CHR programming contest 2010

Jon Sneyers and Peter Van Weert

31 August 2010, Leuven

Rules

There are 6 problems. You have to submit the solution to each problem as fast as possible by bringing us an USB stick with the solution program (the filename is in the title of each problem). You can submit as many solution attempts per problem as you want; only the first (if any) correct solution is taken into account.

Teams have to consist of **two** or **three** persons and there may be at most two teams of two persons. Each team can use only **one** computer with **one** keyboard. Also, each team only gets **one** copy of the questions. We know these constraints are annoying. We are slightly evil.

Each correct solution earns you one point. The team with the most points is the winner. In case of ties, the total time for correct solutions is counted and the fastest teams wins. For example, if you the contest starts at time t, and your team submitted three correct solutions, at times $t + 30$, $t + 70$ and $t + 90$, then your total time is $30 + 70 + 90 = 190$.

For problem 5 ("Complexity classes") you can earn a bonus point. The bonus point counts as 0.5 "real" points and gives a discount of 30 minutes on your total time; also, only the submission time of the "main" problem solution is recorded, in case you first submit a solution that does not solve the bonus part, and later submit an improved version that does the bonus part.

Your solutions have to be written in CHR(Prolog). They will be tested in a recent version of SWI-Prolog. You are allowed to use Prolog code (in particular, disjunction and search), but your solution may not be written exclusively in Prolog without CHR.

In questions that require you to print some output, you have to use Prolog builtins like `write`/1 and `nl`/0 to print the output. After the output is printed, your program may fail or succeed (we don't care).

We don't care about any auxiliary constraints that may be left in the constraint store at the end of the program execution; we only look at the correctness of the "output" constraints.

Run-time efficiency is not important, but if your program does not terminate in a reasonable time (where "reasonable" may depend on our patience), the solution is considered to be incorrect.

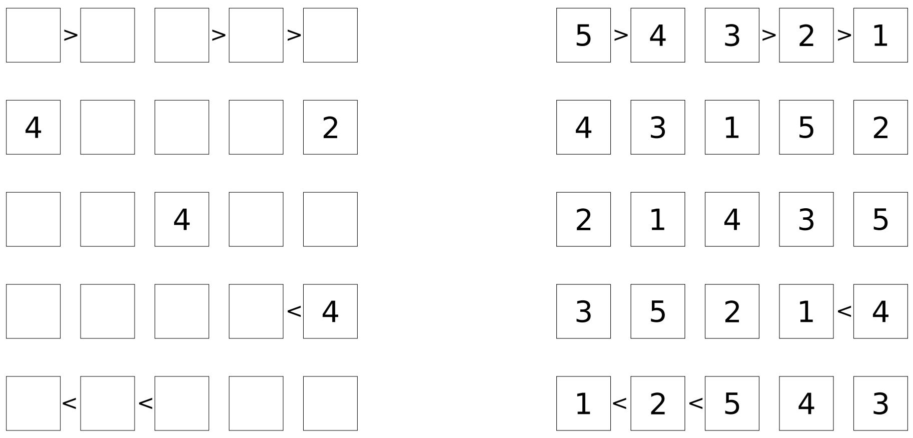
Figure 1: A Futoshiki puzzle (left) and its solution (right).

1 Futoshiki puzzles (`futoshiki.chr`)

Futoshiki is a Japanese puzzle game, played on an $n \times n$ grid. Figure 1 gives an example. The goal is to fill the grid with the numbers $1 \dots n$ such that:

- Every row has all n different numbers
- Every column has all n different numbers
- The given inequalities are satisfied

The input is one `futo`/1 constraint where the argument is a list of rows of values. Every row of values is a list, where variables represent unknown values and numbers represent given values. The inequalities are represented as `gt`/2 constraints, where `gt(A,B)` means that `A > B`. The input ends with one `solve`/0 constraint.

As output you simply have to assign values to all variables in such a way that the puzzle is solved. Valid puzzles have a unique solution.

Hint: Use `findall(s(X,Y,E),(nth1(Y,Input,Row),nth1(X,Row,E)),L)` to transform the input matrix `Input` to a list `L` of `s(X,Y,E)` terms, where `(X,Y)` is the horizontal and vertical position and `E` is the variable or number.

