X, Y, Topic) \leftarrow criterionDependsOn(Id\_diag, Letter, X, Y, Topic), \\ & not\ criterion(Id\_diag, Letter),\ not\ -diagnosis(Id\_diag), \\ & isA(Id\_diag, Id\_sup\_diag),\ diagnosis(Id\_sup\_diag).\end{aligned}$$

5.1.2 Selecting a Candidate Question

The logic module that implements the interactive questionnaire is structured by following the "Guess and Check" programming paradigm of ASP. The "Guess" part of the questionnaire program is composed of a set of disjunctive rules that allow to consider the different possibilities for the choice of the next question. On the other hand, the "Check" part consists of a set of constraints that discard the unwanted models (e.g., models which ask more than one question simultaneously, models in which the question has already been asked or models in which the question is no more relevant). Beside disjunctive rules and constraints, ASP programs are also composed of a set of normal rules (i.e., rules with a single atom in the head) that are used to propagate knowledge in the program (e.g., propagate the consequences of an answer). The program identifies a set of n questions that could be asked; it is necessary to specify that each possible question is formulated as binary, i.e., patient's answers are simply "yes" or "no" (to overcome the typical fact that patients are not always able to adequately describe their disorders). Therefore, in order to analyze separately the consequences of the *2* possible answers from the patient, it generates *2n answer sets*. In other words, for each of the n possible questions, the program outputs an *answer set* that represents the outcome of the affirmative answer and an *answer set* that represents the outcome of the negative answer. Overall, an *answer set* represents a possible world in which the system can evolve once the patient answers another question. The following disjunctive rules compose the "Guess" part. In particular, for each topic T considered, we choose it or we don't choose it (r_{16}), for an *independent* type topic T, we ask, or we don't, a possible question regarding T (r_{17}), given a *dependent* type topic T and a possible question regarding it, after verifying the presence of the symptom the topic refers to, we ask the question or we don't (r_{18}), and we exclude the evaluation, in the same set, of more than one question (r_{19}):

$$\begin{aligned}r_{16}: & \; chosenTopic(T) | -chosenTopic(T) \leftarrow topic(T,D). \\ r_{17}: & \; ask(X,Y,T) | -ask(X,Y,T) \leftarrow chosenTopic(T),\ possible(X,Y,T), \\ & \quad topic(T,independent). \\ r_{18}: & \; ask(Name,Y,T) | -ask(Name,Y,T) \leftarrow chosenTopic(T), possible(Name,Y,T), \\ & \quad topic(T,dependent),\ ichdSymptom(Id,Name),\ symptom(Id). \\ r_{19}: & \leftarrow not\ \#count\{X,Y,T : ask(X,Y,T)\} = 1.\end{aligned}$$

5.1.3 Evaluating the Impact of a Candidate Question

The patient's answers will be collected in the ground instances of the predicate *answer(X, Y, Topic, Answer, Type)* where *Answer* is either the constant true or false and the variable *Type* is instantiated by the constant real or simulated to distinguish between the history of answers actually given by the patient and the answers simulated during the evaluation phase for the choice of the next question (r_{20}):

$$r_{20} : answer(X,Y,T,false,simulated) | answer(X,Y,T,true,simulated) \leftarrow ask(X,Y,T).$$

After selecting a question to be asked, the program outputs an answer set that represents the outcome of the affirmative answer and an answer set that represents the outcome of the negative answer. By simulating the patient's answer, we evaluate the effect that such answer would have in the process of determining the diagnoses. The information that would be acquired through each simulated answer is used to activate the predicates which, once propagated in the ICHD encoding, contribute, together with the previously collected data, to infer the eventual *compatible* and *not compatible* diagnoses. The patient's affirmative answer concerning a symptom activates the predicate which models its presence.

$$r_{21} : symptom(Id) \leftarrow answer(Id,\_,symptom,true,\_).$$

On the other hand, the patient's negative answer concerning a symptom allows us to infer the absence (modeled with strong negation) of the predicate which models its presence.

