Constraint Programming

Prof. Dr. Thom Frühwirth
Daniel Gall

Winter Term 2013
Assignment #1

Logic Refresher Exercises

Exercise 1. Check for each of the following formulae which properties are true: validity, satisfiability, falsifiability, unsatisfiability.

(1) $A \vee \neg A$
(2) $A \wedge \neg A$
(3) $A \rightarrow \neg A$
(4) $A \rightarrow (B \rightarrow A)$
(5) $A \rightarrow (A \rightarrow B)$
(6) $A \leftrightarrow \neg A$
(7) $(A \wedge B) \rightarrow (A \vee B)$
(8) $(A \vee B) \rightarrow A$
(9) $(A \wedge B) \wedge \neg A$
(10) $\neg(p(a) \rightarrow \exists X.p(X))$
(11) $(\exists X.p(X)) \rightarrow p(a)$

Exercise 2. For a formula F, are the following statements true or false? Give a counter example for each false statement.

(1) If F is valid, then F is satisfiable.
(2) If F is satisfiable, then $\neg F$ is unsatisfiable.
(3) If F is valid, then $\neg F$ is unsatisfiable.
(4) If F is unsatisfiable, then $\neg F$ is valid.

Exercise 3. Is G a logical consequence of F? For each state if $F \models G$ or $F \not\models G$.

F	G	$F \models G$ or $F \not\models G$
A	$A \vee B$	
A	$A \wedge B$	
A, B	$A \vee B$	
A, B	$A \wedge B$	
$A \wedge B$	A	
$A \vee B$	A	
$A, (A \rightarrow B)$	B	

Exercise 4. For each pair of atomic formulae, give both a most general unifier and a unifier which is not most general. Otherwise show that a unifier cannot exist.

(1) $p(a, X, c, Y, Z)$ and $p(Y, X, c, a, W)$
(2) $p(Y, g(Y, f(Y)), g(X, a))$ and $p(f(U), V, g(h(U, V, W), U))$
(3) $p(g(X), h(a, X), h(Y, Y))$ and $p(U, h(a, g(V)), h(V, g(U)))$
(4) $p(h(f(X), a), X, h(f(f(X)), f(f(X))))$ and $p(h(V, a), U, h(W, f(V)))$

Constraint Programming

Prof. Dr. Thom Frühwirth
Daniel Gall

Winter Term 2013
Assignment #3a

Constraint Logic Programming

The SWI-prolog documentation of CLP-FD library:
`http://www.swi-prolog.org/man/clpfd.html`

Exercise 1 (Cryptoarithmetic 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"):

		T	E	S	T	E
+		F	E	S	T	E
+		D	E	I	N	E
=	K	R	Ä	F	T	E

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 (Dinner)**.** 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 (Sums revisited)**.** In this exercise, we modify the `sum(X,Y,Z)` predicate from the last assignment. *Use only pure Prolog for this exercise, no constraints!*

(1) The sum should only accept natural numbers between 1 and 10 as addends.
   - For given values for `X`, `Y` and `Z`, your predicate just checks if the addends are allowed and if the sum is correct.
   - For given values for `X` and `Y`, it should calculate the sum `Z` of `X` and `Y`.

(2) Modify your implementation such that it can handle queries where one of the arguments is unbound.

(3) Extend your implementation such that it is capable of handling a query where the sum is given and the arguments might be unbound variables.
Hint: Generate two natural numbers between 1 and 10 and check their sum.

(4) How could the search tree be pruned?
Hint: Consider a query `sum(X,Y,10)`. After a value for `X` has been generated, `Y` can be calculated directly, without generating and testing.

Exercise 4 (Factorial revisited)**.** Constraint Logic Programming can be used as a more declarative alternative for ordinary integer arithmetic with `is/2` or `</2`. Modify your Prolog implementation of the factorial from the last assignment and replace every occurence of an arithmetic predicate by a constraint expression. Test the queries `factorial(6,F)`, `factorial(N,720)` and `factorial(N,721)` for the pure Prolog and the `clpfd` implementation.

Constraint Programming

Prof. Dr. Thom Frühwirth
Daniel Gall

Winter Term 2013
Assignment #3b

Constraint Logic Programming

The SWI-prolog documentation of CLP-FD library:
`http://www.swi-prolog.org/man/clpfd.html`

Exercise 1. 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 2 (Babysitting)**.** Each weekday, Bonnie takes care of five of the neighbours' children. The children's names are Keith, Libby, Margo, Nora, and Otto; last names are Fell, Gant, Hall, Ivey, and Jule. Each is a different number of years old, from two to six. Can you find each child's full name and age?