Example (the example of Figure 1):

```
?- futo([[ A1, A2, A3, A4, A5 ],
        [ 4 , B2, B3, B4, 2  ],
        [ C1, C2, 4 , C4, C5 ],
        [ D1, D2, D3, D4, 4  ],
        [ E1, E2, E3, E4, E5 ]]),
   gt(A1,A2), gt(A3,A4), gt(A4,A5), gt(4,D4), gt(E3,E2), gt(E2,E1), solve.
A1 = 5,
A2 = 4,
A3 = 3,
A4 = 2,
A5 = 1,
...
```

2 Manhattan Prime Knights (`mpk.chr`)

Perhaps you have heard about the *n*-queens puzzle. Or perhaps you have not. It does not matter.

The *Manhattan Prime Knight* (MPK) is a very special chess piece. It can move to any position (or take any piece there) which is at a Manhattan distance which is a prime number. The Manhattan distance between two squares on a chess board is the sum of the horizontal and vertical distance. So if square A is at position `(Ax,Ay)` and square B is at position `(Bx,By)`, then the Manhattan distance between A and B is `abs(Ax-Bx)+abs(Ay-By)`. Figure 2 illustrates the moves of an MPK.

Given a generalized chess board of width `W` and height `H`, your task is to print out all possible ways to put `N` MPKs on the board such that no MPK can capture any other MPK. When printing out the board layouts you print "`.`" for empty squares and "`K`" for squares with a MPK on them.

The input is one constraints `board(W,H,N)`. The output is a printout of all board layouts that satisfy the above conditions, separated by empty lines. It does not matter in which order you print them, but you have to print all different solutions, each solution only once, and of course no invalid solutions.

Here are some examples:

```
?- board(5,5,5).
K...K
.....
..K..
.....
K...K
```

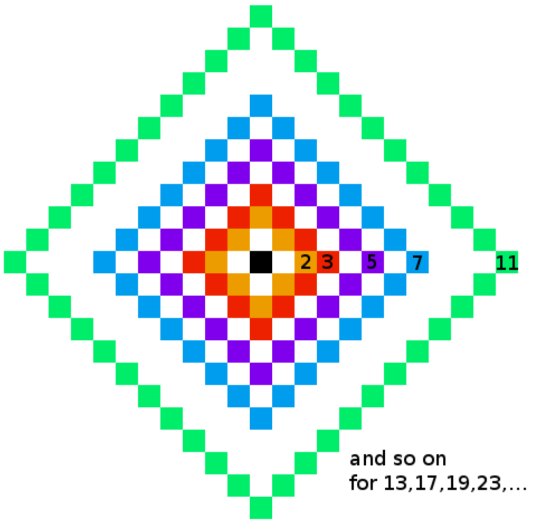

Figure 2: Possible MPK moves from the black square in the middle.

```
?- board(5,1,2).
KK...

K...K

.KK..

..KK.

...KK

?- board(9,4,6).
K.....K..
...K.....
........K
.K...K...

.K...K...
........K
...K.....
K.....K..

..K.....K
.....K...
K........
...K...K.

...K...K.
K........
.....K...
..K.....K

?- board(9,3,5).
K...K...K
.........
..K...K..

..K...K..
.........
K...K...K

?- board(13,1,4).
KK.......KK..

K...K...K...K

.KK.......KK.

..KK.......KK
```

3 Robot bombing (`robot.chr`)

There are a lot of robots trying to take over the world. Obviously you have to save humanity by destroying them. You can do this by dropping bombs. Some robots are bigger than others, but all robots are round (cylinder-shaped; from the air they look like disks). Their engine is in the center, and it is warm. Your bombs are guided by heat, so you can aim them only at the center of robots.

When a robot is destroyed, it explodes. When a robot explodes, the explosion can possibly destroy other robots. The strength of the explosion depends on the kind of robot: some robots are more explosive than others. The more explosive they are, the larger the radius of the explosion. Every robot in that radius gets destroyed, and it suffices that some part of it is within the radius, not necessarily their center. When a robot of radius `R` and explosive power `E` explodes, it destroys everything in a radius of `R+E` around its center. For example, if there is a robot at `(0,0)` and a robot at `(3,0)`, and they each have radius `1` and explosive power `1`, then if one of them explodes, the explosion just barely reaches the other one and destroys it too.

Your task is to destroy all robots using as few bombs as possible, since bombs are expensive and obviously you have to save humanity in a cost-effective way.

As input you get a list of robots as `robot(X,Y,R,E)` constraints, where `X,Y` is the position of the center of the robot, `R` is the radius of the robot, and `E` is the explosive power of the robot, followed by one `save_the_world`/0 constraint.

