
Constraint Programming

Prof. Dr. Thom Frühwirth

Amira Zaki

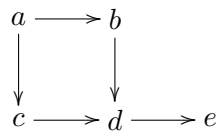
Winter Term 2011

Assignment #1

Exercise 1. Make yourself familiar with SWI Prolog

- Write a “Hello world!” program in Prolog, e.g.,
`start :- write('Hello world!')`.
Compile the source and call it with `start`.
- Create the family database adding facts, representing the family tree as follows:
 - (1) `male(X)` : X is a male
 - (2) `female(X)` : X is a female
 - (3) `parent(X,Y)` : X is a parent of Y
- Implement rules to represent the following relationships:
 - (1) `mother(X,Y)` : X is the mother of Y
 - (2) `father(X,Y)` : X is the father of Y
 - (3) `grandmother(X,Y)` : X is the grandmother of Y
 - (4) `ancestor(X,Y)` : X is an ancestor of Y
 - (5) `siblings(X,Y)` : X and Y are siblings (both parents in common)

Exercise 2. Model the given directed graph into respective facts and implement a predicate `path(X,Y)` that holds if there exists a path between the two points X and Y.



Exercise 3. Write the following arithmetic Prolog predicates:

- (1) `minimum(A,B,C)` which succeeds if C is the minimum of A and B.
- (2) `factorial(X,N)` which succeeds if N is the factorial of X.

Exercise 4. Write the following list manipulating Prolog predicates:

- (1) `member(X,L)` which succeeds if X is a member of the list L.
- (2) `len(N,L)` which succeeds if N is the length of a list L.
- (3) `append(L1, L2, L3)` which succeeds if L3 is the concatenation of L1 and L2.
- (4) `reverse(L1,L2)` which succeeds if the list L2 is the reverse of the list L1.
- (5) `remove(X,L1, L2)` which succeeds if L2 is the result of removing X from L1.
- (6) `union(L1, L2, L3)` which succeeds if L3 is the union of L1 and L2.
- (7) `intersect(L1, L2, L3)` which succeeds if L3 is the intersection of L1 and L2.

The next exercise to be submitted by e-mail to: amira.g.zaki@gmail.com. The deadline is on 09.11.2011 by 10:00. You are allowed to work in a group of two people. Please send only one e-mail per group, containing the solution and both team member names.

Exercise 5. Implement a Prolog program to help scheduling routes for the freight forwarding business. The program uses the following knowledge base, giving distances between major cities in Germany.

Hamburg, Bremen, 80	Hamburg, Hannover, 110
Hamburg, Berlin, 230	Bremen, Hannover, 100
Bremen, Dortmund, 200	Hannover, Kassel, 140
Hannover, Nuernberg, 380	Dortmund, Kassel, 130
Dortmund, Koeln, 80	Kassel, Wuerzburg, 180
Kassel, Frankfurt, 180	Frankfurt, Wuerzburg, 110
Nuernberg, Muenchen, 160	

- Represent the distances with a Prolog predicate `dist(city0,city1,km)`.
Test your implementation with queries like `?- dist(dortmund,bremen,Km)`,
`?- dist(From,To,180)`, and `?- dist(From,To,Km)`.
- The Prolog predicate `route(From,To)` should succeed, iff there is a route between `From` and `To`.
Tests should include `route(muenchen,berlin)`, `route(ulm,halle)`, and finally `route(frankfurt,To)`.
- Define a ternary predicate `route(From,To,Route)` which has the same functionality as `route/2` w.r.t. the first and second parameters `From` and `To`. Additionally, the third parameter, `Route`, contains the cities already visited in order to avoid cycles (i.e. visiting a city more than once).
Thorough testing should include ground and non-ground queries, e.g.,
`?- route(dortmund,X,[kassel])`.
- The Prolog predicate `shortestpath(From,To,Route,Km)` should succeed, iff the minimal distance between the cities `From` and `To` is `Km` along `Route`.
Tests should include, e.g., `shortestpath(kassel,bremen,[hannover],X)`.

Mind the upper/lower-case notation and the closing period in Prolog.
--

Constraint Programming

Prof. Dr. Thom Frühwirth

Amira Zaki

Winter Term 2011

Assignment #2

For quick help on how to use SWI-Prolog:

<http://www.cs.bris.ac.uk/Teaching/Resources/COMS30106/labs/SWI-gettingstarted.html>

The SWI-prolog documentation of CLP-FD library:

<http://www.swi-prolog.org/man/clpfd.html>

Exercise 1 (CLP Cryptoarithmic Puzzle).

