

Modelling and Implementing a Knowledge Base for Checking Medical Invoices with DLV

Christoph Beierle¹, Oliver Dusso¹, Gabriele Kern-Isberner²

¹Dept. of Computer Science, FernUniversität in Hagen, 58084 Hagen, Germany

²Dept. of Computer Science, University of Dortmund, 44221 Dortmund, Germany

beierle@fernuni-hagen.de, oliver.dusso@fernuni-hagen.de,

gabriele.kern-isberner@cs.uni-dortmund.de

Abstract. Checking medical invoices, done by every health insurance company, is a labor-intensive task. Both speed and quality of executing this task may be increased by the knowledge-based decision support system ACMI which we present in this paper. As the relevant regulations also contain various default rules, ACMI's knowledge core is modelled using the answer set programming paradigm. It turned out that all relevant rules could be expressed directly in this framework, providing for a declarative and easily extendable and modifiable knowledge base. ACMI is implemented using the DLV system.

1 Introduction

In contrast to Germany's compulsory health insurance, in the private health insurance system, a physician or a physiotherapist does not have a contractual relationship with the patient's insurance company, but only with the patient himself. He issues an invoice to the patient which the patient has to pay. Having a private health insurance, the patient will hand the doctor's bill to his insurance company for reimbursement. Typically, the insurance company will check whether the invoice obeys various legal and other regulations, in particular, whether it conforms to e.g. the *Gebührenordnung für Ärzte* (GOÄ, scale of fees for physicians) [4].

The checking of the invoices requires detailed knowledge of many regulations and is a labor-intensive task. When investigating the official regulations concerning medical invoices (e.g. [4], [5]) and the corresponding business rules of one of Germany's large insurance companies, it turned out that also various default rules are used. Thus, a high-level modelling of a knowledge base reflecting the regulations and rules should also allow for the formulation of defaults.

On the other hand, default reasoning requires a powerful inferences system. In recent years, the paradigm of answer set programming (ASP) [6] turned out to provide such a powerful inference system while supporting for logic rules, negation and defaults. Nowadays, there are various systems implementing the ASP paradigm (e.g. Smodels [7] or DLV [3]).

The purpose of this paper is twofold: to model and implement a knowledge based decision support system for checking medical invoices, and to investigate whether the language of ASP is suitable for this modelling. As a result of this investigation, we

developed the ACMI system (Automatic Checking of Medical Invoices) providing an automated decision support system for the checking of such invoices [2]. We will give an overview of the system, illustrate its knowledge base formulated in the formal logic language of ASP, and present a detailed example as a walk through the system implemented in DLV. Additionally, we develop three more abstract rule schemas using sets (which are not supported by DLV) and demonstrate that these are sufficient for modelling ACMI's knowledge base and that they can be translated automatically to DLV.


Herr Max Musterkrank Musterweg 1 11111 Musterdorf Kd-Nr. 100200300	Praxis Peter Praktiker Hustengasse 22 22222 Praktikerstadt Tel. 0123 / 4567- 0	Osteopath Heilpraktiker	
			Rechnungsnummer 400500600
Datum	GebüH-Ziffer	Leistungsbezeichnung	Gebühr
07.12.03	1	eingehende Untersuchung	12,30 €
	2	Durchführung des vollständigen Krankenexamens mit Repertorisation	15,40 €
	4	eingehende Beratung	16,40 €
	12.2	Hamuntersuchung	4,60 €
	20.8	Einreibungen zu therapeutischen Zwecken	5,50 €
	21.1	Akupunktur zur Schmerzbehandlung	10,30 €
	28.1	Behandlung mittels paravertebraler Infiltration	7,70 €
TOTAL			72,20 €

Fig. 1. An invoice handed in by the patient Max Musterkrank

2 Overview of the system

ACMI is embedded in an environment processing the workflow from the incoming patients invoices to the refunding decision done by the insurance company's person in charge. Figure 1 shows a typical invoice. The first column specifies the date when the treatment was carried out. The second column specifies a number from the respective *Gebührenordnung* (scale of fees). In this case, the invoice is from an alternative practitioner (*Heilpraktiker*) so the relevant *Gebührenordnung* is the *GebüH* (*Gebührenordnung für Heilpraktiker*, scale of fees for alternative practitioners) [5]. The third column gives a verbal description of the treatment corresponding to the fee number, and the fourth column contains the invoiced fee (in €) for that treatment (which is not fixed, but must be in a certain range specified by the *GebüH*).

