

Introduction to CLP

- introduction
- constraint domains
- CLP(\mathbb{R})
- CLP(finite domain)

- born with AI; constraints used as **problem representation**
- a problem is specified as a **set of constraints**
- solution is an **assignment of values** that satisfies them
- example: $SEND + MORE = MONEY$ puzzle

History

- some ancestors: SKETCHPAD (1963), THINGLAB(1981), MACSYMA (1983), Waltz's algorithm, ...
- constraint solving in logic languages (general theory and practical systems)
- equation solving libraries (ILOG) in imperative languages
- extensions of functional languages (function evaluation with free variables)

Features

- **Constraint Programming** is a useful paradigm for formulating and solving problems
- those problems that can be naturally defined in terms of constraints among a set of variables, called **constraint satisfaction**
- **CLP** (constraint logic programming) combines the constraint approach with logic programming
- where constraint satisfaction is embedded into a logic programming language

Features

- a constraint satisfaction problem is stated as follows:
 - a set of **variables**
 - the **domain** from which the variables can take values
 - a set of **constraints** the variables have to satisfy
- the solution to these problems is an **assignment** of values to the variables that satisfies the constraints,
- or another set of constraints which is a **simplification** of the given one.

Features

- in optimization problems, when there are several assignments that satisfy the constraints, we can specify a **criterion** to choose among them
- typical application for CLP come from scheduling, logistics, resource management, etc
- example: tasks: a, b, c, d of durations 2, 3, 4, 5 hours respectively, with the following constraints: a has to precede b, c , and b have to precede d .
Problem: find starting times Ta, Tb, Tc, Td so that the starting time is 0 and the finishing time F is minimal.
- solution: $Ta = 0, Tb = 2, 2 \leq Tc \leq 4, Td = 5, F = 9$

Features

- constraint satisfaction problems are often depicted as graphs, called **constraint networks**
- the **nodes** of the graph correspond to the variables
- the **arcs** of the graphs correspond to the constraints
- for each binary constraint $p(X, Y)$ there are two directed arcs (X, Y) and (Y, X) in the graph
- only binary constraints are allowed, but others can be expressed in these terms
- to find a solution, various **consistency algorithms** can be used

A comparison with Prolog: pure Prolog

```
q(X,Y,Z) :- Z = X+Y.  
p(X,Y,Z) :- Z is X+Y.
```

```
? q(3,4,Z).  
Z=3+4
```

```
? q(X,Y,3+4).  
X=3, Y=4
```

```
? q(X,Y,Z).  
Z=X+Y
```

```
? p(3,4,Z).  
Z=7
```

```
? p(X,4,7).  
{INSTANTIATION ERROR}
```

A comparison with Prolog: CLP(Real)

```
q(X,Y,Z) :- Z .=. X+Y.
```

```
% no is/2 predicate !! It is subsumed by .=./2
```

```
? q(3,4,Z).
```

```
Z=7
```

```
? q(X,4,7).
```

```
X=3
```

```
? q(X,Y,7).
```

```
X=7-Y
```

A comparison with LP

- Advantages:
 - helps making programs expressive and flexible
 - may save much coding
 - in some cases more efficient solving due to specific solvers
 - search space reduction (LP: generate and test; CLP : constrain and generate)
- Disadvantages:
 - complexity of solver algorithms can affect performance
- Solutions:
 - better algorithms
 - compile-time optimizations (program transformations, global analysis, etc)
 - parallelism

- example of search space reduction (generate-and-test vs constraint-and-generate)

```
solution(X,Y,Z) :- p(X), p(Y), p(Z), test(X,Y,Z).
test(X,Y,Z) :- Y is X+1, Z is Y+1.
p(14). p(15). p(16). p(7). p(3). p(11).
% 458 steps to find 1st solution (all solution 465 steps)
```

```
solution(X,Y,Z) :- test(X,Y,Z), p(X), p(Y), p(Z).
test(X,Y,Z) :- Y .=. X+1, Z .=. Y+1.
p(14). p(15). p(16). p(7). p(3). p(11).
% 11 steps to find 1st solution (all solution 11 steps)
```

