Medical Treatment Conflict Resolving in Answer Set Programming

Forrest Sheng Bao
Department of Computer Science
Texas Tech University
Lubbock, Texas 79409
forrest.bao @ gmail.com

Zhizheng Zhang
School of Computer Science
and Engineering
Southeast University
Nanjing, China 210096
zzzhang.gm @ gmail.com

Yuanlin Zhang
Department of Computer Science
Texas Tech University
Lubbock, Texas 79409
y.zhang @ ttu.edu

Abstract

Medical treatment decision making is a good application of knowledge representation and reasoning. We are particularly interested in using them to resolve treatment conflicts, a complicated condition when two treatments cannot be given simultaneously to a patient suffering from multiple symptoms. The logic system is required to reason on cases with and without treatment conflicts. Thanks to the nonmonotonicity of Answer Set Programming (ASP), we give an elegant solution for resolving a medical treatment conflict on an example problem and show the importance of nonmonotonicity in medical reasoning.

Introduction

Making medical treatment decisions is very challenging and sophisticated. Physicians make mistakes sometimes due to negligence. Logic programming provides a natural way to represent medical knowledge into logic rules and reasoning on top of them. The declarative property of logic programming makes problem modeling effective. Therefore, logic programming can help physicians make medical treatment decisions. Compared with human reasoning, another advantage of logic programming here is that the knowledge used in reasoning can come from more than one experienced physicians.

One difficult situation in medical treatment decision making is treatment conflict when a patient has multiple symptoms. By treatment conflict, we mean the treatments to at least two symptoms have opposite effects on a patient. For example, *hemoptysis* (i.e., coughing up blood) can be relieved by *increasing* coagulation (i.e., blood clotting) whereas *hypertension* (i.e., high blood pressure) can be relieved by *decreasing* coagulation. It is very easy to decide a treatment for either one of the two symptoms. But when a patient suffers from both, it is impossible to relieve both by increased and decreased coagulation simultaneously. For such a case, a physician may prioritize symptoms and treat the one that is most life-threatening. In other words, treatments are prioritized according to priority ranking of symptoms. Additionally, when treatment conflict happens, the treatment to the most life-threatening symptom may be different from that when there is no treatment conflict.

Answer Set Programming (Gelfond 2008), a kind of logic programming, is a Prolog-style knowledge representation tool with powerful nonmonotonic reasoning ability. Many efficient ASP inference engines (i.e., solvers) have been developed and used widely. We are very interested in using ASP to resolve treatment conflicts in medical problems by using prioritized symptoms of patients provided by physicians. Exceptions happen frequently in most statements in the medical domain (e.g., allergies or incomplete information). ASP provides a good support for reasoning on defaults with exceptions.

An ASP program consists of rules in the form:

$$l_0 :- l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$$

where each l_i for $i \in [0..n]$ is a logic literal. The reasoning result of an ASP program is called an answer set. The rule means "l_0 is believed if $l_1, \ldots, l_m$ are believed, and there is no reason to believe $l_{m+1}, \ldots, l_n$." The not is called negation as failure, which brings us great convenience to express defaults with exceptions. For instance, to express the idea that "increase patient X's coagulation, if he/she is suffering from hemoptysis and there is no reason not to do so," we can write an ASP rule

```
should(increase, X) :-    hemoptysis(X),
                   not -should(increase, X).
```

Normally, if `hemoptysis(X)` is true, we will have `should(increase,X)`. This is the default case. The condition "there is no reason not to do so" is called an exception (e.g., increasing coagulation could worsen a more life-threatening symptom). When exception happens, `not -should(increase, X)` is false, and we will not have `should(increase,X)`, even though patient X is suffering from hemoptysis.

In the rest of the paper, we will use an example to demonstrate how ASP can be employed to model medical treatment decision making that involves treatment conflict, and how an ASP solver can resolve the conflict and generate the proper treatment. The example we will use is very simple. But in reality, the problem could be way more complex and we hope ASP can help physicians on those difficult cases.