$$r_{22} : -symptom(Id) \leftarrow answer(Id,\_,symptom,false,\_).$$

Finally, we compute the number of diagnoses that would be *determined*. To do so, we use the aggregation construct *#count*: we count the diagnoses that would be *compatible* and those ones that would be *not compatible*.

$$r_{23} : compatibleDiag(N) \leftarrow \#count\{Id : diagnosis(Id)\} = N.$$
$$r_{24} : notCompatibleDiag(N) \leftarrow \#count\{Id : -diagnosis(Id)\} = N.$$

The number of potential *determined* diagnoses is then inferred by the following rule:

$$r_{25} : determinedDiag(D) \leftarrow compatibleDiag(N),\ notCompatibleDiag(M),\ D = N + M.$$

5.2 The Next Question Strategy

Such a logic module allows, at each step of the questionnaire, to identify the convenient *next question* described in the following. We discard from the set of candidate next questions those that are inappropriate or irrelevant (an example of an inappropriate question is a question related to the presence of the symptom "diplopia" if the patient already reported not to have the symptom "visual disorder"; an example of an irrelevant question is a question concerning a diagnosis that has already been inferred as *not compatible*.) The resizing of the admissible questions set, in order to let them conform to the history of answers, is made possible via the usage of some constraints. In particular, not relevant questions cannot be asked (r_{26}), a question that has been previously answered cannot be asked (r_{27}), and it is not possible to consider, in a question, a numerical value for the *duration* of a symptom that is not in the previously identified range (r_{28}):

$$\begin{aligned}
r_{26}:\ &\leftarrow ask(X,Y,T),\ not\ relevant(X,Y,T).\\
r_{27}:\ &\leftarrow ask(X,Y,T),\ answer(X,Y,T,\_,real).\\
r_{28}:\ &\leftarrow ask(Name,V,duration),\ ichdSymptom(Id,Name),\ symptom(Id),\\
&\quad minDuration(Id,Y),\ V<Y.
\end{aligned}$$

At this point, we implement a greedy strategy that consists in computing, for each candidate next question, the minimum number of determined diagnoses resulting from the double choice of affirmative and negative answer, and then, the next question is the one corresponding to the maximum value.

6 SYSTEM IMPLEMENTATION AND TESTING

In this chapter we will show the architecture of the system from a technological point of view. The decision support system has been implemented into two Web applications: a REST Web service, and a Web graphical interface as will be detailed in Section 6.1. Furthermore, in Section 6.2 we will report about the results obtained on 3000 questionnaires that led to at least one headache disorder, and so, which are representative of realistic headache anamneses.

6.1 The System Architecture

The decision support system has been implemented into two Web applications: a REST Web service, and a Web graphical interface. Web applications are one of the most common types of distributed applications. In general, they are accessible without requiring any installation process and are cross-platform by design.

