Solving Combinatorial Search Problems Using B-Prolog

Neng-Fa Zhou
周 能法
The City University of New York
zhou@sci.brooklyn.cuny.edu

B-Prolog: Prolog + Tabling + CLP(FD)

- Prolog
  - Rule-based relational language
    - SQL + Recursion + Unification + Backtracking
- Tabling
  - Memorize and reuse intermediate results
    - Suitable for dynamic programming problems
- CLP(FD)
  - Constraint Logic Programming over Finite Domains
    - Suitable for constraint satisfaction problems (NP-complete)

Prolog

- A program consists of *relations* defined by *facts* and *rules*
- *Unification*
- *Recursion*
- Nondeterminism realized through *backtracking*

Prolog – An example

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

```
| ?- cl(app)
Compiling::app.pl
compiled in 0 milliseconds
loading::app.out

yes
| ?- app([a,b],[c,d],L)
L = [a,b,c,d]
yes
| ?- app(L1,L2,[a,b,c])
L1 = []
L2 = [a,b,c] ?;
L1 = [a]
L2 = [b,c] ?;
L1 = [a,b]
L2 = [c] ?;
L1 = [a,b,c]
L2 = [] ?;
no
```

Syntax of Prolog

Term

- Atom
  - string of letters, digits, and '_' starting with a low-case letter
  - string of characters enclosed in quotes
- Number
  - integer & real
- Variable
  - string of letters, digits and '_' starting with a capital letter or '_'

Syntax of Prolog (Cont)

- Structure
  - $f(t_1,t_2,...,t_n)$
    - f is an atom, called the *functor* of the structure
    - $t_1,t_2,...,t_n$ are terms
- List
  - '.'(H,T) => [H|T]
  - '.'(1,'.'(2,'.'(3,[]))) => [1,2,3]

Syntax of Prolog (Cont)

- Clause
  - Fact
    - $p(t_1,t_2,\ldots,t_n)$
  - Rule
    - H :- B1,B2,...,Bm.

Head

Body

- Predicate
  - a sequence of clauses
- Program
  - a set of predicates
- Query

Unification

- t1 = t2 succeeds if
  - t1 and t2 are identical
  - there exists a substitution θ for the variables in t1 and t2 such that t1θ = t2θ.

<u>f(X,b)=f(a,Y)</u>.
X=a
Y=b

$\theta = \{X/a, Y/b\}$

Unification: Examples

?-X=1. assignment
X=1
?- f(a,b)=f(a,b). test
yes
?- a=b. test
no
?- f(X,Y)=f(a,b) matching
X=a
Y=b
?-f(X,b)=f(a,Y). unification
X=a
Y=b
?-X = f(X). without occur checking
X=f(f(......

Operational Semantics of Prolog (Resolution)

G_0: initial query

G_i: $(A_1,A_2,\ldots,A_n)$

$$\Big\downarrow \; H\text{:-}B_1,\ldots,B_m \quad A_1\theta=H\theta$$

G_{i+1}: $(B_1,\ldots,B_m,A_2,\ldots,A_n)\theta$

Succeed if G_k is empty for some k.

Backtrack if G_k is a dead end (no clause can be used).

Deductive Database

```
parent(Parent,Child):-father(Parent,Child).
parent(Parent,Child):-mother(Parent,Child).
uncle(Uncle,Person) :-
        brother(Uncle,Parent), parent(Parent,Person).
sibling(Sib1,Sib2) :-
        parent(Parent,Sib1), parent(Parent,Sib2),
        Sib1 ¥= Sib2.
cousin(Cousin1,Cousin2) :-
        parent(Parent1,Cousin1),
        parent(Parent2,Cousin2),
        sibling(Parent1,Parent2).
```

Exercises

- Define the following relations
  - son(X,Y) -- X is a son of Y
  - daughter(X,Y) -- X is a daughter of Y
  - grandfather(X,Y) -- X is the grandfather of Y
  - grandparent(X,Y) -- X is a grandparent of Y
  - ancestor(X,Y) – X is an ancestor of Y

Recursive Programming on Lists

- A list is a special structure whose functor is '.'/2

```
– []
– '.'(H,T)     => [H|T]
– '.'(1,'.'(2,'.'(3,[])))  => [1,2,3]
```

- Unification of lists

```
– [X|Xs]=[1,2,3]
   X= 1   Xs=[2,3]
– [1,2,3] = [1|[2|X]]
   X=[3]
– [1,2|3] = [1|X]
   X=[2|3]
```

Relations on Lists

- isList(Xs)

```
isList([]).
isList([X|Xs]):-isList(Xs).
```

- member(X,Xs)

```
member(X,[X|Xs]).
member(X,[_|Xs]):-member(X,Xs).
```

- append(Xs,Ys,Zs)