- we show the main ideas of a **constraint satisfaction algorithm**
- these solvers are based in the observation that if the domain for any variables is **empty**, then the problem is unsatisfiable
- so the algorithm works on the idea of transforming the formulation of the problem into an equivalent one in with **decreased domain**
- **equivalent** means that the two problems have the same set of solutions
- these solvers work by considering **each constraint** in turn, and using information about the domain of each variable to eliminate values from the domain of the other variables

- consider variables X, Y with domains Dx, Dy respectively, and a binary constraint $p(X, Y)$
- the arc (X, Y) is said to be **arc consistent** if for each value in Dx for X there is some value in Dy for Y satisfying $p(X, Y)$
- if (X, Y) is not arc consistent, then we can **delete** all values in Dx for which there is no corresponding value in Dy
- this step makes (X, Y) arc consistent
- example: X, Y in $0..10$ and $X + 4 \leq Y$

- after reducing some domain, some **other arcs** may become inconsistent
- so we can apply again this step to one inconsistent arc
- this may propagate through the network, possibly **cyclically**, for some time until either all the arcs become consistent, or some domain becomes empty.
- if **all arcs become consistent**, and we have only **valuation domains** (all variables are mapped to singletons) then we have a solution
- if **all arcs become consistent**, and there is at least one **multiple-valued**, then this is not a guarantee that all possible combinations of domain values are solutions to the constraints
- it may be even the case that no solution exists
- example: $X \in \{6, 7\}$, $Y \in \{4, 3\}$ with constraints $X + Y = 10$, $odd(X + Y)$

- some **combinatorial search** is needed to choose one of the multi-valued domains and try to find a solution
- one possibility is to assign **an arbitrary value** to variable with a multi-valued domain, and then searching for possible new inconsistent arcs so the previous algorithm can be applied again.
- if the domains are **finite**, this will eventually result in an empty domain, or all the domains single-valued.
- several alternatives strategies exist for this search

- example: let's consider the constraint graph of the previous scheduling problem

Step	Arc	T_a	T_b	T_c	T_d	F
start		0..10	0..10	0..10	0..10	0..10
1	(T_b, T_a)		2..10			
2	(T_d, T_b)				5..10	
3	(T_f, T_d)					9..10
4	(T_d, T_f)				5..6	
5	(T_b, T_d)		2..3			
6	(T_a, T_b)	0..1				
7	(T_c, T_a)			2..10		
8	(T_c, T_f)			2..5		

Constraint Domains

- semantics of CLP languages is parameterized by the **constraint domain**: $\text{CLP}(X)$ where $X = (\Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T})$
- signature Σ contains a set of predicate and function symbols, with arities
- $\mathcal{L} \subseteq \Sigma$ – formulas are the constraints
- \mathcal{D} is the actual elements in the domain. Pre-interpretations on \mathcal{D} give the meaning of predicate and function symbols
- \mathcal{T} is a FOL theory axiomatizing some properties of \mathcal{D}
- Assumptions: \mathcal{L} is built upon a first-order language, $= \in \Sigma$ is identity in \mathcal{D} , etc.

Examples:

- Let's assume $\mathfrak{R} = (\Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T})$ where $\Sigma = \{0, 1, +, *, =, <, \leq\}$, $D = \mathbb{R}$, \mathcal{D} and \mathcal{T} interprets Σ as usual, and L is any formula in this language
- then this is arithmetic over the reals, e.g. $x^2 + 2xy < y/x \wedge x > 0$
- Let's assume $\mathfrak{R}_{Lin} = (\Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T})$ where $\Sigma = \{0, 1, +, =, <, \leq\}$, $D = \mathbb{R}$, \mathcal{D} and \mathcal{T} interprets Σ as usual, and \mathcal{L} is a set of linear disequations
- then this is linear arithmetic, e.g. $3x - y < 3$
- Let's assume $\Sigma = \{0, 1, +, =\}$, $\mathfrak{R}_{LinEq} = (\mathcal{D}'', \mathcal{L}'')$
- then this are linear equations, e.g. $3x + y = 5 \wedge y = 2x$