Replace distinct letters by distinct digits (numbers have no leading zeros), such that the following calculation holds (a literal translation to English is “Test thoroughly your strengths”):

$$\begin{array}{rcccccc} & & & & & & & T & E & S & T & E \\ + & & & & & & & F & E & S & T & E \\ + & & & & & & & D & E & I & N & E \\ \hline = & K & R & A & F & T & E & & & & & \end{array}$$

Use the `clpfd` library to model the previous puzzle as done in the lecture. You might find it useful to use the constraint `all_different(+Vars)`.

Exercise 2. Model the following dinner problem as a constraint problem in `clpfd`:

We are going out to dinner taking 1-6 grandparents, 1-10 parents and 1-40 children. Grandparents cost 3 euros for dinner, parents cost 2 euros and children 0.50 euros. There must be 20 total people at dinner and it must cost exactly 20 euros. The problem to be solved is to find how many grandparents, parents and children are going to dinner.

Exercise 3. Can you find the ages according to the following dialogue?

Alex: How old are you mama?

Mama: Our three ages add up to exactly seventy years.

Alex: And how old are you papa?

Papa: Just six times as old as you, my son.

Alex: Shall I ever be half as old as you, papa?

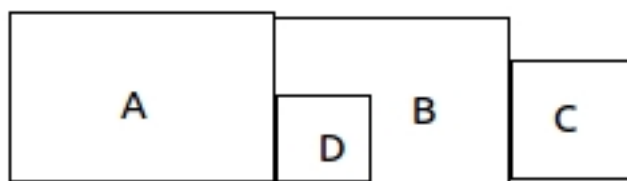
Papa: Yes Alex; and when that happens our three ages will add up to exactly twice as much as today.

Write a `clpfd` program to solve this puzzle and return the age of Alex, Mama and Papa today.
Hint: model the age in months.

Exercise 4. The map coloring problem tries to find a way to color a map such that no two boarding states have the same color. The predicate `color(States, NumColors, Adj)` should succeed iff it is possible using a maximum number of colors, `NumColors`, and the list of adjacent pair of states, `Adj`, to satisfy the map coloring rules and color the states as in `States`.

Implement the predicate `color` using `clpfd`. Test coloring of the 4 states shown below using 3 colors, with the predicate:

```
?- color([A,B,C,D],3,[(A,B),(B,C),(B,D),(D,A)]).
```



Exercise 5 (Magic Square).

A “magic square” is a rectangular array of numbers, usually from 1 to n^2 , such that each column, row, and both diagonals have the same sum. The constant sum in every row, column and diagonal is called the magic sum, M . Using the `clpfd` library, solve the puzzle for a magic square with $n = 4$ and $M = 34$. Make sure you write the full square on the console.

The next exercise is to be submitted by e-mail to: amira.zaki@uni-ulm.de. The deadline is on 16.11.2011 by 10:00. You are allowed to work in a group of two people. Please send only one e-mail per group, containing the solution and both team member names.

Exercise 6 (Survo Puzzle).

Survo puzzle is a kind of logic puzzle presented (in April 2006) and studied by Seppo Mustonen. The name of the puzzle is associated to Mustonen’s Survo system which is a general environment for statistical computing and related areas.

In a Survo puzzle the task is to fill an $m \times n$ table by integers $1, 2, \dots, m \times n$ so that each of these numbers appears only once and their row and column sums are equal to integers given on the bottom and the right side of the table. Often some of the integers are given readily in the table in order to guarantee uniqueness of the solution and/or for making the task easier [taken from Wikipedia].

Here is a simple Survo puzzle with 3 rows and 4 columns:

	A	B	C	D	
1		6			30
2	8				18
3			3		30
	27	16	10	25	

Solve the Survo puzzle using the `clpfd` library for 3×4 tables.

Constraint Programming

Prof. Dr. Thom Frühwirth

Amira Zaki

Winter Term 2011

Assignment #3

Write and query Prolog, CLP and CHR online using:

<http://chr.informatik.uni-ulm.de/~webchr/>

CCLP

We use (a subset of) the Constraint Handling Rules (CHRs) to program in the CCLP paradigm.

The following CCLP-clause $H \leftarrow C \mid G$

is written as $N @ H \Leftarrow C \mid G$

where $N @$ is an optional name for the rule.

- (1) Read the SWI-Prolog manual on how to use CHRs:

<http://www.swi-prolog.org/man/chr.html>

- (2) Before using CHR rules, the CHR library must be included by `:- use_module(library(chr)).`
- (3) User-defined CCLP predicates must be declared, e.g. `constraints con/1.`
- (4) To enforce that guards should be checked for (illegal) variable bindings, use:
`:- chr_option(check_guard_bindings,on).`

Exercise 1 (Comparison of CLP and CCLP).