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

- length(Xs,N)

```
length([],0).
length([X|Xs],N):-length(Xs,N1),N is N1+1.
```

Exercise

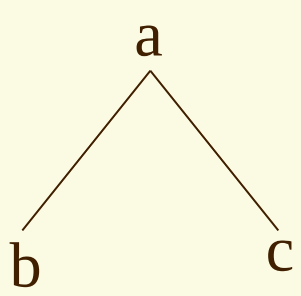
- Implement the following predicates.
  - length(Xs,N)
    - the length of Xs is N
  - last(X,Xs)
    - X is the last element of Xs.
  - prefix(Pre,Xs)
    - Pre is a prefix of Xs.
  - suffix(Pos,Xs)
    - suffix is a postfix of Xs
  - reverse(Xs,Ys)
    - Ys is the reverse of Xs
  - sum(Xs,N)
    - N is the sum of the integers in the list Xs
  - sum1(Xs,Ys)
    - assume Xs is [x1,x2,...,xn], then Ys will be [y1,y2,...,yn] where yi is xi+1.
  - sort(L,SortedL)
    - use the exchange sort algorithm

Recursive Programming on Binary Trees

- Representation of binary trees

`void`	--	empty tree
`t(N, L,R)`	--	N : node
		L : Left child
		R : Right child

- Example



`t(a, t(b, void,void), t(c,void,void))`

Relations on Binary Trees

- `isBinaryTree(T)` -- T is a binary tree

```
isBinaryTree(void).
isBinaryTree(t(N,L,R)):-
        isBinaryTree(L),
        isBinaryTree(R).
```

- `count(T,C)` -- C is the number of nodes in T.

```
count(void,0).
count(t(N,L,R),N):-
        count(L,N1),
        count(R,N2),
        N is N1+N2+1.
```

Relations on Binary Trees (Cont.)

`preorder(T,L)`

- `L` is a pre-order traversal of the binary tree `T`.

```
preorder(void,[]).
preorder(t(N,Left,Right),L):-
      preorder(Left,L1),
      preorder(Right,L2),
      append([N|L1],L2,L).
```

Exercise

- Write the following predicates on binary trees.
  - `leaves(T,L)`: `L` is the list of leaves in `T`. The order is preserved.
  - `equal(T1,T2)`: `T1` and `T2` are the same tree.
  - `postorder(T,L)`: `L` is the post-order traversal of `T`.

Tabling (Why?)

- Eliminate infinite loops

```
:-table path/2.
path(X,Y):-edge(X,Y).
path(X,Y):-edge(X,Z),path(Z,Y).
```

- Reduce redundant computations

```
:-table fib/2.
fib(0,1).
fib(1,1).
fib(N,F):-
    N>1,
    N1 is N-1,fib(N1,F1),
    N2 is N-2,fib(N2,F2),
    F is F1+F2.
```

Mode-Directed Tabling

- Table mode declaration

```
:-table p(M1,...,Mn):C.
```

  - `C`: Cardinality limit
  - Modes
    - `+` : input
    - `-` : output
    - `min`: minimized
    - `max`: maximized

Shortest Path Problem

```
:-table sp(+,+,-,min).
sp(X,Y,[(X,Y)],W) :-
     edge(X,Y,W).
sp(X,Y,[(X,Z)|Path],W) :-
     edge(X,Z,W1),
     sp(Z,Y,Path,W2),
     W is W1+W2.
```

- `sp(X,Y,P,W)`
  - `P` is a shortest path between `X` and `Y` with weight `W`.

Knapsack Problem

http://probp.com/examples/tabling/knapsack.pl

```
:- table knapsack(+,+,-,max).
knapsack(_,0,[],0).
knapsack([_|L],K,Selected,V) :-
    knapsack(L,K,Selected,V).
knapsack([F|L],K,[F|Selected],V) :-
    K1 is K - F, K1 >= 0,
    knapsack(L,K1,Selected,V1),
    V is V1 + 1.
```

- `knapsack(L,K,Selected,V)`
  - `L`: the list of items
  - `K`: the total capacity
  - `Selected`: the list of selected items
  - `V`: the length of `Selected`

Exercises (Dynamic Programming)

1. *Maximum Value Contiguous Subsequence*. Given a sequence of n real numbers a_1, ... a_n, determine a contiguous subsequence A_i ... A_j for which the sum of elements in the subsequence is maximized.
2. Given two text strings A of length n and B of length m, you want to transform A into B with a minimum number of operations of the following types: delete a character from A, insert a character into A, or change some character in A into a new character. The minimal number of such operations required to transform A into B is called the edit distance between A and B.