As output you have to give `bomb`/2 constraints with the coordinates you are bombing. The total number of bombs should be minimal. If there are multiple minimal solutions you can give any of them.

Some examples (the first one is illustrated in Figure 3):

```
?- robot(4,7,1,1), robot(5,2,1,1), robot(7,5,1,2),
   robot(11,3,2,1), robot(15,5,2,1), save_the_world.
bomb(7,5)

?- robot(4,7,1,1), robot(5,2,1,1), robot(7,5,1,1),
   robot(11,3,2,1), robot(15,5,2,1), save_the_world.
bomb(11,3)
bomb(7,5)
bomb(5,2)
bomb(4,7)
```

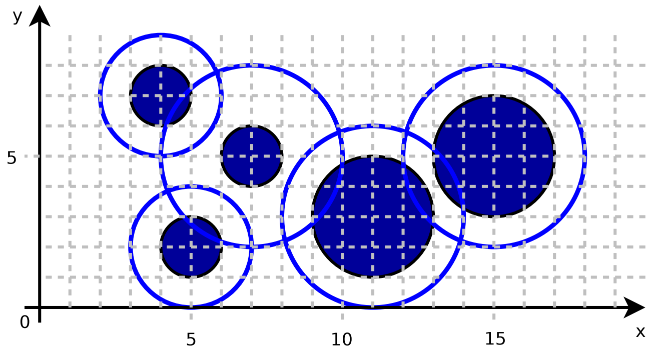

Figure 3: Five robots.

4 Hobo cigarettes (`hobo.chr`)

A certain hobo can make one cigarette out of four cigarette butts (the butt is what is left after smoking a cigarette). If he finds some cigarettes and some cigarette butts, how many cigarettes can he smoke in total?

The input is any number of constraints of this form: `cigarette`/0, `butt`/0, and `pack`/1. Each `cigarette` represents one cigarette, and each `butt` represents one cigarette butt. The constraint `pack(N)` represents a pack containing `N` cigarettes.

The output is one constraint of the form `smoked(T)`, where `T` is the total number of cigarettes the hobo has smoked. There can be no cigarettes left (the hobo smokes every cigarette he finds or makes), but there can be (will be) cigarette butts left (always less than four).

Examples:

```
?- pack(4).
butt
smoked(5)

?- pack(25).
butt
smoked(33)

?- butt, pack(3).
butt
smoked(4)
```

5 Complexity classes (`complexity.chr`)

In August 2010, Vinay Deolalikar made a (failed) proof attempt to show that P $\neq$ NP. We know that P $\subseteq$ NP, but we still do not know if it also holds that NP $\subseteq$ P (so P = NP) or P $\neq$ NP. We also know that PSPACE $\subseteq$ EXPTIME and NP $\subseteq$ PH and NL $\subseteq$ P and PH $\subseteq$ PSPACE etcetera etcetera. Your task is to help a complexity theorist to keep track of all those complexity classes.

As input, you get some `set/2` constraints, where the first argument is a Prolog variable and the second argument is an atom which represents the name of the set. For example `set(A,'P'), set(B,'NP')`. You also get some `subset/2` constraints, for example `subset(A,B)`. The input query is terminated by a `show`/0 constraint.

As output you have to print the relationships between the sets as one long chain of subset inclusions and equalities, if possible, using the names of the set. If it is not possible to do this, you have to print the message “`Error: not a single chain.`”.

Here are some examples:

```
?- set(P,'P'),set(N,'NP'),set(S,'PSPACE'),subset(P,N),subset(N,S),show.
P subset_of NP subset_of PSPACE

?- set(P,'P'),set(N,'NP'),set(S,'PSPACE'),subset(P,N),subset(N,S),subset(N,P),show.
NP=P subset_of PSPACE

?- set(NL,'NL'),set(P,'P'),set(N,'NP'),set(S,'PSPACE'),set(E,'EXPTIME'),
   subset(NL,P),subset(P,N),subset(N,S),subset(S,EXP),subset(E,N),show.
NL subset_of P subset_of PSPACE=EXPTIME=NP

?- set(P,'P'),set(N,'NP'),set(C,'NPC'),subset(P,N),subset(C,N),show.
Error: not a single chain.
```

BONUS POINT!