The invoices are scanned in and processed by optical character recognition software. The fee numbers, the descriptions and the charged amounts are extracted and –

together with an assigned invoice number, the patient’s name, his insurance identification number and his tariff identification extracted from the corresponding database – are presented in a graphical user interface to the person in charge (Fig. 3). For every item with a fee number, ACMI then automatically performs checks based on its knowledge base and reports the results. In Fig. 2, ACMI’s knowledge sources are summarized.

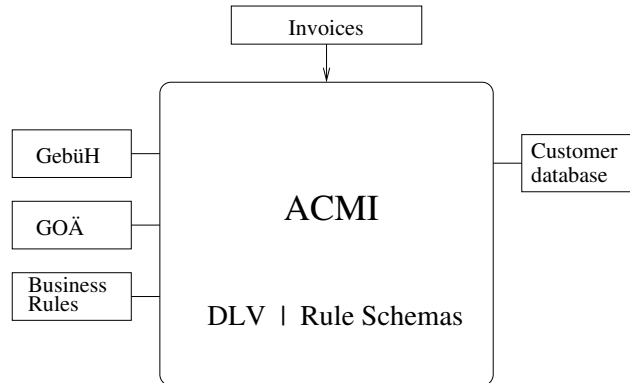


Fig. 2. ACMI’s knowledge sources

3 ACMI’s Knowledge Base

In the following, we restrict ourselves to ACMI’s knowledge base with respect to the GebüH [5]. The GebüH contains 148 different fee numbers (as a matter of fact, the GOÄ [4] contains many more). Each combination of fee numbers occurring in an invoice for a particular day of treatment must satisfy various constraints given in the GebüH, e.g.:

- C1** *Number 4 may not occur more than once, and it may occur simultaneously only with 1 or with 17.1.*
- C2** *Number 20.8 may not occur more than once.*
- C3** *If number 28.1 occurs more than once, replace all its occurrences with one occurrence of 28.2.*
- C4** *Number 12.9 may not occur simultaneously with 12.10 and 12.11.*
- C5** *Number 5.0 may not occur simultaneously with 6.0, 7.0 or 8.0.*

Note that there are constraints about the frequency of certain numbers, different restrictions about the simultaneous occurrences of combinations of numbers, and even prescriptions where numbers have to be replaced by another one.

In addition, there are internal business rules (cf. [1]) of the insurance company attached to a fee number, e.g:

T1 Number 21.1 is only refundable in insurance contracts with full coverage of the GebüH.

H1 For number 12.2 a notification text is to be displayed to the person in charge.

When considering all these regulations for the GebüH, 78 rules were identified. In addition, medical invoice checking in the insurance company is based on the following general rule for each fee number N :

R1 If a number N occurs in an invoice and N is not found to be invalid, then N is valid.

Note that this rule is a typical default rule as it is being investigated in formal default logic (e.g. [8]) and as it can be expressed neatly in the paradigm of answer set programming (ASP) [6]:

$valid(N) :- in_invoice(N), not \neg valid(N).$

This ASP rule can be read as R1 above: \neg is the classical logic negation operator (here yielding N invalid), and *not* is the default negation (here yielding N is not found to be invalid). Since also many other rules need negation and can be expressed quite straightforwardly in ASP, we decided to implement ACMI's knowledge representation and processing part in an ASP system like Smodels [7] or DLV [3].