CLP(FD) by Example (I)

- The rabbit and chicken problem
- The Kakuro puzzle
- The knapsack problem
- Exercises

The Rabbit and Chicken Problem

- In a farmyard, there are only chickens and rabbits. Its is known that there are 18 heads and 58 feet. How many chickens and rabbits are there?

```
go:-
        [X,Y] :: 1..18,
        X+Y #= 18,
        2*X+4*Y #= 58,
        labeling([X,Y]),
        writeln([X,Y]).
```

Break the Code Down

```
go:-
      [X,Y] :: 1..58,
      X+Y #= 18,
      2*X+4*Y #= 58,
      labeling([X,Y]),
      writeln([X,Y]).
```

- `go` -- a predicate
- `X,Y` -- variables
- `1..58` -- a domain
- `X :: D` -- a domain declaration
- `E1 #= E2` -- equation (or equality constraint)
- `labeling(Vars)` -- find a valuation for variables that satisfies the constraints
- `writeln(T)` -- a Prolog built-in

Running the Program

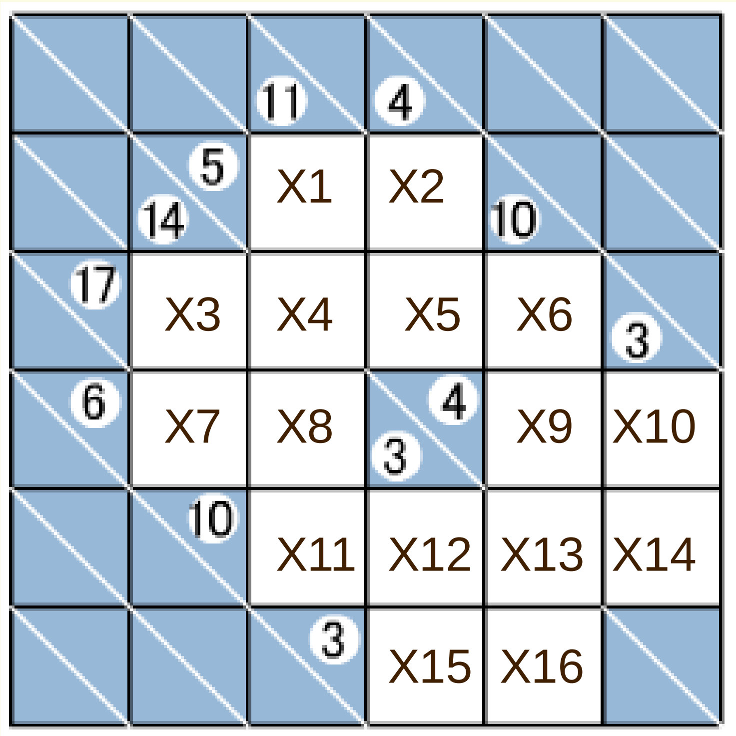
```
| ?- cl(rabbit)
Compiling::rabbit.pl
compiled in 0 milliseconds
loading::rabbit.out

yes
| ?- go
[7,11]
```

The Kakuro Puzzle

- Kakuro, another puzzle originated in Japan after Sudoku, is a mathematical version of a crossword puzzle that uses sums of digits instead of words. The objective of Kakuro is to fill in the white squares with digits such that each down and across “word” has the given sum. No digit can be used more than once in each “word”.

An Example



A Kakuro puzzle

```
go:-
    Vars=[X1,X2,…,X16],
    Vars :: 1..9,
    word([X1,X2],5),
    word([X3,X4,X5,X6],17),
    …
    word([X10,X14],3),
    labeling(Vars),
    writeln(Vars).

word(L,Sum):-
    sum(L) #= Sum,
    all_different(L).
```

Break the Code Down

- `sum(L) #= Sum`
  The sum of the elements in `L` makes `Sum`.

  e.g., `sum([X1,X2,X3]) #= Y` is the same as `X1+X2+X3 #= Y.`

- `all_different(L)`
  Every element in `L` is different.

The Knapsack Problem

- A smuggler has a knapsack of 9 units. He can smuggle in bottles of whiskey of size 4 units, bottles of perfume of size 3 units, and cartons of cigarettes of size 2 units. The profit of smuggling a bottle of whiskey, a bottle of perfume or a carton of cigarettes is 15, 10 and 7, respectively. If the smuggler will only take a trip, how can he take to make the largest profit?

```
go:-
    [W,P,C] :: 0..9,
    4*W+3*P+2*C #=< 9,
    maxof(labeling([W,P,C]),15*W+10*P+7*C),
    writeln([W,P,C]).
```

Break the Code Down

- `maxof(Goal,Exp)`
  Find an instance of `Goal` that is true and maximizes `Exp`.

Exercises

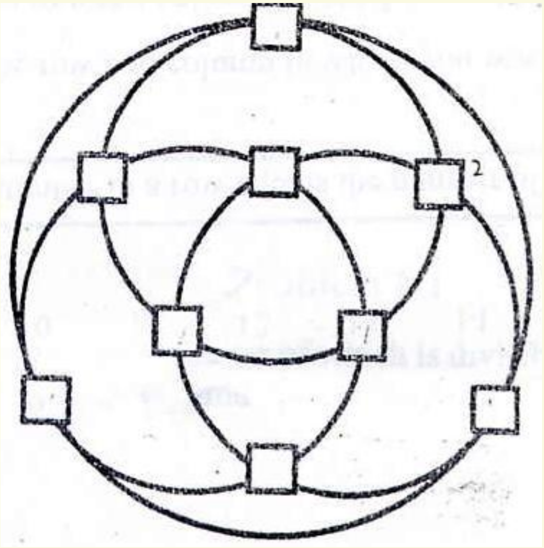
1. Tickets to a carnival cost 250 JPY for students and 400 JPY for adults. If a group buys 10 tickets for a total of 3100 JPY, how many of the tickets are for students?
2. The product of the ages, in years, of three teenagers is 4590. None of the teens are the same age. What are the ages of the teenagers?
3. Suppose that you have 100 pennies, 100 nickels, and 100 dimes. Using at least one coin of each type, select 21 coins that have a total value of exactly $1.00. How many of each type did you select?