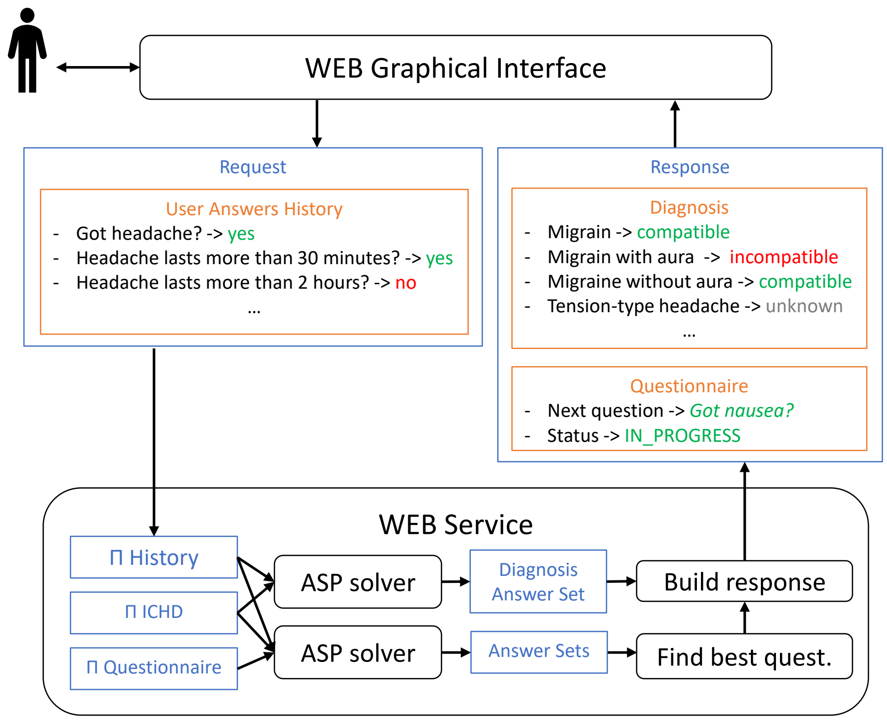


Figure 6.1: System architecture

Figure 6.1 presents, with an example, the architecture of Head-ASP. The user (a physician) interacts directly with the Web Graphical Interface which communicates under the form of HTTP requests and responses with the Web Service, i.e., it sends the patient's answer history to the web service, making use of an http request. At this point, the HEAD-ASP web service (i) builds an ASP program as the union of the encoding of ICHD, of the encoding that implements the questionnaire and of the history of answers provided by the patient, and (ii) runs it on an ASP system which outputs the candidate answer sets. Finally, the web service selects the next question, updates the diagnostic picture and communicates the output via HTTP response to the Graphical Interface, which will dynamically update. In this project DLV2 has been used as the ASP solver of the application. It is worth noting that the Web Service is not intended to be used directly by humans, but it is, instead, intended to be invoked by other programs. It has been implemented in JAVA using *Spring*, which is a well-known framework for Web development.

The implemented service follows the REST architectural style [RR08]. As per the REST architecture, a RESTful Web Service is stateless, i.e., it has no concept of state or session, and thus it can scale well horizontally [MMSW07]: more instances can be deployed simultaneously and it is not important which instance is handling a request. The Web Service also exposes a documentation that shows its communication protocol, i.e., how to invoke it and how to interpret its output. In our implementation, the Web Service exposes a method that accepts a patient history of answers and returns the current diagnosis and the next question. The response contains special tokens in the cases where the questionnaire is completed (i.e., every diagnosis is either *compatible* or *not compatible*) or the questionnaire can not be continued (i.e., there is no relevant question to ask).

The HEAD-ASP Web Graphical Interface demonstrates the functionalities of the DSS and, contrary to the Web Service, it is intended to be used by humans. Currently, it is used as the primary interface of HEAD-ASP until the future integration of the DSS in the *Alcmeone* project platform. To develop the Web Interface we used *Angular 8*, which is a popular *JavaScript* framework for the development of Web graphical interfaces. The Web interface and the Web service (and its documentation) are available at `https://head-asp.github.io/ichd-dss/`.

6.2 Verification and Validation of the Approach

To minimize the number of defects in the implementation and investigate the performance of the approach, we implemented a testing framework where questions are answered randomly. In this section, we report the results obtained by the current release of HEAD-ASP on approximately 7400 questionnaires. Among these, we focused on those –3000 in total– that led to at least one headache disorder, which are representative of real cases of headache. Figures 6.2 and 6.3 show two histograms summarizing the results of our tests.