Versicherte Person	Rechnungsnummer	Versicherungsnummer	Tarif
Max Musterkrank	400500600	100200300	VC

Gebührenziffer	Beschreibung	Rechnungsbetrag
1.	Für die eingehende, das gewöhnliche Maß übersteigende Untersuchung	12,30 €
2.	Durchführung des vollständigen Krankenexamens mit Repertorisation nach den Regeln der klassischen Homöopathie	15,40 €
4.	Eingehende Beratung, die das gewöhnliche Maß übersteigt, von mindestens 15 Minuten Dauer, gegebenenfalls einschließlich einer Untersuchung	16,40 €
12.2	Harnuntersuchung	4,60 €
20.8	Einreibungen zu therapeutischen Zwecken in die Haut	5,50 €
21.1	Akupunktur einschließlich Pulsdiagnose	10,30 €
28.1	Behandlung mittels paravertebraler Infiltration	7,70 €

Datum	07.12.2003	Forderung	72,20 €	Erstattung	?	Prüfen
-------	------------	-----------	---------	------------	---	---------------

Fig. 3. Graphical user interface with a preprocessed invoice

4 Answer Set Programming and DLV

Answer sets can be regarded as solutions to problems described by logic programs. A logic program consists of rules of the form

$$H \leftarrow A_1, \dots, A_n, \text{not } B_1, \dots, \text{not } B_m \quad (1)$$

where $H, A_1, \dots, A_n, B_1, \dots, B_m$ are literals, i.e. logical atoms in positive or negated form. Each such rule may be read as

If A_1, \dots, A_n all hold, and it can be assumed that none of B_1, \dots, B_m holds, then conclude H

Answer sets provide precise answers to any query by returning the truth values of logical atoms. The presence of default negation makes answer set programming quite complicated, as in general, several possible solutions have to be taken into account in an efficient way. However, it is just default rules like (1) that make logic programs concise and readable representations of knowledge. Default rules specify general knowledge while leaving room for exceptions. This requires a nonclassical logical framework which is provided by answer set programming in a convenient way. Answer set programming generalizes classical logic programming, so strict rules (without exceptions) may be used as well.

We will not go further into the details of answer set programming but refer the interested reader to other works on this topic (e.g. [6]).

DLV [3] is a system implementing answer set programming which has been developed at the universities of Vienna, Austria, and Calabria, Italy. As a special feature, it also allows for disjunctive information in the heads of logic program rules.

5 Design and Implementation of ACMI

Here, we describe some details of ACMI's implementation in DLV. Internally, each fee number is represented by a four digit number, e.g., 4 by 0400, 10 by 1000, and 20.8 by 2008. The occurrence of a number N at the I 'th position in the invoice is given by the literal $p(I, N)$. Thus, the invoice depicted in Figure 1 is internally represented by

```
p(1,0100) . p(2,0200) . p(3,0400) .  
p(4,1202) . p(5,2008) . p(6,2101) .  
p(7,2801) . tariff(vc) .
```

where the literal *tariff(vc)* reflects the patient's insurance tariff, being extracted from the corresponding data base.

5.1 Direct Realization by DLV rules

Each of ACMI's knowledge base rules can be expressed directly by DLV rules. For instance, the rules C1, C2, C3, C4, C5, T1, and H1 given above can be implemented

by:

```
%----- rule C1: -----
-z(0400) :- 1 < #count{I : p(I,0400)}.
-z(0400) :- 1 = #count{I : p(I,0400)},
            1 <= #count{I : p(I,Z),
            Z != 0001, Z != 1701}.

%----- rule C2: -----
-z(2008) :- 1 < #count{I : p(I,2008)}.

%----- rule C3: -----
-z(2801) :- 1 < #count{I : p(I,2801)}.
r(2802)  :- 1 < #count{I : p(I,2801)}.

%----- rule C4: -----
-z(1209) :- p(I,1209), p(I,1210),
            p(I,1211).

%----- rule C5: -----
-z(0500) :- p(I,0500), p(I,0600).
-z(0500) :- p(I,0500), p(I,0700).
-z(0500) :- p(I,0500), p(I,0800).

%----- rule T1: -----
t(2101)  :- p(I,2101), not fullGebuH.

%----- rule H1: -----
h(1202)  :- p(I,1202).
```