Exercises (Cont.)

4. If m and n are positive integers, neither of which is divisible by 10, and if mn = 10,000, find the sum m+n.
5. The arithmetic cryptographic puzzle: Find distinct digits for S, E, N, D, M, O, R, Y such that S and M are non-zero and the equation SEND+MORE=MONEY is satisfied.
6. A magic square of order 3x3 is an arrangement of integers from 1 to 9 such that all rows, all columns, and both diagonals have the same sum.

Exercises (Cont.)

7. Place the numbers 2,3,4,5,6,7,8,9,10 in the boxes so that the sum of the numbers in the boxes of each of the four circles is 27.
8. Sudoku puzzle.



8	6	7			5	9	1	
1				7			8	5
	3							
			7	6	2	1		
	8			9			6	
		2	8	1	4			
							3	
9	1			3				6
	4	3	1			8	2	9

Exercises (Cont.)

9. A factory has four workers *w1*,*w2*,*w3*,*w4* and four products *p1*,*p2*,*p3*,*p4*. The problem is to assign workers to products so that each worker is assigned to one product, each product is assigned to one worker, and the profit maximized. The profit made by each worker working on each product is given in the matrix.

Profit matrix is:

	$p1$	$p2$	$p3$	$p4$
$w1$	7	1	3	4
$w2$	8	2	5	1
$w3$	4	3	7	2
$w4$	3	1	6	3

Review of CLP(FD)

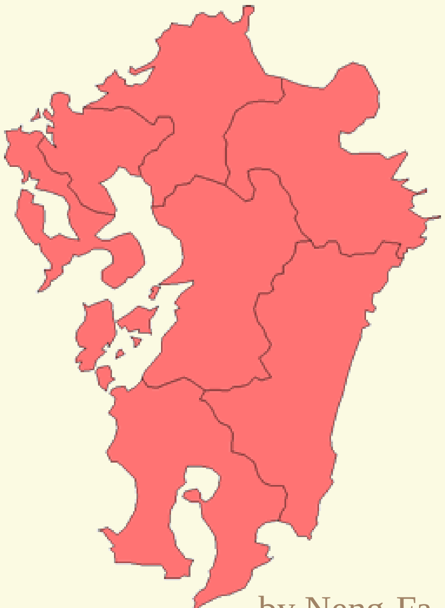
- Declaration of domain variables
  - X :: L..U
  - [X1,X2,...,Xn] :: L..U
- Constraints
  - Exp R Exp (
    - R is one of the following: #=, #¥=, #>, #>=, #<, #=<
    - Exp may contain +, -, *, /, //, mod, sum, min, max
  - all_different(L)
- Labeling
  - labeling(L)
  - minof(labeling(L),Exp) and maxof(labeling(L),Exp)

CLP(FD) by Example (II)

- The graph coloring problem
- The N-queens problem
- The magic square problem
- Exercises

Graph Coloring

- Given a graph G=(V,E) and a set of colors, assign a color to each vertex in V so that no two adjacent vertices share the same color.



The map of Kyushu

Fukuoka
Kagoshima
Kumamoto
Miyazaki
Nagasaki
Oita
Saga

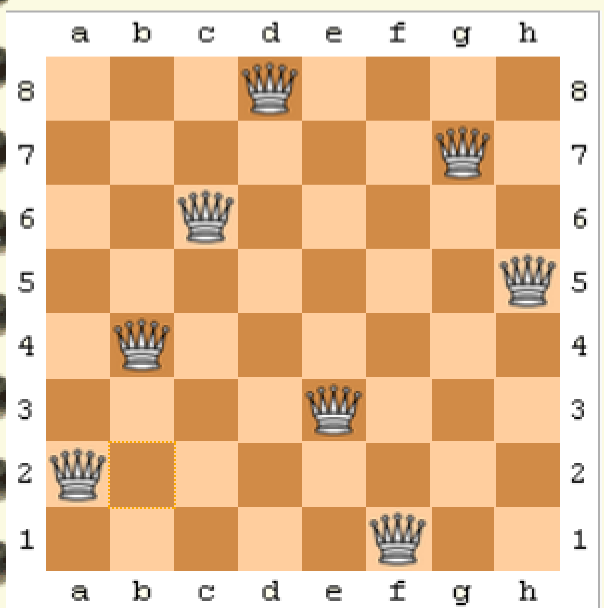
Color the Map of Kyushu

```
go:-
      Vars=[Cf,Cka,Cku,Cm,Cn,Co,Cs],
      Vars :: [red,blue,purple],
      Cf #¥= Cs,
      Cf #¥= Co,
      …
      labeling(Vars),
      writeln(Vars).
```

- Atoms
  - red, blue, purple

The N-Queens Problem

- Find a layout for the N queens on an NxN chessboard so that no queens attack each other. Two queens attack each other if they are placed in the same row, the same column, or the same diagonal.