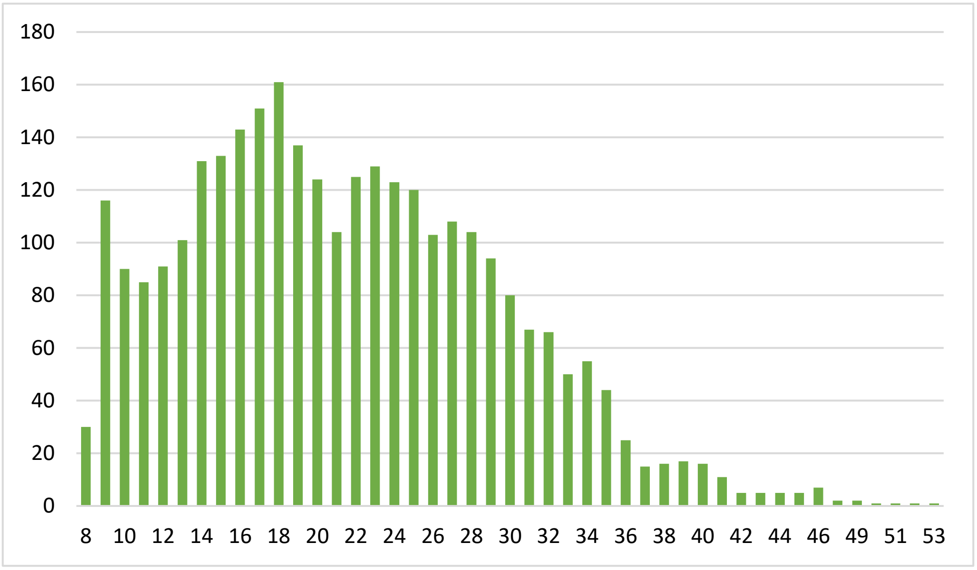


Figure 6.2: Reaching the complete diagnostic picture.

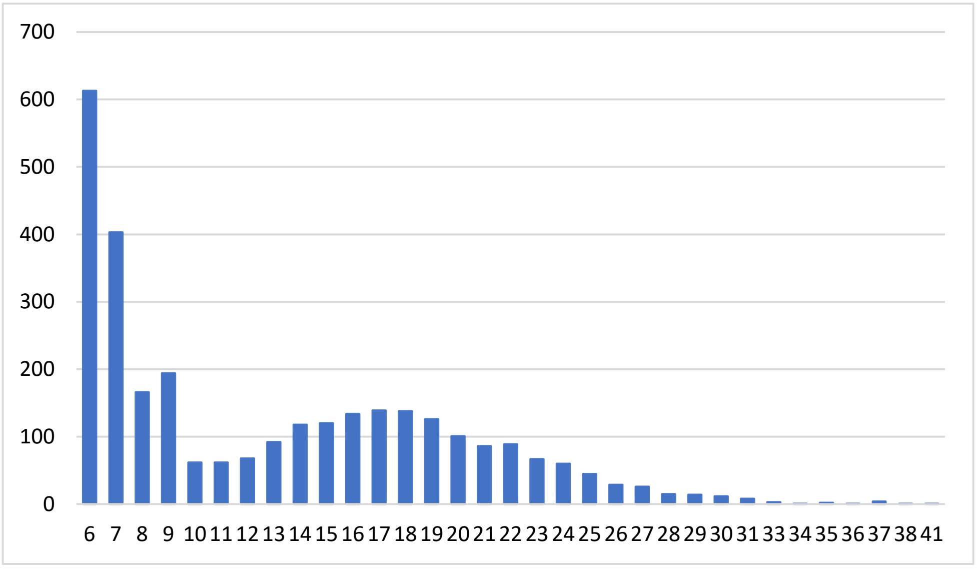


Figure 6.3: Reaching the first compatible diagnosis.

In both histograms, each bar on top of a value x represents how many questionnaires had a length x. In Figure 6.2, we plot the distribution of the number of questionnaires by overall length (i.e., the length necessary to reach a complete diagnostic picture); in Figure 6.3, we consider the distribution of the number of questionnaires by the length at which the first compatible diagnosis has been reached. The average length is of 21.44 questions in the first case and of 12.99 in the second one. It is worth noting that these results are in line with the neurologists' experience and expectations. Finally, considering the fact that there are more than 150 candidate questions encoded in the system, the results of our experiment reveal how HEAD-ASP is able to effectively discard unnecessary questions when needed, pruning the search space and producing short questionnaires.