(1) One child is named Libby Jule.
(2) Keith is one year older than the Ivey child, who is one year older than Nora.
(3) The Fell child is three years older than Margo.
(4) Otto is twice as many years old as the Hall child.

Source: http://brownbuffalo.sourceforge.net/BabysittingClues.html

Constraint Programming

Prof. Dr. Thom Frühwirth
Daniel Gall

Winter Term 2013
Assignment #4

Constraint Logic Programming

The SWI-prolog documentation of CLP-FD library:
`http://www.swi-prolog.org/man/clpfd.html`

Exercise 1 (Composed Constraints)**.** Constraints can be composed from other constraints. Write the following constraints using `clpfd`:

- `prime(N)`: `N` is a prime number.
- `square(N)`: `N` is a square number.
- `sums(L,S)`: the sum of the numbers in the list `L` is `S`.

Test your constraints for the following queries:

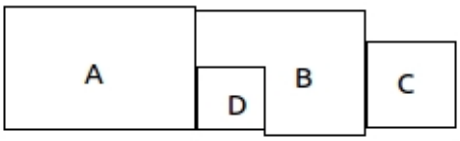
- `prime(1).`
- `prime(2).`
- `prime(3553).`
- `prime(3557).`
- `prime(N).`
- `square(N).`
- `N in 0..100, square(N), label([N]).`
- `[A,B] ins 0..sup, sums([A,B],5), label([A,B]).`

Use your constraints to find the solutions for the following numbers:

- The next prime after 3557.
- All two-digit primes with cross sum 17.
- The sum of a square number and a prime number is 32. The cross sum of the square number is the prime number.

Exercise 2 (Map Coloring)**.** 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 adjcacent 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 on the right using 3 colors, with the predicate:




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

Exercise 3 (Magic Square)**.**
A "magic square" is a rectangular array of *distinct* 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: `daniel.gall@uni-ulm.de`. The deadline is on 27.11.2012 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 4 (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 $1, 2, ..., m \times n$ 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.
Extra: Think of a generic solution for any Survo puzzle.

Constraint Programming

Prof. Dr. Thom Frühwirth
Daniel Gall

Winter Term 2013
Assignment #5

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 paradima.
The following CCLP-clause $H \leftarrow C \mid G$
is written as `N @ H <=> C | G`
where `N @` is an optional name for the rule.

- Read the SWI-Prolog manual on CHR: `http://www.swi-prolog.org/man/chr.html`
- Before using CHR, the CHR library must be included, `:- use_module(library(chr)).`
- User-defined CCLP predicates must be declared, `:- chr_constraint contraint/arity.`
- 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, X = b | true.
p5 @ p(X) <=> X = a | X = b.
p6 @ 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 minimum program in CHR:

- Variant 1:

$$min_{1,1} \ @ \ min_1(X,Y,Z) \leftarrow \top \mid X \leq Y, \ Z = X$$
$$min_{1,2} \ @ \ min_1(X,Y,Z) \leftarrow \top \mid Y \leq X, \ Z = Y$$

- Variant 2:

$$min_{2,1} \ @ \ min_2(X,Y,Z) \leftarrow X \leq Y \mid Z = X$$
$$min_{2,2} \ @ \ min_2(X,Y,Z) \leftarrow Y \leq X \mid Z = Y$$

- Variant 3:

$$min_{3,1} \ @ \ min_3(X,Y,Z) \leftarrow X \leq Y, \ Z = X \mid \top$$
$$min_{3,2} \ @ \ min_3(X,Y,Z) \leftarrow Y \leq X, \ Z = Y \mid \top$$

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,...`

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:

$$[deposit(100), withdraw(25), balance(75)]$$

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: `daniel.gall@uni-ulm.de`. The deadline is on 04.12.2013 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,...`

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
Daniel Gall

Winter Term 2013
Assignment #6

Constraint Handling Rules

For debugging, you can use the tracer. To enable the tracer, enter "`chr_trace`". To disable it, 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 similar rules produce different answers.

```
c1 @ c(X), c(X) <=> q(X,X).
c2 @ c(X), c(Y) <=> r(X,Y).
c3 @ c(X), c(X) ==> q(X,X).
c4 @ c(X), c(Y) ==> r(X,Y).
```