Qi: the number of the row for the i^{th} queen.

for each two different variables Qi and Qj

```
    Qi  #¥= Qj                    %not same row
    abs(Qi-Qj) #¥= abs(i-j)  %not same diagonal
```

The N-Queens Problem (Cont.)

http://probp.com/examples/foreach/queens.pl

```
queens(N):-
    length(Qs,N),
    Qs :: 1..N,
    foreach(I in 1..N-1, J in I+1..N,
            (Qs[I] #¥= Qs[J],
             abs(Qs[I]-Qs[J]) #¥= J-I)),
    labeling_ff(Qs),
    writeln(Qs).
```

Break the Code Down

- length(L,N)

```
?-length([a,b,c],N)
N = 3
?-length(L,3)
L = [_310,_318,_320]
```

- foreach(I1 in D1,…,In in Dn,Goal)

```
?-L=[a,b,c],foreach(E in L, writeln(E))
```

- Array access notation A[I1,…,In]

Break the Code Down

- `labeling_ff(L)`
  - Label the variables in L by selecting first a variable with the smallest domain. If there are multiple variables with the same domain size, then choose the left-most one (First-fail principle).

Magic Square

- A magic square of order NxN is an arrangement of integers from 1 to N^2 such that all rows, all columns, and both principal diagonals have the same sum

X_{11}	X_{12}	…	X_{1n}
	…		
X_{n1}	X_{n2}	…	X_{nn}

$$\forall_{i=1\ldots n} \sum_{j=1}^{n} X_{ij} = Sum$$
$$\forall_{j=1\ldots n} \sum_{i=1}^{n} X_{ij} = Sum$$
$$\sum_{i=1}^{n} X_{ii} = Sum$$
$$\sum_{i=1}^{n} X_{i(n-i+1)} = Sum$$

Magic Square (Cont.)

http://probp.com/examples/foreach/magic.pl

```
go(N):-
    new_array(Board,[N,N]),
    NN is N*N,
    Vars @= [Board[I,J] : I in 1..N, J in 1..N],
    Vars :: 1..NN,
    Sum is NN*(NN+1)//(2*N),
    foreach(I in 1..N,
            sum([Board[I,J] : J in 1..N]) #= Sum),
    foreach(J in 1..N,
            sum([Board[I,J] : I in 1..N]) #= Sum),
    sum([Board[I,I] : I in 1..N]) #= Sum,
    sum([Board[I,N-I+1] : I in 1..N]) #= Sum,
    all_different(Vars),
    labeling([ffc],Vars),
    writeln(Board).
```

Break the Code Down

- List comprehension

  $[T : E_1 \text{ in } D_1, \ldots, E_n \text{ in } D_n, Goal]$

  – Calls to @=/2

```
?- L @= [X : X in 1..5].
L=[1,2,3,4,5]

?-L @= [(A,I): A in [a,b], I in 1..2].
L= [(a,1),(a,2),(b,1),(b,2)]
```

  – Arithmetic constraints

```
sum([A[I,J] : I in 1..N, J in 1..N]) #= N*N
```

Exercises

1. Write a CLP(FD) program to test if the map of Japan is 3-colorable (can be colored with three colors).
2. Write a program in your favorite language to generate a CLP(FD) program for solving the magic square problem.

Exercises (Cont.)

3. Find an integer programming problem and convert it into CLP(FD).
4. Find a constraint satisfaction or optimization problem and write a CLP(FD) program to solve it.

CLP(Boolean): A Special Case of CLP(FD)

```
<BooleanExpression> ::=
     0 |                                                       /* false */
     1 |                                                       /* true */
     <Variable> |
     <Expression> #= <Expression> |
     <Expression> #\= <Expression> |
     <Expression> #> <Expression> |
     <Expression> #>= <Expression> |
     <Expression> #< <Expression> |
     <Expression> #=< <Expression> |
     #\ <BooleanExpression> |                                   /* not */
     <BooleanExpression> #/\ <BooleanExpression> |              /* and */
     <BooleanExpression> #\/ <BooleanExpression> |              /* or */
     <BooleanExpression> #=> <BooleanExpression> |              /* imply */
     <BooleanExpression> #<=> <BooleanExpression> |             /* equivalent */
     <BooleanExpression> #\ <BooleanExpression>                 /* xor */
```