Part III

RELATED WORK AND CONCLUSION

7 RELATED WORK

In this section we present several works concerning decision support systems for the diagnosis of headaches that can be found in the literature. In the field concerning decision support systems for headache diagnosis a large amount of works can be found. Among the most recent ones, it is possible to cite a work presented in 2014 by researchers from the University of Zhejiang [DYH+14]. The authors developed a headache computerized *clinical decision support systems (CDSS)* based on ICHD-3 beta and validated it in a study that involved 543 patients, affected by headache, from the International Headache Center at the Chinese PLA General hospital, Beijing, China. This work is based on a 3-steps translation of ICHD-3: first in terms of flow-charts, then in terms of an ontological model and, finally, in terms of rules. More in details, the paper proposes a method of constructing a computerized clinical guideline model and medical knowledge-base achieved by exploiting the joint efforts of clinical specialists and knowledge engineers, according to a translation process that allowed to express ICHD-3 in the form of a flowchart. In order to be directly run by computer, the computerized guidelines were translated into rules that could be executed in the inference engine of the CDSS. On the basis of clinical information for headache disorders fed as input to the CDSS by inexperienced doctors, the system made a computerized diagnosis. Qualified and experienced specialists in headache neurology reviewed this information and made the final diagnosis. The system is described mostly from an architectural point-of-view and in-depth details of the translation and of the diagnosis process are not provided.

An efficient technique for knowledge-based decision support even in domains involving medicine is the *rule-based fuzzy logic (RBFL)*. In this scenario, we can cite a system implemented at the Clinical Centre Vojvodina, Institute of Neurology in Novi Sad [SSSSI08] and developed as a tool for diagnosing some types of primary headaches in workers using the rule-based fuzzy logic. The system does not include all diagnoses but is limited to some specific types of primary headache (migraine without aura, migraine with aura, tension-type headache and other primary headaches). The researchers show the workflow of the basic model of rule-based fuzzy logic systems, in which the rules are expressed as a collection of IF-THEN statements. In particular, the information can be extracted by the patients in the form of IF-THEN statements; finally, these rules can be modeled using fuzzy logic system. The rules are provided according to the *International Headache Disorder Criteria (IHDC)* and, once established, the system can be viewed as an input-to-output mapping. Questions based on the *IHDC* represent the input of the model while the output represents the specific type of headache (migraine without aura, migraine with aura, tension type headache and other primary headaches).

A CDSS based on a hybrid intelligent reasoning method for primary headache disorder [YMLD] involves the conventional *Rule-Based Reasoning(RBR)* and the *Case-Based Reasoning(CBR)* techniques. It has been developed in order to help general practitioners to improve diagnostic accuracy of primary headache disorders. The decision process able to decide the headache disorder can be divided into two steps. RBR is the first one and concerns rules coming from clinical guidelines for headaches *International Classification of Headache Disorders - 3rd Edition* which can be executed by the inference engine of CDSS. However, in cases regarding the differentiation of the headaches with atypical symptoms, the system is not able to get an accurate diagnostic result. For this reason, a second step including CBR techniques is necessary to improve the accuracy of the entire diagnostic process. CBR is an artificial intelligent technology considered to be one of the most effective ways when dealing with implicit knowledge whose core idea is solving new problems based on the solutions of similar past problems. The authors show the architecture of the hybrid intelligent system which consists of the two aforementioned modules (RBR and CBR), emphasizing the importance of the CBR module as a supplementary technique to RBR module.