Problem Modeling in ASP

The ASP program to resolve treatment conflict can be partitioned into two parts, the knowledge part and the patient part. The knowledge part contains general medical knowledge independent from the patient. The patient part is a collection of facts about the patient, including symptoms and priority ranking of symptoms. Treatments will be prioritized (by physicians or an ASP solver) according to ranking of symptoms.

Knowledge Part

The knowledge in our running example can be abstracted as follows.

1. A patient's coagulation should be increased (decreased), if he/she is suffering from hemoptysis (hypertension), unless it should not.
2. A patient will be prescribed medication A (B) and should take it, if his/her coagulation should be increased (decreased) unless hemoptysis (hypertension) is not a priority. "Unless" here expresses an exception.
3. Medications A and B cannot be taken at the same time.
4. If a patient's coagulation should be increased (decrease) while being prescribed medication B (A), half the dose.

Knowledge 1 can be represented as:

```
should(increase, X) :-    hemoptysis(X),
                 not -should(increase, X).
should(decrease, X) :-  hypertension(X),
                 not -should(decrease, X).
```

where X refers to a patient.

Knowledge 2 and 3 can be combined into:

```
takes(X, a) :-       should(increase, X),
                          not takes(X, b),
            not -priority(hemoptysis, X).

takes(X, b) :-       should(decrease, X),
                          not takes(X, a),
          not -priority(hypertension, X).
```

where `priority(hemoptysis, X)` can be interpreted as "the treatment to hemoptysis should be given (first)."

Knowledge 4 can be expressed as:

```
half_dose(X,a) :-   should(decrease, X),
                              takes(X, a).

half_dose(X,b) :-   should(increase, X),
                              takes(X, b).
```

Patient Part

We create a patient Tom, who has both symptoms and hemoptysis is the priority. These information can be explicitly specified as:

```
hemoptysis(tom).
hypertension(tom).                 % rule r1
priority(hemoptysis, tom).         % rule r2
-priority(hypertension, tom).      % rule r3
```

Case Analysis

We consider two cases here, without and with treatment conflict, respectively. The program defined above is solved by `lparse`, `smodels` and `mkatoms`. We set the solver to show only literals with predicates `takes`, `should` and `half_dose` below.

In the first case, we comment out rules $r1 - r3$ in patient part, such that there is no treatment conflict on Tom. A simple answer set is found:

```
takes(tom,a)
should(increase,tom)
```

In the second case, we first introduce treatment conflict by uncommenting `r1` only. Two answer sets are found:

```
half_dose(tom,a)
takes(tom,a)
should(increase,tom)
should(decrease,tom)
```

and

```
half_dose(tom,b)
takes(tom,b)
should(increase,tom)
should(decrease,tom)
```

The ASP solver gives one treatment in each answer set because it does not know which symptom is the priority for Tom. Hence, a treatment conflict happens. Please note that in both answer sets, the coagulation of Tom should be increased and decreased.

Now we further uncomment rules `r2` and `r3` to introduce symptom priority information. Then only the first answer set is left. It correctly tells that Tom should take medication A because his hemoptysis is the priority, but in half of normal dose. Therefore, the treatment conflict is resolved.

Comparing the results above, we can see that with the change of given information (the availability of prioritized symptoms), the answer set (i.e., believes) changes. This is nonmonotonicity.

The experimental results show that our approach can resolve treatment conflict as expected and can also reason correctly when there is no treatment conflict.

Discussions

In this paper, we exploit the nonmonotonicity of ASP to automate medical treatment conflict resolving. We will extent our method for problems with more than two symptoms in the future.

Acknowledgment

Zhizheng Zhang's contribution in this work was supported by Chinese NSF under grant 60803061. Yuanlin Zhang's contribution in this work was supported by NSF under grant IIS-1018031.

References

Gelfond, M. 2008. *Answer sets*. Handbook of Knowledge Representation. Elsevier. 285–316.