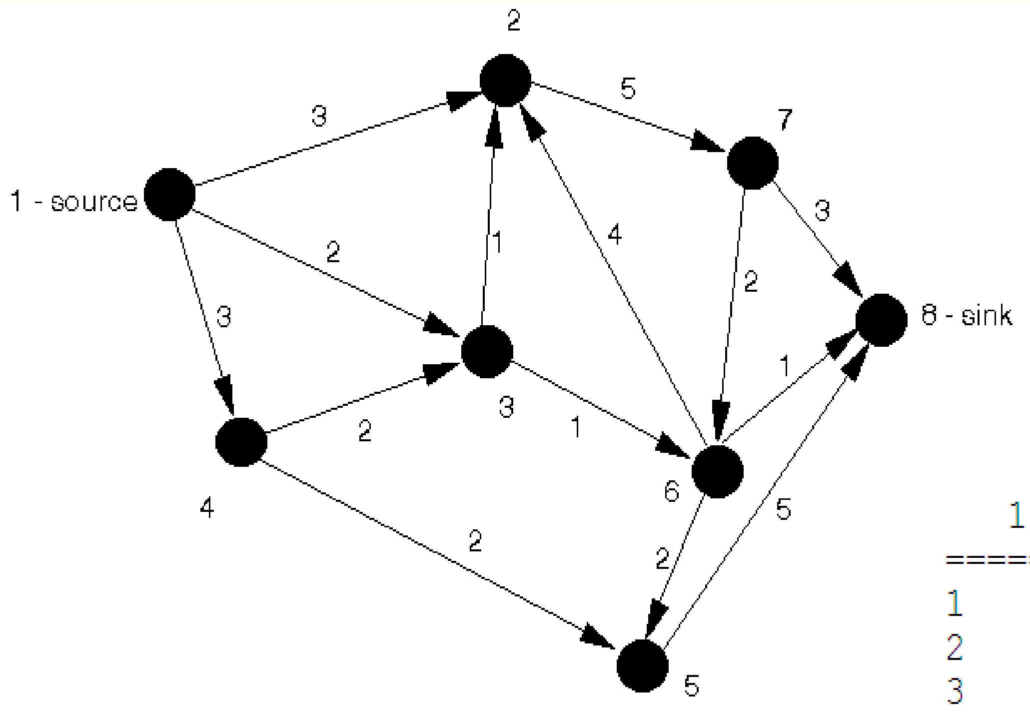
CLP(FD) by Example (III)

- Maximum flow
- Scheduling
- Traveling salesman problem (TSP)
- Planning
- Routing
- Protein structure predication

Maximum Flow Problem

- Given a network G=(N,A) where N is a set of nodes and A is a set of arcs. Each arc (i,j) in A has a capacity Cij which limits the amount of flow that can be sent throw it. Find the maximum flow that can be sent between a single source and a single sink.

Maximum Flow Problem (Cont.)



Capacity matrix

	1	2	3	4	5	6	7	8
1		3	2	3				
2							5	
3		1				1		
4			2		2			
5								5
6		4			2			1
7						2		3
8								

Maximum Flow Problem (Cont.)

```
go:-
    Vars=[X12,X13,X14,X27,X32,X36,X43,
          X45,X58,X62,X65,X68,X76,X78],
    X12 :: 0..3, X13 :: 0..2, X14 :: 0..3,
    X27 :: 0..5, X32 :: 0..1, X36 :: 0..1,
    X43 :: 0..2, X45 :: 0..2, X58 :: 0..5,
    X62 :: 0..4, X65 :: 0..5, X68 :: 0..1,
    X76 :: 0..2, X78 :: 0..3,
    X12+X32+X62-X27 #= 0,
    X13+X43-X32-X36 #= 0,
    X14-X43-X45 #= 0,
    X45+X65-X58 #= 0,
    X36+X76-X62-X65-X68 #= 0,
    X27-X76-X78 #= 0,
    Max #= X58+X68+X78,
    maxof(labeling(Vars),Max),
    writeln(sol(Vars,Max)).
```

Other Network Problems

- Routing
  - Find routes from sources and sinks in a graph
- Upgrading
  - Upgrade nodes in a network to meet certain performance requirement with the minimum cost
- Tomography
  - Determine the paths for probing packages

Scheduling Problem

- Four roommates are subscribing to four newspapers. The following gives the amounts of time each person spend on each newspaper:

Person/Newspaper/Minutes

Person	Asahi	Nishi	Orient	Sankei
Akiko	60	30	2	5
Bobby	75	3	15	10
Cho	5	15	10	30
Dola	90	1	1	1

Akiko gets up at 7:00, Bobby gets up at 7:15, Cho gets up at 7:15, and Dola gets up at 8:00. Nobody can read more than one newspaper at a time and at any time a newspaper can be read by only one person. Schedule the newspapers such that the four persons finish the newspapers at an earliest possible time.

Scheduling Problem (Cont.)