A literal of the form $z(N)$ (resp. $-z(N)$) indicates that N occurs in the invoice and should be refunded (resp. should not be refunded). A literal $r(N)$ says that N serves as a replacement for (one or more) other number occurrences. A literal $h(N)$ indicates that a notification text should be presented to the person in charge, and $t(N)$ says that the patient's tariff does not cover fee number N . Note that these rules use DLV's aggregate facility. For instance, $\#count\{I : p(I,2801)\}$ evaluates to the number of literals of the form $p(I,2801)$, i.e. the number of occurrences of fee number 2801 in the invoice. Rules like

```
fullGebuh :- tariff(an).
fullGebuh :- tariff(abc).
```

specify the tariffs providing full coverage of the GebüH. Incidentally, Max Musterkrank's tariff VC is not among them. The rule for T1 uses DLV's default negation facility: As long as *fullGebuh* is not known, fee number 2101 is not to be refunded. Also rule R1 (cf. Sec. 3) can be expressed easily using the DLV default rule:

Versicherte Person		Rechnungsnummer	Versicherungsnummer	Tarif
Max Musterkrank		400500600	100200300	VC
Gebührenziffer	Prüfungsergebnis	Erklärungstext	Zu viel berechnet	
1.	Okay			
2.	Okay			
4.	x REGELVERSTOß	REGELVERSTOß: wenn 4, dann nur mit 1 oder 17.1 zu einem Datum	-16,40 €	
12.2	Okay	Hinweis: Es ist anzugeben auf welchen Stoff untersucht wurde.		
20.8	Okay			
21.1	x Nicht im Tarifumfang		-10,30 €	
28.1	Okay			
Datum 07.12.2003		Forderung 72,20 €	Erstattung 45,50 €	Nächster Fall

Fig. 4. Result of checking an invoice

```
%----- rule R1: -----
z(N) :- p(I,N), not -z(N).
```

Using this DLV implementation, the following answer set is generated:

```
z(0100), z(0200), -z(0400), z(1202),
h(1202), z(2008), z(2101), t(2101), z(2801)
```

Thus, numbers 0100, 0200, 2008, and 2801 are valid and no further action has to take place. 1202 is also valid, but a notification text (indicated by h(1202)) has to be presented to the person in charge. However, 0400 falsifies a constraint (the second part of C1) and is not valid. 2101 is valid, but since tariff VC does not cover it (indicated by t(2101)), the fee will not be reimbursed. The answer set is processed by ACMI's backend and presented to the person in charge as illustrated in Fig. 4.

5.2 Realization with Rule Schemas

Whereas the previous subsection realizes a direct implementation of each of ACMI's 78 rules, we also developed a representation on a more abstract level. By investigating all individual rules, the following three general types of rule schemas could be identified when checking a fee number N :

Frequency constraint: N may occur at most k times.

Forbidden combination: N may *not* occur simultaneously with certain other fee numbers.

Restricted combination: If N occurs simultaneously with other fee numbers, it may occur *only* with certain other fee numbers.

Likewise, four different types of consequences of a rule violation arising when checking N can be observed:

Reject: N violates a hard constraint and is rejected.

Replace: N - and possibly some other fee numbers - must be replaced by another number.

Tariff: N might not be refunded depending on the customer's insurance tariff.

Note: A notification text regarding N is delivered to the person in charge who will decide on any further action.

Using these observations, each of ACMI's rule information can be expressed using the following three predicates, where N is the number to be checked, $k \geq 0$ is a number indicating a maximal frequency, \mathcal{M} is a set of fee numbers, $Cons \in \{reject, replace, tariff, note\}$ indicates the type of consequence of a violation, and R is a fee number being used in the output part of the rule (e.g. the number to be rejected or the number be used in a replacement):

frequency($N, k, Cons, R$): If N occurs more than k times, execute $Cons$ for N with number R .

forbidden($N, k, \mathcal{M}, Cons, R$): If N occurs and more than k fee numbers from \mathcal{M} occur, execute $Cons$ for N with number R .

restricted($N, \mathcal{M}, Cons, R$): If N occurs and another fee number occurs that is not in \mathcal{M} , execute $Cons$ for N with number R .