Queries:

a) `c(X), c(X)`
b) `c(X), c(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).
q4 @ q(Z,Y), p(X,Z) ==> q(X,Y).
q5 @ p(X,Z) \ q(Z,Y) <=> q(X,Y).
q6 @ q(Z,Y) \ p(X,Z) <=> q(X,Y).
```

Queries:

d) `p(a,b), q(b,c)`
e) `p(a,b), q(b,c), p(d,a)`

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 $<$) and `leq/2` (encoding $\leq$) and their mutual relations/interactions in CHR. Use the lecture's CHR program for the $\leq$ constraint helpful (also present on WebCHR). 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` .
Do additional tests consisting of combined `less` and `leq` constraints.

Submit the next exercise by e-mail to: `daniel.gall@uni-ulm.de` latest by 11.12.2013, 10:00.

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
Daniel Gall

Winter Term 2013
Assignment #7

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 promted. 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 #6-2 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]).
```

Submit the next exercise by e-mail to: `daniel.gall@uni-ulm.de`, latest by 18.12.2013 at 10:00.

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 #6-2. 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 unequality 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
Daniel Gall

Winter Term 2013
Assignment #8

Constraint System B

Download `boole.pl` from the lecture web page (also sent to your email): `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.

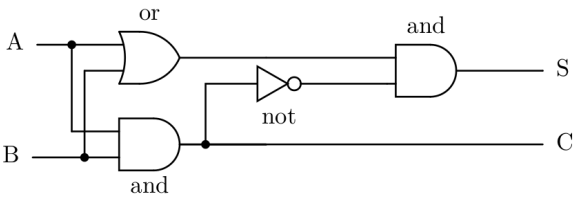
Exercise 1 (Equivalence)**.**

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 (Half Adder)**.**

Write a CHR constraint `add(A,B,S,C)`, which implements a half adder by means of Boolean constraints.
Test with queries `add(1,0,S,C)`, `add(A,B,S,1)` and `add(A,B,S,0)`.



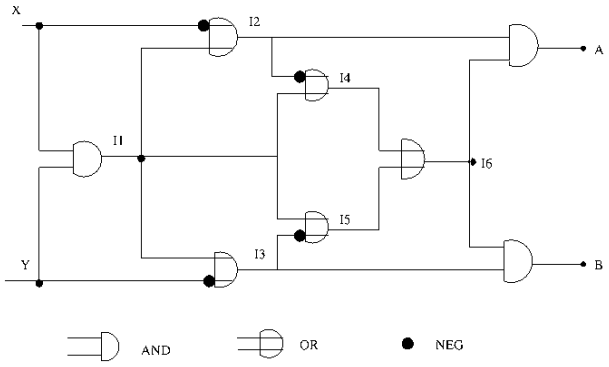
Exercise 3 (Who is lying?)**.**

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`.

Exercise 4 (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)`.



Submit the next exercises by e-mail to: `daniel.gall@uni-ulm.de`, latest by 08.01.2014 at 10:00.

Exercise 5 (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.

Exercise 6 (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)).
```

Constraint Programming

Prof. Dr. Thom Frühwirth
Daniel Gall

Winter Term 2013
Assignment #9

Constraint System B revisited

Exercise 1 (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
```

Constraint System *Rational Trees*

Exercise 2. Implement the axioms of the theory of rational trees for the CHR equality constraint `X eq Y`.

Hints:

- Change terms to list of functor and arguments by: `f(X1,...,XN)=..[f,X1,...,XN]`
- 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) eq p(Y,X,c,a,W)`
gives $Y = a$, $Z = W$

(b) `p(Y,g(Y,f(Y)),g(X,a)) 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)))) 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)) 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 eq f(Y), Y eq f(X).`

Submit the next exercise by e-mail to: `daniel.gall@uni-ulm.de`, latest by 15.01.2014 at 10:00.

Exercise 3. The theory of rational trees should define the (purely) syntactic inequality $\dot{\neq}$ between two terms :

irreflexivity	$\forall(x \dot{\neq} x \rightarrow \bot)$
symmetry	$\forall(x \dot{\neq} y \rightarrow y \dot{\neq} x)$
compatibility	$\forall(x_1 \dot{\neq} y_1 \vee \ldots \vee x_n \dot{\neq} y_n \rightarrow f(x_1,\ldots,x_n) \dot{\neq} f(y_1,\ldots,y_n))$
decomposition	$\forall(f(x_1,\ldots,x_n) \dot{\neq} f(y_1,\ldots,y_n) \rightarrow x_1 \dot{\neq} y_1 \vee \ldots \vee x_n \dot{\neq} y_n)$
distinctness	$\forall(\top \rightarrow f(x_1,\ldots,x_n) \dot{\neq} g(y_1,\ldots,y_m))$ if $f \neq g$ or $n \neq m$

Implement a CHR-constraint `X 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. Specialize the symmetry rule similarly to the `orientation` rule for `eq` such that it exchanges the variables of a `neq` constraint if they are not in the correct order (`@<`).

