184.701 UE Deklaratives Problemlösen SS 2017

Exercise 2: Answer-Set Programming Basics

Uwe Egly, Hans Tompits, and Christoph Redl

Institut für Informationssysteme,
Arbeitsbereich Wissensbasierte Systeme 184/3,
Technische Universität Wien

1 Introduction

This exercise deals with problems concerning the assignment of teachers to classes. The task is to construct suitable *extended logic programs* (ELPs) for each problem such that the solutions of a given problem are determined by the answer sets of the respective program. The problem ought to be solved by means of the *guess-and-check* methodology in answer-set programming, and furthermore by using *default negation*, *disjunctions*, and *integrity constraints* as essential formalisation tools.

The evaluation of the constructed programs shall be carried out with the system DLV, version 2012-12-17 (`http://www.dlvsystem.com`). Please note that the **next exercise** will require you to use the Potassco system instead (see `https://potassco.org/`). Special features of these systems, which extend the standard answer-set programming language, such as *weak constraints*, *aggregate functions*, *weight constraints*, or *optimise statements* shall **not** be employed in your solutions to the problem described here—those features will be the subject of the next exercise.

We further want to recommend the usage of the IDE `SeaLion` for answer-set programming. It supports both DLV and Potassco including useful features such as syntax highlighting, visualisation of answer sets, and a debugging system. `SeaLion` is realised as a plugin for the `Eclipse` platform and is available from `http://www.sealion.at/` where standalone versions as well as instructions how to get `SeaLion` via Eclipse's update mechanism are offered. The `SeaLion` plugin `Kara` (also available in the standalone version and the update site) allows for visualising answer sets. We provide an example visualisation program (an answer-set program specifying how an answer set should be visualised) that is tailored towards the example in the exercise, available at TUWEL. If you want to visualise an answer set, open its context menu in `SeaLion`'s interpretation view, select "Visualisation" and choose the visualisation program available from TUWEL.

2 General Problem Description

In the beginning of each semester, school administrators must assemble an assignment of teachers to courses in different classes. This assignment is based upon the qualifications of teachers, the curriculum, the grades of classes, and various other information. As this means a great deal of work when done manually, the administrators would like to automatically generate such assignments from a given set of information about the school's staff and students.

In this exercise, the problem of assigning the teachers to specific courses in classes shall be modeled in terms of logic programs under the answer-set programming paradigm. The precise problem description is as follows:

- The curriculum specifies the teaching needs of each class:
  - All courses are explicitly given.
  - All the classes and their grades (ranging from 1 to 4) are given.
  - It is specified how many lessons of some course are taught in each grade per week.
  - For every teacher and every course, it is known whether he or she is qualified to teach the course.
- Each class has a *form teacher* who represents it and takes administrative duties for it, but is also required to teach it in some course.
- One can only be form teacher for at most one class.
- Every teacher must be assigned to teaching at least one class in some course.
- No teacher should be assigned in a way that requires more than 24 hours of preparation per week. It is assumed that for each lesson in higher grades (grades 3 and 4), 2 hours of preparation are needed, whereas only 1 hour of preparation per lesson is needed in lower grades (grades 1 and 2). E.g., a teacher doing 3 lessons of mathematics in some 1^{st}-grade class and 1 lesson of physics in a 4^{th}-grade class will have a total of $3 \cdot 1 + 1 \cdot 2 = 5$ hours of preparation.
- No teacher should be assigned to more than 4 classes.
- If a teacher is related to some student in a class he or she should not teach that class. The only exception to this is when there is a relative in every class he or she is qualified to teach. He or she then may teach a single class with a relative in it (in any number of courses).
- A teacher may be assigned to teaching a course only if he or she is qualified to teach it.
- The school offer extra-curricular activities like a debating society or a mathematics club. Each of those activities is led by a teacher and needs a certain amount of preparation, contributing to the hours spent for preparing lessons. That is, a teacher should not spend more than 24 hours preparing lessons and extra-curricular activities in total. Furthermore, a teacher should not spend more time preparing extra-curricular activities than lessons.

This problem shall be solved by means of two subtasks described as follows.

3 Subtask 1

3.1 Definition of the Check-Program

In the first part of the task, construct a program `check.dl` which checks, given an assignment of teachers to courses in the classes, whether the following conditions hold:

(i_1) Only courses from the given set (`course/1`) are used to classify assignments, curricula, and teachers' qualifications to teach courses.

(i_2) Classes can only be in grade 1 – 4 and not in more than one grade.

(i_3) If for some grade and some course no curriculum is given, 0 lessons are assumed to be required.

(a_1) If a class does not require any lessons in some course, no teacher should be assigned to teach the course in that class.

(a_2) Only teachers who are qualified for a course may teach it.

(a_3) Each class should be taught each course by at most one teacher.

(a_4) No teacher has to be without any teaching assignment.

(a_5) No teacher should teach more than 4 different classes.

(a_6) A teacher may not teach a class he or she has a relative in, except when there is a relative in every class, to which he or she could teach at least one course. In that case, he or she may teach a single class with a relative in it (and any number of courses in that class).

(a_7) No teacher should be required to spend more than 24 hours preparing lessons and extra-curricular activities in total every week, where each lesson for grades 1 and 2 accounts for 1 hour of preparation time and each lesson for grades 3 and 4 accounts for 2 hours of preparation time.

(f_1) A class must not have than one form teacher.

(f_2) A class must be taught by its form teacher.

(f_3) A teacher must not be form teacher at more than one class.