Compare the following CLP- (in the left column) and CCLP-programs (in the right column), which consist of *one* of the given rules by posing the queries given below. Check your answers with the system's answers. Make sure, you understand why seemingly innocuous rules produce different answers.

`p(a) :- true.`

`p(X) :- X=a.`

`p(X) :- X = a, X = b.`

`p1 @ p(a) <=> true | true.`

`p2 @ p(X) <=> X = a | true.`

`p3 @ p(X) <=> true | X = a.`

`p4 @ p(X) <=> X = a | true.`

`p5 @ p(X) <=> X = a, X = b | true.`

`p6 @ p(X) <=> X = a | X = b.`

`p7 @ p(X) <=> true | X = a, X = b.`

Queries: (a) `p(a)`, (b) `p(b)`, and (c) `p(C)`.

Exercise 2. Implement the following three variants of the CCLP program `min` (minimum) in CHR:

- Variant 1:

$min_{1_1}(X, Y, Z) \leftarrow \top \mid X \leq Y, Z = X$

$min_{1_2}(X, Y, Z) \leftarrow \top \mid Y \leq X, Z = Y$

- Variant 2:

$min_{2_1}(X, Y, Z) \leftarrow X \leq Y \mid Z = X$

$min_{2_2}(X, Y, Z) \leftarrow Y \leq X \mid Z = Y$

- Variant 3:

$min_{3_1}(X, Y, Z) \leftarrow X \leq Y, Z = X \mid \top$

$min_{3_2}(X, Y, Z) \leftarrow Y \leq X, Z = Y \mid \top$

Test and explain the different responses of the variants by posing the following six queries (only one at a time):

`min(1,2,C).` `min(A,2,1).` `min(A,2,3).`

`min(A,A,B).` `min(1,2,1).` `min(1,2,3).`

Exercise 3 (Hamming's problem).

Consider the classical *Hamming's problem*, which is to compute an ordered ascending chain of all numbers whose only prime factors are 2, 3, or 5. The chain starts with the numbers:

$$1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24, 25, \dots$$

Define a non-terminating process `hamming(N)` that will produce the numbers as elements of the infinite chain starting with 1. *Hints:*

- pretend that the sequence is already known
- no base cases in recursion as sequences are infinite
- concurrent-process network, processes can be executed in parallel
- to view the stream, you will need an `observe/1` rule that writes elements of the stream as they are produced

Exercise 4 (Bank Transactions).

A classical and good example of a producer/consumer interaction is a bank account model.

The bank "consumes" client's commands which are in the form:

`withdraw(Amount)` - withdraws the `Amount` from the client's balance

`deposit(Amount)` - deposits the `Amount` to the client's balance

`balance(Amount)` - binds `Amount` to the client's balance

where `Amount` is a free variable.

For example, a client could produce the following:

$$[\text{deposit}(100), \text{withdraw}(25), \text{balance}(75)]$$

The bank process observes these commands and in its second argument modifies the current balance accordingly. Implement CHR `bank/2` rules that handle a stream of the three possible client commands and update the balance accordingly. Also implement a starter rule `bank/1` which initially sets the balance to zero, then calls the `bank/2` handlers.

You can use the Prolog predicate `read/1` to read the next Prolog term from the current input stream. Thus additionally, implement a producer/client (`client/1`) that would continuously read a term from the user.

The idea is to create these consumer and producer "threads", to run them using the query:

```
?- bank(L), client(L).
```

The next exercises are to be submitted by e-mail to: amira.zaki@uni-ulm.de. The deadline is on 30.11.2011 by 10:00. You are allowed to work in a group of two people. Please send only one e-mail per group, containing the solution and both team member names.

Exercise 5 (Fibonacci Sequence).

By definition, the first two numbers in the Fibonacci sequence are 0 and 1, and each subsequent number is the sum of the previous two. The chain starts with the numbers:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \dots$$

Define a non-terminating process `fib(N)` that will produce the numbers as elements of the infinite chain starting with 0. *Hint:* You want to create an infinite stream, so do not create a base case and you can provide the first two Fibonacci elements in the query, along with the `observe/1`.

Exercise 6 (Minimum of Stream). Create a recursive stream that produces minimal elements found so far in a stream (i.e. a sequence of current minima) using `min/3`. An example query:

```
?- observe(ML), ML=[31|_], minlist([31,56,13,45,24,52,2,634,58,632,23,1543],ML).
```

would produce the stream: 31,31,31,13,13,13,13,2,2,2,2,2,2.

