Principles of Knowledge Representation and Reasoning

B. Nebel, M. Helmert, S. Wölfl
G. Röger
Summer Semester 2008

University of Freiburg
Department of Computer Science

Projekt P1

Due: July 11, 2008

Submit by email to `roeger@informatik.uni-freiburg.de`.

In this project, we implement some CSP-related algorithms.

> *Note:* The emphasis of the projects lies on the implementation, not so much on theoretical aspects. If you are not sure at some points how the algorithms and definitions from the lecture can be implemented in practise, feel free to ask questions.
>
> You may use C, C++, Java or Python for your implementation. If you want to use another language, please contact us before you start the project.
>
> The programs that you submit must conform to the required in- and output formats, pass several tests and be **commented** sufficiently. We will not accept programs that do not meet these demands but you may amend them until the due date. Hence, we ask for early submissions so that enough time remains for amendments.

Exercise P1.1 (Arbitrary binary CSPs, 2 marks)

(a) Write a generator that randomly creates binary constraint networks depending on the following parameters:

- **v** number of variables
- **w** number of values in the domains
- **c** number of constraints
- **d** density of the constraints, i.e. the number of actual entries divided by the number of possible entries

The parameters should be specified as command line arguments. Use the following example as guidance for your input and output (in particular, respect the sorting of constraints and set elements).

```
# random-network -v 4 -w 3 -c 4 -d 0.4
Variables:
    V = {v_0, v_1, v_2, v_3}
Domain:
    D = {V_0, V_1, V_2}
```

```
Constraints:
    R_0_1 = {(V_0,V_0), (V_1,V_0), (V_1,V_1), (V_2,V_0)}
    R_0_2 = {(V_0,V_0), (V_0,V_1), (V_1,V_1), (V_2,V_1)}
    R_0_3 = {(V_0,V_2), (V_1,V_0), (V_1,V_1), (V_2,V_1)}
    R_1_2 = {(V_1,V_1), (V_1,V_2), (V_2,V_1), (V_2,V_2)}
```

(b) Extend your program by algorithm **EnforcePathConsistency**. Besides the original CSP, also output the constraints of the equivalent path consistent CSP.

Example output:

```
# path-consistency -v 4 -w 3 -c 4 -d 0.4
Variables:
        V = {v_0, v_1, v_2, v_3}
Domain:
        D = {V_0, V_1, V_2}
Constraints:
        R_0_2 = {(V_0,V_0), (V_0,V_1), (V_1,V_0), (V_1,V_1)}
        R_0_3 = {(V_0,V_2), (V_1,V_0), (V_2,V_0), (V_2,V_2)}
        R_1_3 = {(V_0,V_1), (V_1,V_0), (V_1,V_2), (V_2,V_2)}
        R_2_3 = {(V_1,V_0), (V_1,V_1), (V_1,V_2), (V_2,V_0)}
Constraints of the path consistent network:
        R_0_0 = {(V_0,V_0), (V_1,V_1)}
        R_0_1 = {(V_0,V_1), (V_0,V_2), (V_1,V_1)}
        R_0_2 = {(V_0,V_1), (V_1,V_1)}
        R_0_3 = {(V_0,V_2), (V_1,V_0)}
        R_1_1 = {(V_1,V_1), (V_2,V_2)}
        R_1_2 = {(V_1,V_1), (V_2,V_1)}
        R_1_3 = {(V_1,V_0), (V_1,V_2), (V_2,V_2)}
        R_2_2 = {(V_1,V_1)}
        R_2_3 = {(V_1,V_0), (V_1,V_2)}
        R_3_3 = {(V_0,V_0), (V_2,V_2)}
```

Exercise P1.2 (Allen's interval calculus, 2 marks)

(a) Write an parser that reads CSPs (for Allen's interval calculus) that are given in the following form:

- The first line contains the number v of variables
- The next v lines contain the variables
- The next line contains the number c of constraints
- Each of the next c lines contains one constraint which is written as `<variable1>:<variable2>:(<base relations>)`

The input file on the left hand side should result in the output on the right hand side:

```
4
P1
P2
P3
P4
4
P2:P1:(d)
P2:P3:(< m)
P3:P1:(d)
P4:P2:(d f)
```

```
Variables:
    V = {P1, P2, P3, P4}
Constraints:
    R_P1_P1 = {=}
    R_P1_P2 = {di}
    R_P1_P3 = {di}
    R_P1_P4 = {<, =, >, d, di, f, fi, m, mi,
               o, oi, s, si}
    R_P2_P1 = {d}
    R_P2_P2 = {=}
    R_P2_P3 = {<, m}
    R_P2_P4 = {di, fi}
    R_P3_P1 = {d}
    R_P3_P2 = {>, mi}
    R_P3_P3 = {=}
    R_P3_P4 = {<, =, >, d, di, f, fi, m, mi,
               o, oi, s, si}
    R_P4_P1 = {<, =, >, d, di, f, fi, m, mi,
               o, oi, s, si}
    R_P4_P2 = {d, f}
    R_P4_P3 = {<, =, >, d, di, f, fi, m, mi,
               o, oi, s, si}
    R_P4_P4 = {=}
```

(b) Implement algorithm **EnforcePathConsistency2**. On the website you can find the composition table in a format that is easy to parse. Again, print the resulting constraints, e.g.

```
Constraints:
    R_P1_P1 = {=}
    R_P1_P2 = {di}
    R_P1_P3 = {di}
    R_P1_P4 = {di}
    R_P2_P1 = {d}
    R_P2_P2 = {=}
    R_P2_P3 = {<, m}
    R_P2_P4 = {di, fi}
    R_P3_P1 = {d}
    R_P3_P2 = {>, mi}
    R_P3_P3 = {=}
    R_P3_P4 = {>, mi}
    R_P4_P1 = {d}
    R_P4_P2 = {d, f}
    R_P4_P3 = {<, m}
    R_P4_P4 = {=}
```

Exercise P1.3 (Comparison, 1 mark)

Write a translator that translates CSPs from exercise 2 to the style required in exercise 1 by instantiating the symbolic constraints for the domain $\{(x, y) \mid x, y \in \{1, \ldots, n\}\}$, where n is given as command line argument. Use your implementation in exercise 1 to make the resulting constraint network path consistent.

Furthermore, use your implementation from exercise 2 to make the (symbolic) source CSP path consistent and instantiate the resulting constraints with the domain from the previous part.

Compare the results for different n (example input CSPs will be published on the website).