Our work has also connections with *AIDA Cefalee*, a system presented in 2004 from the researchers of University Federico II of Naples [DSCR+07, DSMB04]. AIDA Cefalee [DSMB04], is a database for the storage of symptoms and diagnosis data of patients with headache disorders, and includes a diagnostic module that can suggest possible diagnosis (only if all symptoms have been acquired). The core of AIDA Cefalee is an expert diagnostic system, based on the *International Classification of Headache Disorders* - 2*nd* *edition* (*ICHD-II*), able to suggest the correct diagnosis once all the clinical features of a patient's headache have been collected. The system contains a sophisticated database that can be synchronized over the network allowing a continuous sharing of the patients' information and a cooperation between different research groups. The development of the system was part of a validation study which involved five Italian headache centres that have selected clinical data stored on previously diagnosed primary headache cases. The data were, then, entered into the AIDA diagnostic tool in order to compare the tool diagnostic accuracy with the diagnosis reached by the standard clinical method (SCM). The diagnostic tool has been validated experimentally [DSCR+07], but no details of the classification method are provided.

Our system is also related to a computerized program designed to diagnose primary headache based on *ICHD-II* criteria [EREHN+13]. The latter implements a questionnaire divided into five main parts. Within the two most important steps, the system provides questions related to headache disorders and, eventually, shows the final decision with a detailed explanation of it. This system uses simple human-like algorithmic logic to derive the most appropriate type of headache. However, the accuracy of the diagnosis depends also on the accuracy of the patient's response. The article does not provide further details on the classification method and the system is not publicly available.

In [VDBL+18] the authors present a DSS for the diagnosis of headaches that consists of three modules: (i) a mobile application that captures symptomatic data from patients, (ii) an automated diagnosis support module that generates an interpretable decision tree, based on data semantically annotated with expert knowledge and (iii) a web application that helps the physician to efficiently interpret captured data and learned insights by means of visualizations. Here, the diagnosis process is based on supervised machine learning models. One of the most important modules of the proposed decision support system is an automated diagnosis support module. In this module, an interpretable predictive model is generated from the data collected by the mobile application, using supervised classification. Supervised classification is a sub-domain of machine learning. Additionally, a knowledge base is constructed using expert knowledge, the ICHD document and ontologies such as *SNOMED*. Both the collected data and the prior knowledge is used to generate feature vectors which are fed to the machine learning technique. Before feeding them, the class distribution in the training dataset is balanced in order to make it more uniform.

8 CONCLUSION

In this work, we have presented HEAD-ASP: a novel decision support system for the diagnosis of headache disorders (one of the most common and disabling conditions of the nervous system throughout the world). The system is currently tested by a group of neurologists that are profitably using it within *Alcmeone* and are leaving very positive feedback.

Although design and implementation of the DSS was quite challenging, we found very natural to use logic programming both to encode the ICHD classification and to model the heuristics for determining the next question. In particular, the advantages can be summarized as follows:

- simple and complex diagnoses can be encoded in a natural and precise way;
- ICHD updates can be easily and locally transferred to the encoding;
- medical knowledge can be represented and integrated in a declarative way;
- identifying candidate questions and simulating their effects are tasks that can be carried out in the same framework.

Concerning our future plans, the ultimate objective is to promote the system inside the Italian Society for the Study of Headache (SISC) so that it can become a valid support to clinicians and specialists. To this end, we are still refining and improving it according to the feedback we are currently receiving. In particular, the next steps include:

- completing the ICHD-3 encoding;
- analyzing the next-question problem from a theoretical perspective;
- further reducing the average number of questions needed to reach a complete diagnosis.

Furthermore, we would like to generalize our methodology to be easily applicable in similar contexts.

BIBLIOGRAPHY

[ABW88] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, 1988.

[BET11] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[BLR00] Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Enhancing disjunctive datalog by constraints. *IEEE Trans. Knowl. Data Eng.*, 12(5):845–860, 2000.

[CCIL08] Francesco Calimeri, Susanna Cozza, Giovambattista Ianni, and Nicola Leone. Computable functions in ASP: theory and implementation. In Maria Garcia de la Banda and Enrico Pontelli, editors, *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings*, volume 5366 of *Lecture Notes in Computer Science*, pages 407–424. Springer, 2008.

[CR96] Alain Colmerauer and Philippe Roussel. The birth of prolog. In *History of programming languages—II*, pages 331–367. 1996.