You get one bonus point if you also print the inclusions as an ASCII-art Venn diagram, in the following way:

```
?- set(P,'P'),set(N,'NP'),set(S,'PSPACE'),subset(P,N),subset(N,S),show.
P subset_of NP subset_of PSPACE


     ________
    /        \
   / ________ \
  / /        \ \
 / /  ______  \ \
( (  (______)  ) )
 \ \    P     / /
  \ \________/ /
   \    NP    /
    \________/
      PSPACE
```

```
?- set(P,'P'),set(N,'NP'),set(S,'PSPACE'),subset(P,N),subset(N,S),subset(N,P),show.
NP=P subset_of PSPACE

   ______
  /      \
 /  ____  \
(  (____)  )
 \  NP=P  /
  \______/
   PSPACE

?- set(NL,'NL'),set(P,'P'),set(N,'NP'),set(S,'PSPACE'),set(E,'EXPTIME'),
   subset(NL,P),subset(P,N),subset(N,S),subset(S,E),subset(E,N),show.
NL subset_of P subset_of PSPACE=EXPTIME=NP

     ________
    /        \
   / ________ \
  / /        \ \
 / /  ______  \ \
( (  (______)  ) )
 \ \    NL    / /
  \ \________/ /
   \    P     /
    \________/
    PSPACE=EXPTIME=NP

?- set(L,'L'),set(NL,'NL'),set(P,'P'),set(N,'NP'),set(S,'PSPACE'),set(E,'EXPTIME'),
   subset(L,NL),subset(NL,P),subset(P,N),subset(N,S),subset(S,E),show.
L subset_of NL subset_of P subset_of NP subset_of PSPACE subset_of EXPTIME

           ______________
          /              \
         / ______________ \
        / /              \ \
       / / ______________ \ \
      / / /              \ \ \
     / / / ______________ \ \ \
    / / / /              \ \ \ \
   / / / / ______________ \ \ \ \
  / / / / /              \ \ \ \ \
 / / / / /  ____________  \ \ \ \ \
( ( ( ( (  (____________)  ) ) ) ) )
 \ \ \ \ \       L        / / / / /
  \ \ \ \ \______________/ / / / /
   \ \ \ \       NL       / / / /
    \ \ \ \______________/ / / /
     \ \ \       P        / / /
      \ \ \______________/ / /
       \ \       NP       / /
        \ \______________/ /
         \     PSPACE     /
          \______________/
              EXPTIME
```

6 Soccer preference conflicts (`soccer.chr`)

A local sport club is organizing an old-fashioned dance party where dancing is done in (male,female)-pairs. However, since many of the participants are fervent soccer supporters (including even some dangerous hooligans), they are worried about potential fighting incidents, which would of course spoil the party fun.

In particular, they want to avoid pairing up persons with conflicting soccer team preferences. For example, if Jef prefers Ajax over Liverpool, while Marie prefers Liverpool over Ajax, then the organizers do not want Jef and Marie to dance together, to avoid any potential violence. Of course preferences are transitive: if somebody prefers Milan over Madrid, and Madrid over Ajax, then obviously she implicitly prefers Milan over Ajax.

Most of the participants are rational, in the sense that their soccer team preferences are consistent (i.e. their preferences correspond to a strict partial order). However, some of the participants are hooligans. Hooligans have irrational preferences: they can for example simultaneously think that A is better than B and that B is better than A. For obvious safety reasons, irrational participants should not be allowed to dance with anyone, and in fact, they are not even allowed at the party.

As input you get the participant list as `male`/1 and `female`/1 constraints (the argument being the name of the person). Their soccer team preferences are given as `prefers`/3 constraints, where `prefers(Name,Team1,Team2)` means that `Name` likes the team `Team1` more than the team `Team2`.

As output you have to give all possible 'peaceful' pairs as `pair`/2 constraints, where `pair(Name1,Name2)` means that `Name1` is a male participant and `Name2` is a female participant and they have no conflicting preferences. Also, the bouncers want to know the names of all hooligans; these names are to be outputted as `irrational`/1 constraints.

Example input:

```
male(jean), male(louis), male(jef),
  prefers(jean,madrid,milan), prefers(jean,liverpool,ajax),
  prefers(louis,madrid,ajax), prefers(louis,ajax,liverpool),
  prefers(jef,ajax,madrid), prefers(jef,madrid,milan), prefers(jef,milan,ajax),
female(marie), female(anna), female(sophie),
  prefers(marie,milan,liverpool), prefers(marie,liverpool,madrid),
  prefers(anna,ajax,milan),
  prefers(sophie,ajax,liverpool).
```

Example output:

```
pair(louis,sophie)
pair(jean,anna)
pair(louis,anna)
irrational(jef)
```