(e_1) An extra-curricular activity (`activity/2`) is led by exactly one teacher.

(e_2) No teacher should spend more time preparing extra-curricular activities than regular lessons.

The given assignment of teachers to classes (in terms of courses of those classes) is assumed to be stored in some input files (described below) containing the following data:

- the names of the teachers, their qualifications, and which classes they have relatives in;
- the names of the classes and their respective grades;
- the names of courses;
- the number of lessons required for each course in each grade;
- the assignment of teachers to the courses in classes and as form teachers;
- the extra-curricular activities including weekly preparation time;

- the assignment of teachers to extra-curricular activities.

The input has to be specified such that these conditions are met. Furthermore, you have to use the following predicates for the construction of the program `check.dl` as well as for the input data:

- `teacher(T)`: `T` is a teacher;
- `class(C,G)`: `C` is a class, `G` is its grade;
- `course(S)`: `S` is the name of a course;
- `qualified(T,S)`: teacher `T` is qualified for course `S`;
- `curriculum(G,S,L)`: in grade `G` the curriculum requires `L` weekly lessons in course `S`;
- `formteacherOf(T,C)`: `T` is the form teacher of class `C`;
- `hasRelativeIn(T,C)`: `T` is related to some student in class `C`;
- `assigned(T,C,S)`: teacher `T` is assigned to teach course `S` to class `C`;
- `activity(A,L)`: `A` is an extra-curricular activity offered by the school, which requires `L` hours preparation per week;
- `responsibleFor(T,A)`: teacher `T` is assigned to lead the extra-curricular activity `A`;

The program `check.dl` should satisfy the following condition:

- `check.dl`, together with the input data, possesses an answer set precisely when conditions (i_1)–(i_3), (a_1)–(a_7), (f_1)–(f_3), and (e_1)–(e_2) are met.

Important: Do not use any aggregate atoms for constructing the program `check.dl`!

3.2 Testing the Check-Program

Test the functionality of your program `check.dl` by constructing **five test cases**. To this end, for each test case n ($1 \leq n \leq 5$), put your data in the following files:

- `teachers`n`.facts`: contains the names of the teachers, i.e., facts of form

  `teacher(T)` and `hasRelativeIn(E,C)`,

  where `T` is a teacher and `C` is a class, and the qualification relations

  `qualified(T,S)`,

  where `T` is a teacher and `S` is a course.

- `classes`n`.facts`: contains the classes, i.e., facts of form

  `class(C,G)`,

  where `C` is a class and `G` is the grade, as well as the curriculum, i.e., facts of form

```
curriculum(G,S,L),
```

where `G` is a grade, `S` is a course, and `L` is the number of lessons per week;

- `courses`n`.facts`: contains the list of course names, i.e., facts of form

```
course(S),
```

where `S` is a course.

- `activities`n`.facts`: contains the list of extra-curricular activities, i.e., facts of form

```
activity(A,L),
```

where `A` is an extra-curricular activity and `L` the preparation time needed per week.

- `asgn`n`.facts`: contains the assignment of teachers `T` to courses `S` in classes `C`, i.e., facts of form

```
assigned(T,C,S).
```

In addition, it includes the information about the form teachers, given by

```
formteacherOf(T,C),
```

as well as the assignment of teachers `T` to extra-curricular activities `A` given by

```
responsibleFor(T,A).
```

For testing your program, note that it is beneficial to use both *positive tests* (corresponding to correct behaviour) and *negative tests* (violating the specification).

4 Subtask 2

4.1 Definition of the Guessing Program

Now construct a program `guess.dl` which, given a collection of teachers, courses and classes, assigns the teachers to courses in classes in such a way that the following conditions are satisfied:

(g_1) for each course requiring $L > 0$ lessons in the curriculum of the grade of a class, exactly one teacher is assigned to teach that course in that class;

(g_2) each class has exactly one form teacher, who is also teaching that class in some course;

(g_3) for each extra-curricular activity requiring $L > 0$ hours of preparation per week, exactly one teacher is assigned.

Use the above defined predicates

`teacher(T)`, `formteacherOf(T,C)`, `class(C,G)`, `course(S)`, `qualified(T,S)`, `curriculum(G,S,L)`, `assigned(T,C,S)`, `activity(A,L)`, and `responsibleFor(T,A)`.

The program `guess.dl` should be constructed in such a way that the following property is met:

- the answer sets of `guess.dl`, together with the input data, yield all assignments of teachers to the courses in classes which satisfy conditions (g_1)–(g_3).

4.2 Testing the Overall Program

Use your test files from subsection 3.2—*but without the files* `asgn`n`.facts` ($1 \leq n \leq 5$)—to test the overall program consisting of `guess.dl` and `check.dl`.

Important:

1. Be sure that you follow the naming of predicates and files as pointed out above.
2. All of the files mentioned above should be contained in **one directory**!
3. Do not include the command `#maxint` in your files—rather, execute DLV with the corresponding command-line option `-N` to specify the known numbers.
4. Do not use any weak constraints or aggregate atoms in your programs.

5 Submission

Upload a `zip` file with your solution to the TUWEL system. Before the deadline you can upload your solution several times. Submissions after the deadline will be rejected. Use the target

```
make zip
```

from `Makefile` to create your `zip` file. The result will be `XXXXXXX_ex2.zip`, where `XXXXXXX` is replaced by your matriculation number (make sure you have set the `MNR` variable in the `Makefile` to the correct value).

In case of questions, please post a message to the official TISS forums or send an email to

`dps-2017s@kr.tuwien.ac.at`.