Constraint Programming

Prof. Dr. Thom Frühwirth

Amira Zaki

Winter Term 2011

Assignment #4

For debugging, you can use the tracer. To enable the tracer, enter “chr_trace”. To disable it again, use “chr_notrace”. The tracer will show you the computation step by step.

Exercise 1. Compare the following CHR programs, which consist of *one* of the given rules by posing the given queries. Check your answers with the system’s answers. Make sure, you understand why seemingly innocuous rules produce different answers.

c1 @ c(X), c(X) <=> q(X,X).	Queries:
c2 @ c(X), c(Y) <=> r(X,Y).	a) c(X), c(X)
c3 @ c(X), c(X) ==> q(X,X).	b) c(X), c(Y)
c4 @ c(X), c(Y) ==> r(X,Y).	c) c(X), c(Y), X=Y

More variants:

q1 @ p(X,Z), q(Z,Y) <=> q(X,Y).	
q2 @ q(Z,Y), p(X,Z) <=> q(X,Y).	
q3 @ p(X,Z), q(Z,Y) ==> q(X,Y).	Queries:
q4 @ q(Z,Y), p(X,Z) ==> q(X,Y).	d) p(a,b), q(b,c)
q5 @ p(X,Z) \ q(Z,Y) <=> q(X,Y).	e) p(a,b), q(b,c), p(d,a)
q6 @ q(Z,Y) \ p(X,Z) <=> q(X,Y).	

Comment on the system’s answers for queries a) to e). Comment on the system’s answers for the rule q5 and the following two queries:

p(X,C), p(Y,C), q(C,A) and p(Y,C), p(X,C), q(C,A).

Exercise 2. Implement the constraints `less/2` (encoding $<$) und `leq/2` (encoding \leq) and their mutual relations/interactions in CHR. You may find the lecture’s CHR program for the \leq constraint helpful. For an example query, take your last name as a sequence of variables with \leq constraints between succeeding characters. The name *Fruehwirth* translates to the query: `leq(F,R), leq(R,U), leq(U,E), leq(E,H), leq(H,W), leq(W,I), leq(I,R), leq(R,T), leq(T,H)` with answer `leq(F,E), R = T, U = T, E = T, H = T, W = T, I = T`. Tests should include (at least) three more queries consisting of combined `less` and `leq` constraints.

The next exercise is to be submitted by e-mail to: amira.zaki@uni-ulm.de. The deadline is on 07.12.2011 by 10:00. You are allowed to work in a group of two people. Please send only one e-mail per group, containing the solution and both team member names.

Exercise 3. The addition of natural numbers written in successor notation (the natural number 3 is written as `s(s(s(0)))`) can be implemented by:

```
add(0, X, X).
add(s(X), Y, s(Z)) :- add(X, Y, Z).
```

Write a constraint solver in CHR, that solves as many as possible single `add`-constraints, e.g., the first Prolog clause implies:

```
add(0, Y, Z) <=> Y=Z.
add(X, Y, Y) <=> X=0.
```

You will need to write the other base cases and the recursive cases required for the addition.

Constraint Programming

Prof. Dr. Thom Frühwirth, Amira Zaki

Winter Term 2011

Assignment #5

Exercise 1 (Warmup – generate and test).

Implement a Prolog-predicate `permutation/2` that generates permutations: `permutation(A,B)` generates a permutation `B` of a list `A` with fixed length. All permutations can be computed by backtracking using “;” when prompted. Do not use built-in predicates for manipulating lists. Describe your observations with the following queries:

- `permutation([1,2,3],[L,M,N])`.
- `permutation([L,M,N],[1,2,3])`.
- `permutation([1,2,3],B)`.
- `permutation(A,[1,2,3])`.

Exercise 2 (Permutation Sort – “generate and test”).

Use the *permutation sort* algorithm for sorting a list of integers. To implement the *generate and test version*, use three Prolog predicates `permsort(List,Sorted)`, `permutation(List,Sorted)`, and `sorted(Sorted)` (all arguments are lists).

Exercise 3 (Permutation Sort – “constrain and generate”). Use the CHR Constraint `leq/2` from assignment #4-2 for a “constrain-and-generate” version of the *permutation sort* algorithm, replacing the Constraint `=<` by the CHR-constraint `leq`.

Your tests should (at least) include the following queries

```
?- permsortCHR([1,A,3],[1,3,7]).
?- permsortCHR([2,A],X).
?- permsortCHR([A,B,A],X).
?- permsortCHR(List,[1,X,3]).
?- permsortCHR([1,X,Y],[X,1,Y]), permsortCHR([4,5,10],[Z,Y,W]).
```

The next exercise is to be submitted by e-mail to: amira.zaki@uni-ulm.de. The deadline is on 14.12.2011 by 10:00. You are allowed to work in a group of two people. Please send only one e-mail per group, containing the solution and both team member names.