[Don76] John J. Donovan. Database system approach to management descision support. *ACM Trans. Database Syst.*, 1(4):344–369, 1976.

[DSCR$^+$07] R De Simone, G Coppola, A Ranieri, G Bussone, P Cortelli, D D'Amico, F d'Onofrio, GC Manzoni, E Marano, F Perini, et al. Validation of aida cefalee, a computer-assisted diagnosis database for the management of headache patients. *Neurological Sciences*, 28(2):S213–S216, 2007.

[DSMB04] R De Simone, E Marano, and V Bonavita. Towards the computerisation of ANIRCEF Headache Centres. Presentation of AIDA CEFALEE, a computer assisted diagnosis database for the management of headache patients. *Neurological Sciences*, 25(3):s218–s222, 2004.

[DYH$^+$14] Zhao Dong, Ziming Yin, Mianwang He, Xiaoyan Chen, Xudong Lv, and Shengyuan Yu. Validation of a guideline-based decision support system for the diagnosis of primary headache disorders based on ICHD-3 beta. *The journal of headache and pain*, 15(1):40, 2014.

[EFLP00] Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Declarative problem-solving using the dlv system. In *Logic-based artificial intelligence*, pages 79–103. Springer, 2000.

[EIK09a] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer set programming: A primer. In Sergio Tessaris, Enrico Franconi, Thomas Eiter, Claudio Gutiérrez, Siegfried Handschuh, Marie-Christine Rousset, and Renate A. Schmidt, editors, *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School 2009, Brixen-Bressanone, Italy, August 30 - September 4, 2009, Tutorial Lectures*, volume 5689 of *Lecture Notes in Computer Science*, pages 40–110. Springer, 2009.

[EIK09b] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. *Answer Set Programming: A Primer*, pages 40–110. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[EIK09c] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer set programming: A primer. In Sergio Tessaris, Enrico Franconi, Thomas Eiter, Claudio Gutiérrez, Siegfried Handschuh, Marie-Christine Rousset, and Renate A. Schmidt, editors, *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School 2009, Brixen-Bressanone, Italy, August 30 - September 4, 2009, Tutorial Lectures*, volume 5689 of *Lecture Notes in Computer Science*, pages 40–110. Springer, 2009.

[EREHN$^+$13] Vahid Eslami, Sadreddin Rouhani-Esfahani, Nima Hafezi-Nejad, Farshid Refaeian, Siamak Abdi, and Mansoureh Togha. A computerized expert system for diagnosing primary headache based on International Classification of Headache Disorder (ICHD-II). *Springerplus*, 2(199):1–4, 2013.

[Eva17] Randolph W. Evans. Incidental findings and normal anatomical variants on MRI of the brain in adults for primary headaches. *Headache: The Journal of Head and Face Pain*, 57(5):780–791, 2017.

[FLP04] Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In *JELIA*, volume 3229 of *LNCS*, pages 200–212. Springer, 2004.

[FPL11] Wolfgang Faber, Gerald Pfeifer, and Nicola Leone. Semantics and complexity of recursive aggregates in answer set programming. *Artif. Intell.*, 175(1):278–298, 2011.

[GL88] Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics for Logic Programming. In *Logic Programming, Proceedings of the*

Fifth International Conference and Symposium, Seattle, WA, Aug 15-19, 1988 (2 Volumes), pages 1070–1080, Cambridge, Mass., 1988. MIT Press.

[GL91] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–386, 1991.

[IHS18] International-Headache-Society. Headache classification committee of the international headache society (IHS) the international classification of headache disorders, 3rd edition. *Cephalalgia*, 38(1):1–211, 2018.

[KB83] Robert A Kowalski and Kenneth A Bowen. Logic programming. In *IFIP Congress*, pages 133–145. Citeseer, 1983.

[Kow88] Robert A Kowalski. The early years of logic programming. *Communications of the ACM*, 31(1):38–43, 1988.