Examples:

- $\Sigma = \{\langle \text{variable and function symbols} \rangle, =\}$, $D =$ finite trees, \mathcal{D} interprets Σ as tree constructors
- Constraints: syntactic tree equality
- $\mathcal{FT} = (D, \mathcal{L})$ constraints over the Herbrand domain, e.g. $g(h(Z), Y) = g(Y, h(a))$
- then $LP \equiv CLP(\mathcal{FT})$

Examples:

- $\Sigma = \{\langle \text{constants symbols} \rangle, \lambda, ., ::, =\}$, $D =$ finite strings of constants, \mathcal{D} interprets $.$ as concatenation, $::$ as string length
- then we have equations over string constants, e.g. $X.A.X = X.A$
- $\Sigma = 0, 1, \neg, \wedge, =$, $D = \text{true, false}$, \mathcal{D} interprets the symbols in Σ as boolean functions
- then we have boolean constraints, e.g. $\neg(x \wedge y) = 1$

CLP(\mathbb{R})

- CLP(\mathbb{R}) is a language based on Prolog, with the addition of constraint solving capabilities over the reals
- CLP(\mathbb{R}) is able to solve directly linear (dis)equations over the reals
- non-linear equations are delayed, waiting for them to eventually become linear

Example: Fibonacci numbers

```
fib(0,0).  
fib(1,1).  
fib(N, F) :- N.>.1, F.=.F1+F2, F1.>=.0, F2.>=.0,  
             fib(N-1,F1), fib(N-2,F2).
```

```
? fib(6,F).  
F=13
```

```
? fib(N,13).  
N=6
```

```
? fib(N,F).  
{N=0,F=0}; {N=1,F=1}; {N=2,F=1}; {N=3,F=2}...
```

```
? fib(N,4).  
...
```

Example: Vector dot product

```
prod([], [], 0).  
prod([X|XTail], [Y|YTail], R) :- R = .X*Y+Rest, prod(XTail, YTail, Rest).
```

```
? prod([2,3], [4,5], R).  
R=23
```

```
? prod([2,3], [5,X], 22).  
X=4
```

```
? prod([2,7,3], [X,Y,Z], 0).  
X=-1.5*Z-3.5*Y
```

- A result is in general an [equation over the variables in the query](#).

Example: Vector dot product Equations

$$3x + y = 5 \quad (1)$$

$$x + 8y = 3 \quad (2)$$

can be solved with

? `prod([3,1],[X,Y],5), prod([1,8],[X,Y],3).`
X=1.6087, Y=0.173913

CLP(finite domain)

- A **finite domain** constraint solver associates each variable with a finite subset of \mathbb{Z}
- $E \in \{-123, -10..4, 10\}$ (represented as $E :: [-123, -10..4, 10]$)
- we can perform arithmetic operation on the variables, and establish linear relationships among expressions ($. = ., . < ., . = < .$)
- these constraints are intended to narrow the domains of the variables
- `fd_min(X, Y)` can be used to set in Y the minimum value in the domain of X

Examples:

? X .=. A+B, A in 1..3, B in 3..7.
X in 4..10, A in 1..3, B in 3..7

? X .=. A-B, A in 1..3, B in 3..7.
X in -6..0, A in 1..3, B in 3..7

? X .=. A-B, A in 1..3, B in 3..7, X .>=. 0.
A=3, B=3, X=0

Example: Project Management Problem

- tasks and duration: $(B, 1)$, $(C, 2)$, $(D, 3)$, $(E, 4)$, $(F, 1)$
- project must be finished before time 10
- constraints: B before E , C before E , C before F , D before F
- normalization of the graph is necessary.