Exercise 4. Write a CHR program for the `maximum(X,Y,Z)`-constraint, which succeeds iff `Z` is the maximum of `X` and `Y`. Use your implementation of the `leq/2`- and `less/2`-constraints from assignment #4-2. Consider the following items in this order:

- (1) Write a CHR rule, which computes the maximum `Z` of two given numbers `X` and `Y`.
- (2) Enhance by inserting a CHR rule, such that queries like `?- maximum(X,X,3)` can be handled satisfactorily (we expect `X = 3`).
- (3) Insert a CHR rule which, given the constraint `maximum(X,Y,Z)`, propagates the constraints `leq(X,Z)` and `leq(Y,Z)`. Test with the query `?- maximum(A,B,C),maximum(C,A,B)`.
- (4) Insert CHR rules, such that under a given inequality between `X` and `Y` (e.g. `X leq Y`) the constraint `maximum(X,Y,Z)` is replaced by the implied equality (e.g. `Y = Z`).
- (5) Insert a CHR rule, such that for a query like `?- less(X,Z), maximum(X,Y,Z)` the answer `Y=Z` is returned.
- (6) In order to handle queries like `?- maximum(X,Y,3),maximum(X,Y,5)` in a satisfactory manner, extend the program for a rule for this class of queries.
- (7) Which are the sufficient conditions, given that two variables are unequal, to allow more inferences from `maximum(X,Y,Z)`.
Implement this case by (additional) rules in your CHR program.

Constraint Programming

Prof. Dr. Thom Frühwirth

Amira Zaki

Winter Term 2011

Assignment #6

Constraint System B

Download `boole.pl` from the lecture web page: http://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.170/home/betz/boole.pl. It contains the implementation of the constraints `neg/2`, `and/3`, `or/3`, `xor/3`, and `imp/2` of the Boolean Algebra. Use this constraint-solver for the following exercises.

Exercise 1 (Equivalence).

Extend `boole.pl` with rules (similar to the ones already defined) in order to introduce equivalence, i.e. implement simplifications for a CHR-constraint `equiv(X,Y,Z)` which obey the given truth table.

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

Exercise 2 (Who is lying?).

Lehmann says Mueller lies.
Mueller says Schulze does not tell the truth.
Schulze says both lie.

Write a Prolog-predicate `tellTruth(Lehmann,Mueller,Schulze)` which determines who of the three people is lying and who is telling the truth; it should succeed iff the three arguments are a valid interpretation of the given statements by Lehmann, Mueller, and Schulze. Use the Boolean constraints `and`, `neg`, ...

Hint: Lehmann's statement can be modeled by `Lehmann = MuellerLies`, or using equivalence, with `MuellerLies` being the negation of `Mueller`.

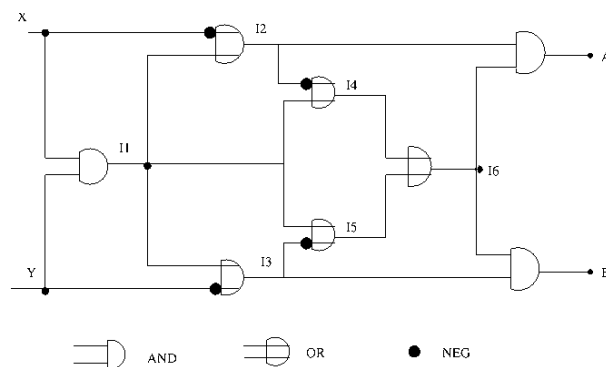
The next exercise is to be submitted by e-mail to: amira.zaki@uni-ulm.de. The deadline is on 11.01.2012 by 10:00. You are allowed to work in a group of two people. Please send only one e-mail per group, containing the solution and both team member names.

Exercise 3 (Cross Circuit).

A cross circuit exchanges two wires/signals with the help of a logic circuit without crossing them physically. For the input pins (X , Y) and the output pins (A , B) we have $A = Y$ and $B = X$.

Write a CHR constraint `cross(X,Y,A,B)`, which implements a cross circuit by means of Boolean constraints.

Test with queries `cross(1,0,A,B)`, `cross(1,Y,1,B)` and `cross(0,Y,A,B)`.



Constraint Programming

Prof. Dr. Thom Frühwirth

Amira Zaki

Winter Term 2011

Assignment #7

Exercise 1 (Fault Analysis).