[LGP$^+$90] Douglas B. Lenat, Ramanathan V. Guha, Karen Pittman, Dexter Pratt, and Mary Shepherd. CYC: toward programs with common sense. *Commun. ACM*, 33(8):30–49, 1990.

[Lif99] Vladimir Lifschitz. Answer set planning. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 373–374. Springer, 1999.

[Lif02] Vladimir Lifschitz. Answer set programming and plan generation. *Artif. Intell.*, 138(1-2):39–54, 2002.

[Lif08] Vladimir Lifschitz. What is answer set programming? In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 1594–1597. AAAI Press, 2008.

[Llo12] John W Lloyd. *Foundations of logic programming*. Springer Science & Business Media, 2012.

[LPF$^+$06a] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, Jul 2006.

[LPF$^+$06b] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV System for Knowledge Representation and Reasoning. 7(3):499–562, July 2006.

[LPV01] Vladimir Lifschitz, David Pearce, and Agustín Valverde. Strongly equivalent logic programs. *ACM Trans. Comput. Log.*, 2(4):526–541, 2001.

[McC80] John McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial intelligence*, 13(1-2):27–39, 1980.

[MMSW07] Maged Michael, Jose E Moreira, Doron Shiloach, and Robert W Wisniewski. Scale-up x scale-out: A case study using nutch/lucene. In *Proc. of IPDPS'07*, pages 1–8, 2007.

[Moo85] Robert C. Moore. Semantical considerations on nonmonotonic logic. *Artif. Intell.*, 25(1):75–94, 1985.

[MT99] Victor W. Marek and Miroslaw Truszczynski. Stable models and an alternative logic programming paradigm. In Krzysztof R. Apt, Victor W. Marek, Mirek Truszczynski, and David Scott Warren, editors, *The Logic Programming Paradigm - A 25-Year Perspective*, Artificial Intelligence, pages 375–398. Springer, 1999.

[Nie99] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Math. Artif. Intell.*, 25(3-4):241–273, 1999.

[Prz91] Teodor C. Przymusinski. Stable semantics for disjunctive programs. *New Gener. Comput.*, 9(3/4):401–424, 1991.

[Rei80] Raymond Reiter. A logic for default reasoning. *Artificial intelligence*, 13(1-2):81–132, 1980.

[RR08] Leonard Richardson and Sam Ruby. *RESTful web services*. O'Reilly Media, Inc., 2008.

[SHJ+07] LJ Stovner, K Hagen, R Jensen, Z Katsarava, RB Lipton, AI Scher, TJ Steiner, and J-A Zwart. The global burden of headache: A documentation of headache prevalence and disability worldwide. *Cephalalgia*, 27(3):193–210, 2007.

[SPB+20] Reed T. Sutton, David Pincock, Daniel C. Baumgart, Daniel C. Sadowski, Richard N. Fedorak, and Karen I. Kroeker. An overview of clinical decision support systems: benefits, risks, and strategies for success. *npj Digit. Med*, 3(1), 2020.

[SSSSI08] Svetlana Simić, Dragan Simić, Petar Slankamenac, and Milana Simić-Ivkov. Computer-assisted diagnosis of primary headaches. In *Proc. of HAIS'08*, pages 314–321, 2008.

[VDBL^{+}18] Gilles Vandewiele, Femke De Backere, Kiani Lannoye, Maarten Vanden Berghe, Olivier Janssens, Sofie Van Hoecke, Vincent Keereman, Koen Paemeleire, Femke Ongenae, and Filip De Turck. A decision support system to follow up and diagnose primary headache patients using semantically enriched data. *BMC Med Inform Decis Mak*, 18(1):98, 2018.

[VEK76] Maarten H Van Emden and Robert A Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM (JACM)*, 23(4):733–742, 1976.

[YMLD] Ziming Yin, Lingtong Min, Xudong Lu, and Huilong Duan. A clinical decision support system for primary headache disorder based on hybrid intelligent reasoning. In *Proc. of BMEI 2014*, pages 683–687.