Constraint Programming
Prof. Dr. Thom Frühwirth, Marc Meister

assignment #12 (winter term 2005)
solutions will be presented Tuesday, 7-Feb-2006, 2 PM, o27/2203
http://www.informatik.uni-ulm.de/pm/index.php?id=112

**Constraint System $\Re$**

**Exercise 1** ($\Re$ as special RT solver)**.**
Modify the solver for rational trees (RTs)

```
idempotency   @ X eq Y \ X eq Y  <=> true.
reflexivity   @ X eq X           <=> true.
orientation   @ T eq X           <=> var(X), X@<T | X eq T.

decompostion  @ T1 eq T2         <=> nonvar(T1), nonvar(T2) | ...

confrontation @ X eq T1, X eq T2 <=> var(X), X@<T1, T1@=<T2 | X eq T1, T1 eq T2.
```

by only changing the body of the `decomposition` rule to handle linear polynomial equations: The resulting equation `T1` minus `T2` is computed in normal form. An equation is in normal form if `X eq A + B*Y + ...` for variables, `X, Y, ...` and numbers `A, B, .....`
Discuss completeness and termination. Test your implementation.

**Exercise 2.**
Implement an optimized solver for $\Re$ as discussed in the lecture: Based on the $\Re$ solver available at WebCHR `http://bach.informatik.uni-ulm.de/~webchr/` (note: actual rules are at the very bottom of the files), that use slack variables to turn inequations into equations use Gaussian-style variable elimination for equations, use Fourier variable elimination and the bridge rule combining the variable elimination approaches for equations and inequations containing only slack variables.

**Constraint System *FD***

**Exercise 3** (Extension of *FD*-solver)**.**
Extend the contraints solver `u12-fd.pl`.

a) Implement rules to let the `leq`, `eq`, and `neq` constraints interact with the given solver. Tests should include `X leq Y, Y leq Z, X in [0,3], Y in [-1,2], Z in [0,1,2,3]`.
b) Introduce the `maximum(X,Y,Z)` constraint to the given solver, where `Z` is the maximum of `X` and `Y`. Use the rules for addition from the lecture as sample and propagate into the direction of `Z` only. Tests should include `maximum(X,Y,Z), X in [0,1], Y in [2,4], Z in [3,5]`.
c) The constraint `allneq(List)` succeeds, iff the variables in `List` are (pairwise) distinct. Test your implementation `allneq` with three variables, s.t. the answer is `no` for one and `yes` for a second query.
d) Implement a `label(List)`-constraint to bind the variables in `List`. An easy test case is, e.g., `X in [0,1], Y in [0,1], X leq Y, label([X])`.

---

**Exercise 4** (ACM World Finals 2004 – Problem C)**.**
Implement a CHR solver to handle the ACM World Finals problem given on the next page. You are free to model the data representation, and need not care about input/output details. Enjoy!

The 2004 28th Annual acm International Collegiate
Programming Contest World Finals
sponsored by IBM

# Problem C
## Image Is Everything
### Input File: image.in

Your new company is building a robot that can hold small lightweight objects. The robot will have the intelligence to determine if an object is light enough to hold. It does this by taking pictures of the object from the 6 cardinal directions, and then inferring an upper limit on the object's weight based on those images. You must write a program to do that for the robot.

You can assume that each object is formed from an $N \times N \times N$ lattice of cubes, some of which may be missing. Each $1 \times 1 \times 1$ cube weighs 1 gram, and each cube is painted a single solid color. The object is not necessarily connected.

## Input

The input for this problem consists of several test cases representing different objects. Every case begins with a line containing $N$, which is the size of the object ($1 \leq N \leq 10$). The next $N$ lines are the different $N \times N$ views of the object, in the order front, left, back, right, top, bottom. Each view will be separated by a single space from the view that follows it. The bottom edge of the top view corresponds to the top edge of the front view. Similarly, the top edge of the bottom view corresponds to the bottom edge of the front view. In each view, colors are represented by single, unique capital letters, while a period (.) indicates that the object can be seen through at that location.

Input for the last test case is followed by a line consisting of the number 0.

## Output

For each test case, print a line containing the maximum possible weight of the object, using the format shown below.

| Sample Input | Output for the Sample Input |
|---|---|
| 3<br>.R. YYR .Y. RYY .Y. .R.<br>GRB YGR BYG RBY GYB GRB<br>.R. YRR .Y. RRY .R. .Y.<br>2<br>ZZ ZZ ZZ ZZ ZZ ZZ<br>ZZ ZZ ZZ ZZ ZZ ZZ<br>0 | Maximum weight: 11 gram(s)<br>Maximum weight: 8 gram(s) |