Use `boole.pl` (from assignment #6) and write a CHR constraint

```
faultanalysis(X,Y,Z,O1,O2,F1,F2,F3,F4,F5)
```

that describes the fault analysis of a full adder:

$\neg F_1 \rightarrow (X \sqcap Y = I_1)$ And gate G_1

$\neg F_2 \rightarrow (X \oplus Y = I_2)$ Xor gate G_2

$\neg F_3 \rightarrow (I_2 \sqcap Z = I_3)$ And gate G_3

$\neg F_4 \rightarrow (I_2 \oplus Z = O_1)$ Xor gate G_4

$\neg F_5 \rightarrow (I_1 \sqcup I_3 = O_2)$ Or gate G_5

Under the hypothesis of minimal conflict we assume that at most one gate in a faulty circuit is faulty:

$$\begin{aligned} & \neg(F_1 \sqcap F_2) \wedge \neg(F_1 \sqcap F_3) \wedge \neg(F_1 \sqcap F_4) \wedge \neg(F_1 \sqcap F_5) \wedge \neg(F_2 \sqcap F_3) \wedge \\ & \neg(F_2 \sqcap F_4) \wedge \neg(F_2 \sqcap F_5) \wedge \neg(F_3 \sqcap F_4) \wedge \neg(F_3 \sqcap F_5) \wedge \neg(F_4 \sqcap F_5) \end{aligned}$$

Note that for completeness the constraint `faultanalysis` must end with a labeling/enumeration predicate for search. Try at least the following constraints:

```
faultanalysis(0,0,1,0,1,F1,F2,F3,F4,F5)
```

```
faultanalysis(1,1,0,0,0,F1,F2,F3,F4,F5)
```

Exercise 2 (Cardinality Constraints).

Extend `boole.pl` to handle cardinality constraints `card/4` with the semantics given in the lecture. Implement the rules together with the required auxiliary predicates. You can test it with calls that bind some variables to 0 or 1 like:

```
card(2,3,[X,Y,Z,W],4), Y=1, W=0
```

The next exercises are to be submitted by e-mail to: amira.zaki@uni-ulm.de. The deadline is on 18.01.2012 by 10:00. You are allowed to work in a group of two people. Please send only one e-mail per group, containing the solution and both team member names.

Exercise 3 (De Morgan's laws).

Translate the rewrite rules for De Morgan's law of propositional logic into CHR simplification rules:

```
not(and(X,Y)) -> or(not(X),not(Y)).
```

```
not(or(X,Y)) -> and(not(X),not(Y)).
```

Exercise 4 (To like or not to like).

I know three people: Pat, Quincy and Ray. I like at least one of them. If I like Pat but not Ray then I also like Quincy. I like both Ray and Quincy or none of them. Describe the facts as Boolean constraints and determine who I like for sure and who I may dislike.

Constraint Programming

Prof. Dr. Thom Frühwirth, Amira Zaki

Winter Term 2011
Assignment #8

Constraint System *FD*

Exercise 1 (Extension of *FD*-solver).

Extend the finite domain constraint solver over enumeration domains presented in the lecture.

- 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]`.
- Introduce the `maximum(X,Y,Z)` constraint to the given solver, where `Z` is the maximum of `X` and `Y`. Tests should include `maximum(X,Y,Z)`, `X in [0,1]`, `Y in [2,4]`, `Z in [3,4]`.
- 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 2 (Pyramid – Crypto-arithmetic Puzzle).

Solve the following crypto-arithmetic puzzle using finite domain constraints. Replace distinct letters by distinct digits, such that each number is the absolute difference of the two numbers below (e.g. $D = |G - H|$), and the numbers are the positive integers from 1 to 6.

$$\begin{array}{ccccc} & & A & & \\ & & B & & C \\ & D & & E & & F \end{array}$$

Hint: You will need to implement CHR rules for an absolute difference `abs/3` constraint, only consider propagation where one of the input values is fixed/known. Additionally implement a constraint `allneq(List)` that succeeds iff the variables in `List` are (pairwise) distinct.

The next exercise is to be submitted by e-mail to: amira.zaki@uni-ulm.de. The deadline is on 25.01.2012 by 10:00. You are allowed to work in a group of two people. Please send only one e-mail per group, containing the solution and both team member names.

Exercise 3 (Zebra Puzzle).

Five men with different nationalities live in the first five houses of a street. They practice five distinct professions, and each of them has a favorite animal and a favorite drink, all of them different. The five houses are painted in different colors.

The Englishman lives in the red house.

The Spaniard owns a dog.

The Japanese is a painter.