- Variables
  - For each activity, a variable is used to represent the start time and another variable is used to represent the end time.
    - A_Asahi : The start time for Akiko to read Asahi
    - EA_Asahi: The time when Akiko finishes reading Asahi
- Constraints
  - A_Asahi #>= 7*60 : Akiko gets up at 7:00
  - Nobody can read more than one newspaper at a time
  - A newspaper can be read by only one person at a time
- The objective function
  - Minimize the maximum end time

Scheduling Problem (Cont.)

```
go:-
    Vars = [A_Asahi,A_Nishi,A_Orient,A_Sankei,…],
    A_Asahi #>= 7*60, A_Nishi #>= 7*60, …
    B_Asahi #>=7*60+15, B_Nishi #>= 7*60+15, …
    …
    cumulative([A_Asahi,A_Nishi,A_Orient,A_Sankei],
               [60,30,2,5],[1,1,1,1],1),
    …
    EA_Asahi #= A_Asahi+60, EA_Nishi #= A_Nishi+30,
    …
    max([EA_Asahi,EA_Nishi,…]) #= Max,
    minof(labeling(Vars),Max),
    writeln(Vars).
```

Break the Code Down

- `cumulative(Starts,Durations,Resources,Limit)`

  Let `Starts` be [S1,S2,...,Sn], `Durations` be [D1,D2,...,Dn] and `Resources` be [R1,R2,...,Rn]. For each job i, Si represents the start time, Di the duration, and Ri the units of resources needed. `Limit` is the units of resources available at any time.

  The jobs are mutually disjoint when `Resources` is [1,…,1] and `Limit` is 1.

  Si #>= Sj+Dj #¥/ Sj #>= Si+Di (for i,j=1..n, i≠ j)

Traveling Salesman Problem

- Given an undirected graph G=(V,E), where V is the set of nodes and E the set of edges, each of which is associated with a positive integer indicating the distance between the two nodes, find a shortest possible Hamiltonian cycle that connects all the nodes.

Traveling Salesman Problem (Cont.)

```
go:-
    max_node_num(N), % Nodes are numbered 1,2, …, N
    length(Vars,N),
    decl_domains(Vars,1),
    circuit(Vars),
    findall(edge(X,Y,W),edge(X,Y,W),Edges),
    collect_weights(Edges,Vars,Weights),
    TotalWeight #= sum(Weights),
    minof(labeling_ff(Vars),TotalWeight,writeln((Vars,TotalWeight))).

decl_domains([],_).
decl_domains([Var|Vars],X):-
    findall(Y,edge(X,Y,_),Ys),
    Var :: Ys,
    X1 is X+1,
    decl_domains(Vars,X1).

collect_weights([],_,[]).
collect_weights([edge(X,Y,W)|Es],Vars,[B*W|Ws]):-
    nth(X,Vars,NX),
    nth(Y,Vars,NY),
    B #<=> (NX#=Y #¥/ NY#=X),
    collect_weights(Es,Vars,Ws).
```

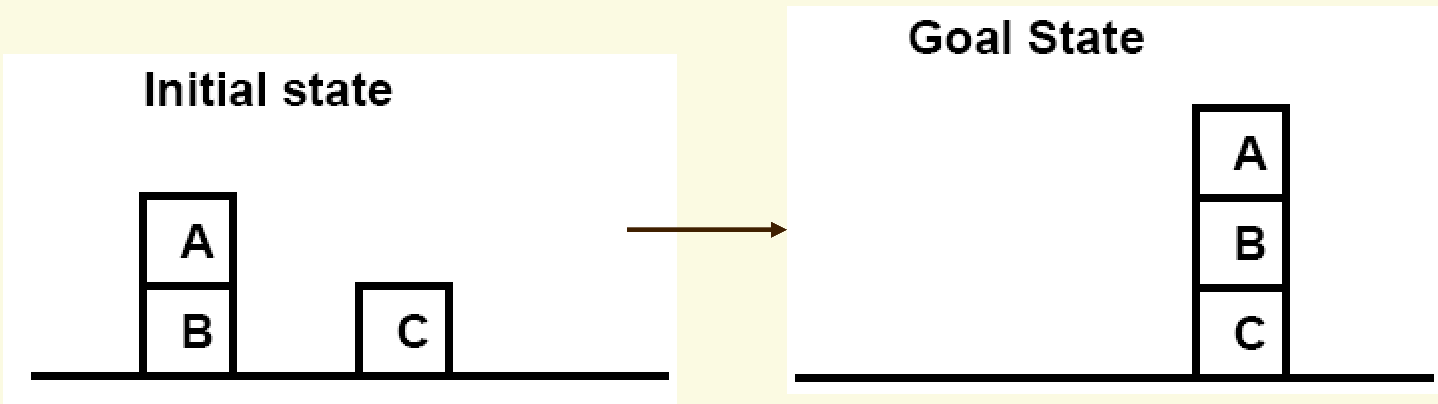
Break the Code Down

- `circuit(L)`
  Let `L=[X1,X2,…,Xn]`. A valuation satisfies the constraint if `1->X1,2->X2,…, n->Xn` forms a Hamilton cycle.
- `minof(Goal,Obj,Report)`
  Call `Report` each time a solution is found.
- Reification constraints

```
B #<=> (NX#=Y #¥/ NY#=X),
```