In all three predicates, "execute $Cons$ for N with number R " is defined by:

- If $Cons = reject$, mark N to be rejected.
- If $Cons = replace$, mark N to be rejected and mark R to be refunded instead.
- If $Cons = tariff$, mark N to be not covered by the insurance tariff unless the current insurance tariff covers the full GebüH.
- If $Cons = note$, mark N to be displayed with a notification to the person in charge.

Now we want to illustrate how the relevant knowledge for checking fee numbers can be represented with the three predicates *frequency*, *forbidden* and *restricted*. E.g., using these predicates, the information of the rules C1, C2, C3, C4, C5, T1, and H1 given in Sec. 3 can be expressed by:

- C1:** *frequency(0400, 1, reject, 0400)*
restricted(0400, {0100, 1701}, reject, 0400)
- C2:** *frequency(2008, 1, reject, 2008)*
- C3:** *frequency(2801, 1, replace, 2802)*
- C4:** *forbidden(1209, 1, {1210,1211}, reject, 1209)*
- C5:** *forbidden(0500, 0, {0600,0700,0800}, reject, 0500)*
- T1:** *frequency(2101, 0, tariff, 2101)*
- H1:** *frequency(1202, 0, note, 1202)*

For instance, *frequency(0400, 1, reject, 0400)* says that if number 0400 occurs more than once, 0400 should be rejected, and *frequency(2801, 1, correct, 2802)* requires that if 2801 occurs more than once, it should be replaced by 2802. In the schema for T1, *frequency(2101, 0, tariff, 2101)* states that 2101 might not be covered, depending on the current tariff.

Note the difference between the schemas for C4 and C5: 1209 may occur with (not more than) one of {1210, 1211}, but 0500 may not occur simultaneously with any of {0600, 0700, 0800}.

Since these rule schemas represent a higher level of abstraction compared to the modelling by DLV rules, they can not be expressed directly within the paradigm of ASP due to their usage of set-valued arguments. Thus, they can not be coded immediately in e.g. DLV (which does not provide data structures like lists or sets). Therefore, we developed a transformation from the general rule schemas to DLV code where each schema instance corresponds to a set of DLV facts and rules. Additionally to the DLV code generated from the schemas, there are rules ensuring that the constraints expressed by them are not violated. In the following, we will illustrate this transformation process from the general rule schemas to DLV code.

For each of the three rule schemas introduced above, one or more facts are generated. For each rule schema of the form *frequency(N, k, Cons, R)*, the DLV fact

`frequency_fact(N, k, Cons, R).`

is generated. For each *forbidden(N, k, M, Cons, R)* with $\mathcal{M} = \{M1, \dots, Mn\}$, n facts of the form

`forbidden_fact(N, k, Mi, Cons, R).`

are generated, and similarly for all *restricted* rule schemas.¹ For instance, for the rule schemas for C1, the following facts are generated:

¹ If there is more than one rule schema of the form *forbidden(N, -, -, -, -)* for a fee number N , a more complicated generation of facts along with a correspondingly extended form of checking the constraints expressed by these facts must be used [2].

```

frequency_fact(0400, 1, replace, 0400).      %
restricted_fact(0400, 0100, reject, 0400).   % C1
restricted_fact(0400, 1701, reject, 0400).   %

```

Checking the constraints expressed by the rule schemas is done by exploiting DLV's count facility. Violation of rules are indicated by making the *execute* predicate true. We illustrate this for the *restricted* schema:

```

execute(N, Cons, R) :-
    restricted_fact(N, _, Cons, R),
    countRestrictedPartners(N, I),
    countPartners(N, All),
    All > I.

countRestrictedPartners(N, I) :-
    p(_, N),
    #count{P: restrictedPartner(N, P)} = I.

restrictedPartner(N, Partner) :-
    p(_, N), p(_, Partner),
    restricted_fact(N, Partner, _, _).

countPartners(N, All) :-
    p(_, N),
    #count{P: p(_, P), P <> N} = All.