The Italian drinks tea.

The Norwegian lives in the first house on the left.

The owner of the green house drinks coffee.

The green house is next to the white and on the right of it.

The sculptor breeds snails.

The diplomat lives in the yellow house.

Milk is drunk in the middle house.

The Norwegian's house is next to the blue one.

The violinist drinks fruit juice.

The fox is in a house next to that of the doctor.

The horse is in a house next to that of the diplomat.

Question: Who owns a Zebra, and who drinks water?

Constraint Programming

Prof. Dr. Thom Frühwirth

Amira Zaki

Winter Term 2011

Assignment #9

Constraint System *Rational Trees*

Exercise 1. Implement the axioms of the theory of rational trees for the CHR equality constraint $X \text{ eq } Y$.

Hints:

- Change terms to list of functor and arguments by: $f(X_1, \dots, X_N) = \dots [f, X_1, \dots, X_N]$
- Rules leading to immediate contradiction should go first in the program text
- For termination reasons remove duplicate constraints

Queries:

- (a) $p(a, X, c, Y, Z) \text{ eq } p(Y, X, c, a, W)$
gives $Y = a, Z = W$
- (b) $p(Y, g(Y, f(Y)), g(X, a)) \text{ eq } p(f(U), V, g(h(U, V, W), U))$
gives $Y = f(a), X = h(a, g(f(a), f(f(a))), W), U = a, V = g(f(a), f(f(a)))$
- (c) $p(h(f(X), a), X, h(f(f(X)), f(f(X)))) \text{ eq } p(h(V, a), U, h(W, f(V)))$
gives $X = U, V = f(U), W = f(f(U))$
- (d) $p(g(X), h(a, X), h(Y, Y)) \text{ eq } p(U, h(a, g(V)), h(V, g(U)))$
gives false

Extend your implementation of rational trees to include the occur-check axiom of Clark's Equality Theory (CET). This means that infinite trees are now disallowed and should fail, an example of such a failing query: $X \text{ eq } f(Y), Y \text{ eq } f(X)$.

The next exercise is to be submitted by e-mail to: amira.zaki@uni-ulm.de. The deadline is on 01.02.2012 by 10:00. You are allowed to work in a group of two people. Please send only one e-mail per group, containing the solution and both team member names.

Exercise 2. The theory of rational trees should define the (purely) syntactic inequality $\dot{\neq}$ between two terms :

<i>irreflexivity</i>	$\forall(x \dot{\neq} x \rightarrow \perp)$
<i>symmetry</i>	$\forall(x \dot{\neq} y \rightarrow y \dot{\neq} x)$
<i>compatibility</i>	$\forall(x_1 \dot{\neq} y_1 \vee \dots \vee x_n \dot{\neq} y_n \rightarrow f(x_1, \dots, x_n) \dot{\neq} f(y_1, \dots, y_n))$
<i>decomposition</i>	$\forall(f(x_1, \dots, x_n) \dot{\neq} f(y_1, \dots, y_n) \rightarrow x_1 \dot{\neq} y_1 \vee \dots \vee x_n \dot{\neq} y_n)$
<i>distinctness</i>	$\forall(\top \rightarrow f(x_1, \dots, x_n) \dot{\neq} g(y_1, \dots, y_m)) \text{ if } f \neq g \text{ or } n \neq m$

Implement a CHR-constraint $X \text{ neq } Y$ that succeeds iff $CT \models X \dot{\neq} Y$. The implementation should follow the one for syntactic equality. A satisfaction-complete implementation is not necessary. Implement the negated `same_args` by a CHR constraint `one_neq/2` that uses disjunction. The two arguments are lists of the same length and the constraint should succeed iff at least one pair of list-elements is unequal.

Queries:

- (a) $X \text{ neq } f(X)$
(b) $f(a, X) \text{ neq } f(X, Y)$
(c) $f(g(X), a) \text{ neq } f(Y, X)$

Constraint Programming

Prof. Dr. Thom Frühwirth, Amira Zaki

Winter Term 2011
Assignment #10

Constraint System *Description Logic*

Download the constraint solver DL sent to your emails and make yourself familiar with the implementation:

- Concept/Role definitions are implemented as built-in constraints.
- `::` is used both for concept and role membership (by abuse of notation).

Exercise 1 (Warmup – Family Relationships).

Add the following T-Box to the constraint solver:

```
male isa nota female.
parent isa human and some child is human.
mother isa parent and female.
proud_parent isa parent and every child is phd.
grandmother isa mother and some child is parent.
motherofsons isa mother and every child is male.
```

Test with the following goals, try to unfold them and explain the answers:

```
sue::proud_parent, (sue,joe)::child, joe:nota phd.
sue::mother and nota grandmother, (sue,joe)::child.
sue::mother and nota grandmother, (sue,joe)::child, labeling.
```

Exercise 2 (Moving furniture). Consider the following scenario.

