The CLP Scheme

CLP($\mathcal{D}$) is a language framework, where D is the domain of the constraints.

Example CLP languages:

- Prolog
- CHIP
- Prolog III – domain is rationals, booleans, and trees
- CLP (Σ*) – domain is regular sets
- CLP($\mathcal{R}$) – domain is reals (plus trees, i.e. the data types that Prolog uses)

CLP($\mathcal{R}$) – Domain and Solver

CLP($\mathcal{R}$) can solve arbitrary collections of linear equality and inequality constraints over the real numbers.

It can also solve other kinds of constraints over the reals if it can find the answer using one-step deductions (first find this variable using one constraint, then find another variable using another constraint, etc — but no simultaneous equations).

Besides the domain of the real numbers, CLP($\mathcal{R}$) has another domain: trees. These allow us to model data structures such as lists, records, and trees.

CLP($\mathcal{R}$) Examples

Sample goals (just using primitive constraints – no user-defined rules) and answers:

```
?- X=Y+1, Y=10.
  X=11, Y=10

?- 2*A+B=7, 3*A+B=9.
  A=2, B=3

? X>=2*Y, Y>=5, X<=10.
  X=10, Y=5.

?- X*X*X + X = 10.
  maybe
```

(The last goal does have a solution `X=2`. The "maybe" answer means the constraints are too hard for CLP($\mathcal{R}$) to solve.)

CLP($\mathcal{R}$) Examples

CLP($\mathcal{R}$) programs are collections of *facts* and *rules*.

Sample rule:

```
/* centigrade-fahrenheit relation */
cf(C,F) :-
  F = 1.8*C + 32.
```

Sample Goals:

```
?- cf(100,A).
  A=212.0

?- cf(A,B), A>100, B<200.
  no.

?- cf(X,X).
  X=-40.0
```

Evaluation in CLP Languages – Informal Discussion

Given an initial goal, a CLP interpreter rewrites any user-defined constraints in the goal using their definitions.

This may yield more user defined constraints, which are then rewritten.

Primitive constraints are kept in a *constraint store*.

We continue until there are only primitive constraints, which are solved by the system.

However, if the constraint store contains an unsatisfiable set of constraints, we can stop rewriting immediately.

We may have multiple rules for a given user-defined constraint. We try these in order, backtracking if one fails.

Example Derivation

CLP($\mathcal{R}$) program:

```
cf(C,F) :-   /* rule R1 */
  F = 1.8*C + 32.

double(X,Y) := Y=2*X.  /* rule R2 */
```

Consider the goal `cf(A,B), double(A,200).`

$\langle$*cf*(A, B), *double*$(A, 200) \mid$ *true*$\rangle$

$\Rightarrow$

using R1:
$\langle A = C, B = F, F = 1.8 * C + 32,$
double$(A, 200) \mid$ *true*$\rangle$

$\Rightarrow$

$\langle B = F, F = 1.8 * C + 32,$ *double*$(A, 200) \mid$
$A = C\rangle$

$\Rightarrow$

$\langle F = 1.8 * C + 32,$ *double*$(A, 200) \mid A = C,$
$B = F\rangle$

$\Rightarrow$

$\langle$*double*$(A, 200) \mid A = C, B = F,$
$F = 1.8 * C + 32\rangle$

$\Rightarrow$

using R2:
$\langle A = X, 200 = Y, Y = 2 * X \mid A = C, B = F,$
$F = 1.8 * C + 32\rangle$

$\Rightarrow$

$\langle 200 = Y, Y = 2 * X \mid A = C, B = F,$
$F = 1.8 * C + 32, A = X\rangle$

$\Rightarrow$

$\langle Y = 2 * X \mid A = C, B = F, F = 1.8 * C + 32,$
$A = X, 200 = Y\rangle,$

$\Rightarrow$

$\langle \Box \mid A = C, B = F, F = 1.8 * C + 32,$
$A = X, 200 = Y, Y = 2 * X\rangle$

Simplifying with respect to the variables in G_0 (namely A, B) we get the answer $A = 100, B = 212$

Some Simple Recursive CLP($\mathcal{R}$) Programs

```
/* LENGTH OF LIST */

length([],0).
length([H|T],N) :-
  N > 0,
  length(T,N-1).

/* compare this with a scheme program:
(define (length x)
  (if (null? x) 0
      (+ 1 (length (cdr x)))))
*/


/* SUM OF THE ELEMENTS IN A LIST */

sum([],0).
sum([X|Xs],X+S) :- sum(Xs,S).
```



```
/* FACTORIAL */
factorial(0, 1).
factorial(N, N * F) :-
        N > 0 ,
        fact(N - 1, F).
```

Greatest Common Divisor

```
/* GREATEST COMMON DIVISOR
   (USING EUCLID'S ALGORITHM) */

gcd(A,B,G) :-
   A < B,
   gcd(A,B-A,G).

gcd(A,B,G) :-
   A > B,
   gcd(A-B,B,G).

gcd(A,A,A).
```


Quicksort

```
quicksort([],[]).
quicksort([X|Xs],Sorted) :-
  partition(X,Xs,Smalls,Bigs),
  quicksort(Smalls,SortedSmalls),
  quicksort(Bigs,SortedBigs),
  append(SortedSmalls,[X|SortedBigs],Sorted).

partition(Pivot,[],[],[]).
partition(Pivot,[X|Xs],[X|Ys],Zs) :-
   X <= Pivot,
   partition(Pivot,Xs,Ys,Zs).
partition(Pivot,[X|Xs],Ys,[X|Zs]) :-
   X > Pivot,
   partition(Pivot,Xs,Ys,Zs).


append([],X,X).
append([X|Xs],Ys,[X|Zs]) :- append(Xs,Ys,Zs).
```