

Constraint Logic Programming (Ch.14)

- **Constraint:** a requirement for, or limitation on, a variable
- **Constraint satisfaction:** finding combinations of variables that satisfy a set of constraints
- **constraint programming:** algorithms that implement constraint sat.
- **Constraint Logic Programming (CLP):** merging CP with LP

- **A constraint satisfaction problem:**
 - a) set of variables
 - b) domains for the variables (float, integer, ...)
 - c) constraints the variables have to satisfy
 - **Problem:** find an assignment to the variables that satisfies constraints

- **Constraint solver:** an algorithm which, given the above, tries to efficiently find a solution

- **CLP is implemented by adding a constraint solver to Prolog**

CLP

- **Types of CLP:**
 - **CLP(R): variables take values over reals (decimal points)**
 - **CLP(Z): integers**
 - **CLP(Q): rationals (fractions)**
 - **CLP(FD): finite domains (sets) and others**
- **Good for solving problems involving:**
 - **scheduling**
 - **linear equations and optimization**
 - **logistics**
- **To run with SICStus:**
 - ?- **use_module(library(clpq)). % CLP package for rationals (fractions)**
 - ?- **use_module(library(clpr)). % CLP package for reals**
 - ?- **use_module(library(clpfd)). % CLP package for finite domains (sets)**

Example 1: arithmetic

?- 1+X = 5.

no

?- 1+X is 5.

no

?- {1+X = 5}. % enclose constraint in {}'s

X = 4

Example 2: Arith

convert(Cent, Fahren) :- **% v1: Faren to Cent only**
 Cent is (Fahren - 32)*5/9.

convert2(Cent, Fahren) :- **% v2: F to C, or C to F, but must use 'var'**
 var(Cent),
 Cent is (Fahren - 32)*5/9.

convert2(Cent, Fahren) :-
 var(Fahren),
 Fahren is 32 + (Cent * 9/5).

convert3(Cent, Fahren) :-
 {Cent = (Fahren - 32)*5/9}.

?- **convert3(35, F).**

F = 95

?- **convert3(C, 95).**

C = 35

?- **convert3(C, F).**

{F = 32.0 + 1.8*C}

?- **convert3(C,F). % using rationals**

{F=32+9/5*C}

Example: integers and Fibonacci

% Regular Prolog...

fib(0, 1).

fib(1, 1).

fin(N, F) :-

N > 1,

N1 is N-1,

fib(N1, F1),

N2 is N-2,

fib(N2, F2),

F is F1+F2.

?- fib(6, F).

F = 13

?- fib(N, 13).

ERROR

Example: Integers and Fibonacci

% CLP...

fib(N, F) :- {N=0, F=1}.

fib(N, F) :- {N=1, F=1}.

fib(N, F) :-

{N>1,

F=F1+F2,

N1 = N-1,

N2 = N-2},

fib(N1, F1),

fib(N2, F2).

?- fib(N, 13).

N = 6

?- fib(N, 4).

INF LOOP -> stack overflow

Fibonacci

- Infinite loop happens because constraint solver doesn't know that there is no fib number = 4; keeps recursing, finding larger and larger possibilities
- Solution: add more constraints
 - $\text{Fib}(N) \geq N$
 - then it won't search for any $N > F$

```
% CLP...
fib(N, F) :- {N=0, F=1}.
fib(N, F) :- {N=1, F=1}.
fib(N, F) :-
    {N>1,
     F=F1+F2,
     N1 = N-1,
     N2 = N-2,
     F1 >= N1,      % new constraints
     F2 >= N2},
    fib(N1, F1),
    fib(N2, F2).
```

Example: money puzzle!

- This implementation (module clpfd - finite domains) uses its own operators, utilities (no { }'s):
 - #= is constraint '='
 - all_different/1: indicates all variables in argument must differ
 - domain/3: indicates integer domain of variables
 - labeling/2: does the actual inference (backtracking, etc.)

Money puzzle

```
?- use_module(library(clpfd)).
```

```
% step 1: declare domains of the variables
```

```
% step 2: post the problem constraints
```

```
% step 3:
```

```
money([S,E,N,D,M,O,R,Y], Type) :-
```

```
    domain([S,E,N,D,M,O,R,Y], 0, 9),    % step 1
```

```
    S#>0,
```

```
    M#>0,
```

```
    all_different([S,E,N,D,M,O,R,Y]),    % step 2a
```

```
    sum(S,E,N,D,M,O,R,Y),                % step 2b
```

```
    labeling(Type, [S,E,N,D,M,O,R,Y]). % step 3
```

```
sum(S,E,N,D,M,O,R,Y) :-
```

```
    1000*S + 100*E + 10*N + D
```

```
    + 1000*M + 100*O + 10*R + E
```

```
    #= 10000*M + 1000*O + 100*N + 10*E + Y.
```

Conclusion

- **CLP is a new, powerful form of logic programming**
 - gives Prolog the “smarts” about arithmetic, sets, numbers
 - Prolog’s search still used; constraints help guide search, and are used to determine values and ranges
- **Many nontrivial problems are solvable with CLP, which would be difficult for plain Prolog.**