- Furniture are goods.
 - Vehicles can use traffic routes.
 - Transporters are vehicles that can transport goods.
 - Automobiles are vehicles that are driven by a motor and that use roads (only).
 - Trucks are automobiles that can transport goods.
 - Trains are vehicles that use rails (only).
 - Freight trains are trains that can transport goods.
 - Furniture trucks are trucks to transport furniture (only).
 - Bulli is a furniture truck.
 - Bulli transports G112 and G235.
 - G112 and G235 are goods.
 - Z521 is a train.
 - Z521 transports bananas and coals.
- (a) Identify the primitive and the compound concepts. Identify the roles. Separate T-Box from A-Box knowledge.
 - (b) Add the T-Box as Prolog facts to the constraint solver. Test the T-Box with queries like `X::transporter`.
 - (c) Add the A-Box as CHR-constraints. Explain the answer to the goal `z521::train` (use labeling.). Adapt (if necessary) your T-Box, such that `z521` does not use the road!
 - (d) Explain the answers when unfolding the following concept terms. Use labeling.
 - (1) Train and not a transporter.
 - (2) Freight train and not a vehicle.

The next exercise is to be submitted by e-mail to: amira.zaki@uni-ulm.de. The deadline is on 08.02.2012 by 10:00. You are allowed to work in a group of two people. Please send only one e-mail per group, containing the solution and both team member names.

Exercise 3. Consider the following animal taxonomy:

- Mammals are animals.
 - Lions are mammals that are carnivores.
 - Sheeps are mammals that are herbivores.
 - Cows are mammals that are herbivores.
 - Carnivores eat other animals.
 - Herbivores do not eat other animals.
 - A vertebrate is any animal that has, amongst other things, a backbone.
 - A bird is a vertebrate that has wings and legs.
 - A mad cow is a cow who feeds on sheep.
- (a) Identify the primitive and the compound concepts. Indentify the roles.
- (b) Add the T-Box as Prolog facts to the constraint solver.
- (c) Test with `x:madcow`. Is the solver complete?
- (d) Give an A-Box which shows if there is such a thing as a mad cow (ie. is the definition for a mad cow consistent?). Explain the unfolding.
- (e) Give an A-Box which shows the different parts of a bird. Explain the unfolding.
- (f) Change the last T-box to be become: “*A mad cow is a cow who eats sheep amongst other things.*” Repeat parts *c* and *d* to explain the difference.

Constraint Programming

Prof. Dr. Thom Frühwirth

Amira Zaki

Winter Term 2011

Assignment #11

Constraint System \mathcal{R}

Exercise 1 (\mathcal{R} as special RT solver).

Modify the solver for rational trees (RTs) of Assignment 9:

idempotency $@ X \text{ eq } Y \setminus X \text{ eq } Y \iff \text{true.}$

reflexivity $@ X \text{ eq } X \iff \text{true.}$

orientation $@ T \text{ eq } X \iff \text{var}(X), X @< T \mid X \text{ eq } T.$

decomposition $@ T1 \text{ eq } T2 \iff \text{nonvar}(T1), \text{nonvar}(T2) \mid \dots$

confrontation $@ X \text{ eq } T1 \setminus X \text{ eq } T2 \iff \text{var}(X), X @< T1, T1 @=< T2 \mid T1 \text{ eq } T2.$

as a polynomial equation solver 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 $X \text{ eq } A + B*Y + \dots$ for variables, X, Y, \dots and numbers A, B, \dots .

You can test your solver with queries like:

$5+3*X \text{ eq } 5+3*X.$

$Y \text{ eq } 7+2*X+1*Z, Y \text{ eq } -2+3*X, X \text{ eq } 3.$

$Y \text{ eq } 7+1*Z+2*X, Y \text{ eq } -2+3*X, X \text{ eq } 3.$

Exercise 2 (Inequations Solver using Fourier Elimination).

Implement an inequations solver for the `geq/2` constraint as discussed in the lecture: use Gaussian variable elimination for equations, use Fourier variable elimination and the bridge rule combining the variable elimination approaches for equations and inequations.

You can test your solver with queries like:

$0+1*Y+1*X \text{ geq } 0, 0+1*Y+(-1)*X \text{ geq } 0.$

$3+2*X+4*Y \text{ geq } 0, 12+4*X+2*Y \text{ eq } 0.$