```

Execution of the consequences of a rule violation is ensured by rules like:

```

%----- replace: -----
-z(N) :- execute(N, replace, R).
r(R)   :- execute(N, replace, R).

% ----- tariff: -----
t(N) :- execute(N, tariff, R),
      not fullGebuh.

```

Similarly, the *frequency* and *forbidden* schemas are processed.

Note that also this alternative implementation of transforming the general rule schemas to DLV exploits DLV's default mechanism: E.g., the default rule for R1 as presented in Sec. 5.1 is also present here.

5.3 DLV rules vs. Rule Schemas

The expressiveness of DLV's input language, in particular its support of defaults and negation and its aggregate facility, made it possible to express all relevant regulations for checking medical invoices directly in DLV, using an obvious encoding of invoices,

tariff information, etc. The resulting system ACMI is operational and can be used by the insurance company's person in charge checking the invoices.

On the other hand, the direct modelling by DLV rules requires knowledge about DLV and about the basics of ASP. Additionally, any changes of tariffs, the relevant regulations concerning any fee numbers etc., force modifications of the DLV code. This is not the case when the knowledge base is modelled using the three rule schemas we have identified. Using appropriate input masks, instances of the *frequency*, *forbidden* and *restricted* schemas can be created and modified directly by the insurance experts who need not have knowledge about DLV. Additionally, the three rule schemas provide a powerful modelling tool: All rules and regulations concerning the GebüH [5] could be expressed straightforwardly. We have also checked that various regulations concerning the GOÄ [4] (containing many more fee numbers and rules) can be formulated directly using *frequency*, *forbidden* and *restricted*. We are currently planning to extend ACMI to cover the full GOÄ, and our expectation is that no extension to the three rule schemas will be needed.

6 Conclusion and Future Work

In this paper, we presented the knowledge-based system ACMI which provides support for checking medical invoices at health insurance companies. ACMI helps performing this task faster and more efficiently.

Using answer set programming for implementing ACMI's knowledge base, we were able to achieve a declarative realization of all rules, stemming either from the relevant scale of fees or from the insurance company's internal regulations. The declarative modelling of ACMI's knowledge base exploits DLV's support of defaults and its aggregate facility. As an alternative approach, we developed a modelling using only the three general and abstract rule schemas *frequency*, *forbidden* and *restricted*. These schemas use sets which are not available in DLV, but they can be translated automatically to DLV code.

Whereas up to now we have implemented a system covering the full GebüH [5], future work will include the testing of ACMI in a large insurance company and its extension to cover the full GOÄ [4].

References

1. P. A. Bonatti, N. Shahmehri, C. Duma, D. Olmedilla, W. Nejdl, M. Baldoni, C. Baroglio, A. Martelli, V. Patti, P. Coraggio, G. Antoniou, J. Peer, and N. E. Fuchs. Rule-based policy specification: State of the art and future work. Project Deliverable D1, Working Group I2, EU NoE REVERSE, Sept. 2004.
2. O. Dusso. Entscheidungssysteme für die Rechnungsprüfung in der Krankenversicherung mit logik-basierten Regelsprachen. Diplomarbeit, Fachbereich Informatik, FernUniversität Hagen, 2005. (to appear).
3. T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative problem solving using the DLV system. In J. Minker, editor, *Logic-Based Artificial Intelligence*, pages 79–103. Kluwer Academic Publishers, Dordrecht, 2000.

4. GOÄ – Gebührenordnung für Ärzte. www.e-bis.de/goae/defaultFrame.htm, 2004.
5. GebüH – Gebührenordnung für Heilpraktiker. www.znh.de/downloads/GebuehrenordnungHP.doc, 2002.
6. M. Gelfond and N. Leone. Logic programming and knowledge representation – the A-prolog perspective. *Artificial Intelligence*, 138:3–38, 2002.
7. I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Math. Artificial Intelligence*, 25(3-4):241–273, 1999.
8. R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.