Constraint Handling Rules

Henning Christiansen
Roskilde University, Computer Science Section
KIIS course, 9 Sept 2008

1. What is constraint logic programming?
2. Intro to CHR + small exercise

1 Constraint Logic Programming & Constraint Handling Rules

Historical background

Constraint logic programming, e.g., CLP($\mathcal{R}$), since 80'es or earlier.
Idea is to treat arithmetic (etc.) in logic programming, with *all* its nice properties:
Reversibility, delayed commitment

Early constraint solvers written as "black-boxes" in C or similar

CHR: an attempt to provide a white-box approach $\approx$ *declarative* language for writing constraint solvers; early 90es; in SICStus from 1998

CHR has proved itself successful for a wide range of other application, e.g.,

- Artificial intelligence and reasoning (e.g., DemoII, Abduction – later in the course)
- Language processing (CHR Grammars, CHRG, Hyprolog)
- General algorithmic language (?!?!?)
- ... check out CHR's www pages for many other applications

Several implementations; significant improvements and optimizations

2 Syntax and semantics of CHR

CHR is an extension of Prolog, inherits its nomenclature, and extends with new sorts of rules:

$$\text{Simplification rules: } c_1, \ldots, c_n \ \texttt{<=>}\ \mathit{Guard} \mid c_{n+1}, \ldots, c_m$$
$$\text{Propagation rules: } c_1, \ldots, c_n \ \texttt{==>}\ \mathit{Guard} \mid c_{n+1}, \ldots, c_m$$

Operational semantics: Transformations on a *constraint store*.
Simp's replace constraints, Prop's add new ones.
Committed choice
NB: Uses one-way matching in the head and not unification!

Declarative semantics: Indicated by arrow, bi-implic. & implic.

In practice: Body can be any Prolog code;
– op.sem. is try from top when constraint is called
– left-to-right, depth first

One more kind of rules:

$$\text{Simpagation rules: } c_1, \ldots, c_i \setminus c_{i+1}, \ldots c_n \ \texttt{<=>}\ \mathit{Guard} \mid c_{n+1}, \ldots, c_m$$
$$\text{which can be though of as: } c_1, \ldots, c_n \ \texttt{<=>}\ \mathit{Guard} \mid c_1, \ldots, c_i, c_{n+1}, \ldots, c_m$$

But operationally a bit smarter.

3 Example: Greatest common divisor

```
:- use_module( library(chr)).
handler gcd.
constraints gcd/1.

gcd(0) <=> true.
gcd(N) \ gcd(M) <=> N=<M | L is M-N, gcd(L).
```

Here are a few test queries.

```
?- gcd(2),gcd(3).
?- X is 37*11*11*7*3, Y is 11*7*5*3, Z is 37*11*5, gcd(X), gcd(Y), gcd(Z).
```

4 Another example: Prime numbers

```
:- use_module(library(chr)).
handler primes.
constraints primes/1, prime/1.

primes(1) <=> true.
primes(N) <=> N>1 | M is N-1, prime(N), primes(M).
prime(I) \ prime(J) <=> J mod I =:= 0 | true.
```

5 Exercise 7.1, p. 60

The following program generates Fibonacci numbers. When you give the query `?- fib(5,N).`, the value returned for `N` is the `N`th Fibonacci number; in addition, the program produces constraints that stores all Fibonacci up till the `N`th.

Fibonacci numbers are the series 1, 1, 2, 3, 5, 8, 13, . . ., i.e., each number is the sum of the two previous ones.

```
:- use_module(library(chr)).
handler fibonacci.
constraints fib/2. % fib(N,F): The Nth Fibonacci number is  F

fib(N,F1)\fib(N,F2) <=> F1=F2.
fib(1,X) ==> X=1.
fib(2,X) ==> X=1.
fib(N,F) ==> N > 2 | N1 is N-1, fib(N1,F1), N2 is N-2, fib(N2,F2), F is F1+F2.
```

This program illustrates the difficult procedural issues of CHR, and the question in this exercise is to compare execution speed for the program as it is and if you swop the first and last rules. Logically, the two program versions are identical and produces the same result; can you argue why one version takes considerably more time than the other one?

6 CHR as a forward chaining expert system shell

```
:- use_module(library(chr)).
handler forward_chaining.
constraints fact/1.

fact(rains), fact(go_out), fact(have_not(umbrella)) ==> fact(get_wet).
fact(shower_bath) ==> fact(get_wet).
fact(need(X)), fact(have_not(X)) ==> fact(go_out), fact(buy(X)).
fact(thirsty) ==> fact(need(beer)).

/*******
?- fact(thirsty), fact(rains), fact(have_not(beer)), fact(have_not(umbrella)).
***/
```

Discuss, why may this rule be useful as first one of this program:

```
fact(X) \ fact(X) <=> true.
```

7 Further on forward chaining in CHR

- [HC1] discusses how conflict resolution (see [MN]) and possibly weighted answers can be obtained in CHR.
- We will later see how Prolog + CHR provides a combination of backward and forward chaining (in section on abduction).

8 Exercise 7.2, p. 61

Identify some problem field of which you have good knowledge, and write down some if-then rules about that domain similarly to those of section 7.2.1.

Implement the rules as a forward chaining CHR program as indicated in section 7.3.

Maybe something about computers that won't work

....