Example: Project Management Problem

```
pn(A,B,C,D,E,F,G) :- A .>=. 0, G .=<. 10,  
    E.>=.B+1, E.>=.C+2, F.>=.C+2, F.>=.D+3,  
    G.>=.E+4, F.>=.F+1,  
    B.>=.A, C.>=.A, D.>=.A.
```

```
? pn(A,B,C,D,E,F,G).  
A in 0..4, B in 0..5, C in 0..4, D in 0..6,  
E in 2..6, F in 3..9, G in 6..10
```

```
? pn(A,B,C,D,E,F,G),minimize(G,G).  
A=0, B in 0..1, C=0, D in 0..2,  
E=2, F in 3..5, G=6
```

Example: *Send + More = Money* puzzle

```
smm(S,E,N,D,M,O,R,Y) :-  
    domain([S,E,N,D,M,O,R,Y],0,9),  
    constrain([S,E,N,D,M,O,R,Y]),  
    labeling([S,E,N,D,M,O,R,Y]).  
  
constrain([S,E,N,D,M,O,R,Y]) :-  
    S .\=. 0, M .\=. 0,  
    all_different([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.
```

Example: N -queens problem

- Problem: place N chess queens in a $N \times N$ board such that they do not attack each other
- A solution is represented as a list holding the column position for each row
- so the final solution is a permutation of $[1, 2, \dots, N]$
- general solution: start with a partial solution, nondeterministically select a new queen, check safety against those already placed, add the queen if compatible.

Example: N -queens problem (cont.) Prolog solution

```
queens(N,Solution) :- queens_list(N,Ns), queens_solve(Ns,[],Solution).

queens_list(0,[]).
queens_list(N,[N|Ns]) :- N>0, N1 is N-1, queens_list(N1,Ns).

queens_solve([],Solution,Solution).
queens_solve(Unplaced, Placed, Solution) :- select(Q,Unplaced,NewUnplaced),
    safe(Placed,Q,1), queens(NewUnplaced, [Q|Placed], Solution).

safe([],_,_).
safe([OtherQueenQs],Queen,Nb) :- Queen =\= OtherQueen+Nb,
    Queen =\= OtherQueen-Nb, Nb1 is Nb+1,
    safe(Qs,Queen,Nb1).

select(X,[X|Ys], Ys).
select(X,[Y|Ys], [Y|Zs]) :- select(X,Ys,Zs).
```

Example: N -queens problem (cont.) CLP(FD) solution

```
queens(N,Solution,Type) :- constrain_values(N,N,Solution),
    all_different(Qs), labeling(Type,Qs).

constrain_values(0,_,[]).
constrain_values(N,Range,[X|Xs]) :- N>0, N1 is N-1, X in 1..Range,
    constrain_values(N1,Range,X), safe(Xs,X,1).

safe([],_,_).
safe([OtherQueenQs],Queen,Nb) :- Queen .\=. OtherQueen+Nb,
    Queen .\=. OtherQueen-Nb, Nb1 is Nb+1, safe(Qs, Queen, Nb1).
```

- **Exercise:** model the following problems, and show their solutions, using CHR and SWI Prolog, or Ciao Prolog.
 - Aaron, Bob and Carla are three suspects for a murder. Anyone who is involved with the murder will always lie, and those not involved will tell the truth. The statements from the suspects are: Aaron - "I am innocent and Bob and Carla were both involved.", Bob - "Carla is innocent.", Carla - "I am guilty or Aaron is.". Can you determine who is involved in the murder? Explain.
 - Determine the satisfiability of the following constraints;
 1. $X = 3 + T \wedge T + X = 3 \wedge Y + 2X = Z \wedge Y + X + 3 = Z$
 2. $2X + 4Y + 6Z = 1 \wedge X + 2Y + 2Z = 0 \wedge Z = 1.$
 - Write a predicate $\text{minmax}(X, Y, Min, Max)$ which holds if Min is the minimum of X and Y and Max is the maximum. Note that the goal $\text{minmax}(X, Y, 3, 4)$ should succeed with two answers and $\text{minmax}(X, Y, 4, 3)$ should fail.