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. Note that for CHR$^\vee$ rules, some Prolog versions might confuse the symbols `|` and `;`. Hence, rules using disjunction should be written as follows:
`rule @ Head <=> true | (Goal1 ; Goal2).` with a compulsory guard.
Test your implementation for the following queries.

Queries:

(a) `X neq f(X)`
(b) `f(a,X) neq f(X,Y)`
(c) `f(g(X),a) neq f(Y,X)`

Constraint Programming

Prof. Dr. Thom Frühwirth
Daniel Gall

Winter Term 2013
Assignment #10

Constraint System *Feature Terms*

Exercise 1 (Feature Term Solver)**.**
Implement the *FT* solver from the lecture slides and extend it with negation, according to the slides. Test your solver with three meaningful examples (of your choice) and document the result.
Extra: The solver from the slides relies on the built-in equality constraint `=`. Write a solver without the use of the built-in equality but the feature term equality constraint instead, i.e. that implements the axioms of `eq`. (Make sure to model the interaction between `eq` and the other constraints!) Test your implementation with your examples from above and document the results.

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 2 (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 3 (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. Indentify 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 anwer to the goal `z521::train` (use labeling.).
(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: `daniel.gall@uni-ulm.de`. The deadline is on 22.01.2014 by 10:00. For bonus points, submit your *complete* solution of exercise 1 (including the extra task).

Exercise 4. 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
Daniel Gall

Winter Term 2013
Assignment #11

Constraint System *FD*

Exercise 1 (Extension of *FD*-solver)**.**
Extend the finite domain constraint solver over **enumeration domains** presented in the lecture.

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`. Tests should include `maximum(X,Y,Z),X in [0,1],Y in [2,4],Z in [3,4]`.
c) 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 (N-Queens)**.**
Implement a solver for the N-Queens puzzle as covered in the lecture. You can change `no_attack/3` to the following:

```
no_attack(X,Y,N) <=> X neq Y,
                     add(X,N,XPN), XPN neq Y,
                     add(Y,N,YPN), YPN neq X.
```

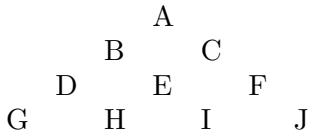
Hint: You will need to implement a special handler for the `add/3` *constraint. Only consider propagation where the first input values is fixed/known and the second is a number.*

Exercise 3 (Sudoku)**.**
Sudoku is a logic puzzle which is solved using logic and reasoning. The objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 sub-grids (or blocks) that compose the grid contains all of the digits from 1 to 9. Implement a sudoku solver in CHR using the above FD-solver.

Hint: Represent each puzzle cell using a constraint `cell(R,C,B,V)`*, where the value* `V` *is stored in the puzzle at row* `R` *and column* `C` *which belongs to the sub-grid or block* `B`*. Then write the appropriate rules that propagate the relevant* `neq` *constraints. Create a* `sudoku` *constraint that produces all 64 cell constraints with the correct variables.*

Submit the next exercise by e-mail to: `daniel.gall@uni-ulm.de`, latest by 29.01.2014 at 10:00.

Exercise 4 (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. $A = |B - C|$), and the numbers are the positive integers from 1 to 10.



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.*

Constraint Programming

Prof. Dr. Thom Frühwirth
Daniel Gall

Winter Term 2013
Assignment #12

Constraint System $\Re$

Exercise 1 ($\Re$ as special RT solver)**.**
Modify the solver for rational trees (RTs) of Assignment #9:

```
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 | T1 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 eq A + B*Y + ...` for variables, `X, Y, ...` and numbers `A, B, ....`

You can test your solver with queries like:

```
5+3*X eq 5+3*X.
Y eq 7+2*X+1*Z, Y eq -2+3*X, X eq 3.
Y eq 7+1*Z+2*X, Y eq -2+3*X, X eq 3.
```