Planning

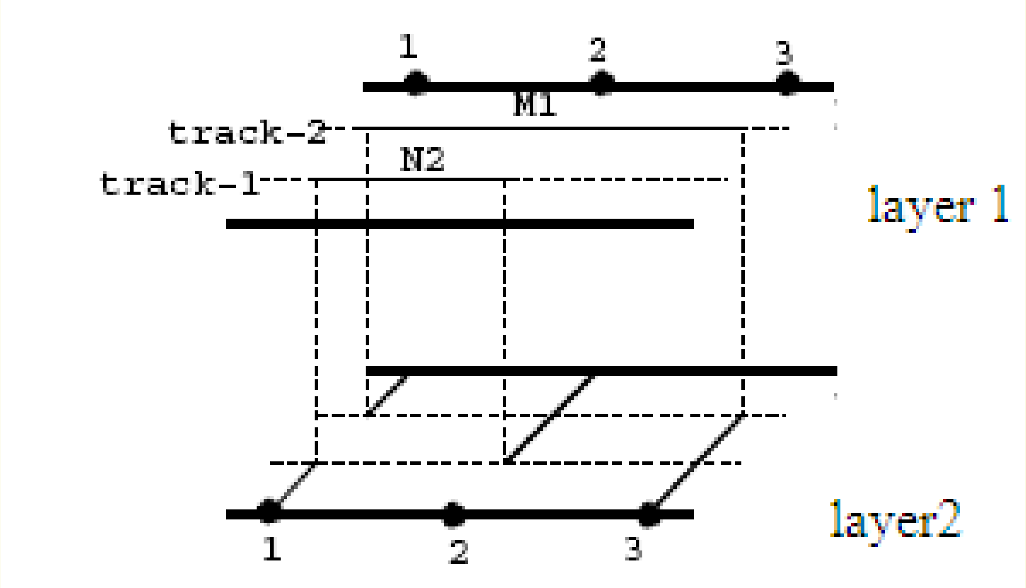
- Blocks world problem



Planning (Cont.)

- States and variables (m blocks and n states)
  $S_1\ S_2\ \ldots\ S_n$
  $S_i=(B_{i1},B_{i2},\ldots,B_{im})$
  $B_{ij} = k$ (block j is on top of block k,
  block 0 means the table)
- Constraints
  - Every transition $S_i \rightarrow S_{i+1}$ must be valid.

Channel Routing

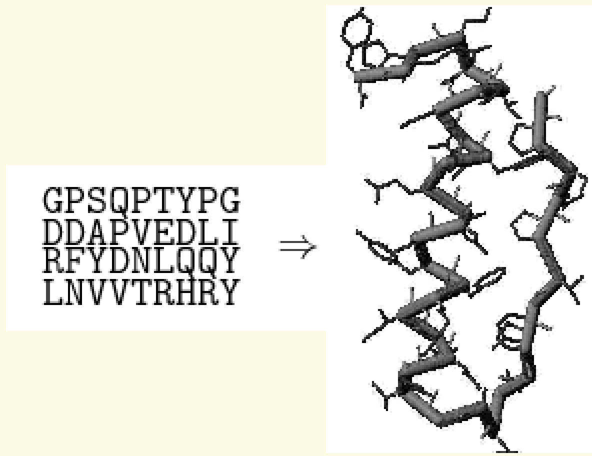


N1={t(1),b(3)}
N2={b(1),t(2)}

Channel Routing (Cont.)

- Variables
  - For each net, use two variables L and T to represent the layer and track respectively
- Constraints
  - No two line segments can overlap
- Objective functions
  - Minimize the length (or areas) of wires

Protein Structure Predication

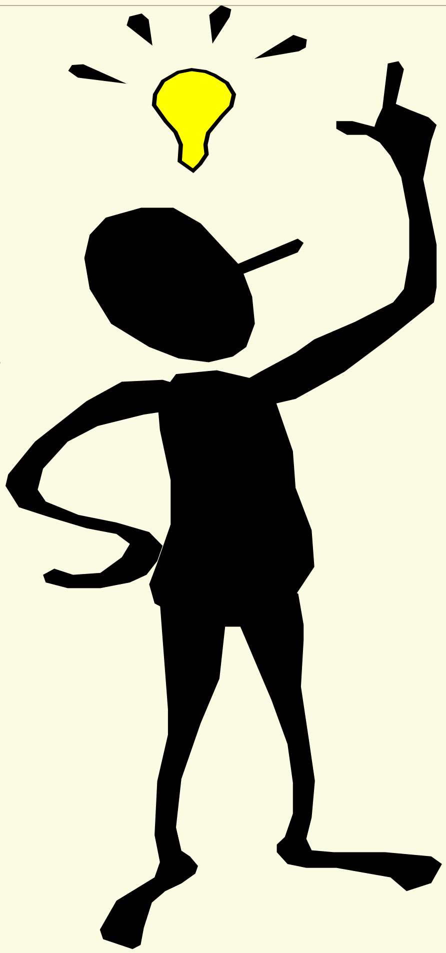


Protein Structure Predication (Cont.)

- Variables
  - Let $R=r_1,\ldots,r_n$ be a sequence of residues. A structure of R is represented by a sequence of points in a three-dimensional space $p_1,\ldots,p_n$ where $p_i=<x_i,y_i,z_i>$.
- Constraints
  - A structure forms a self-avoiding walk in the space
- The objective function
  - The energy is minimized

Demo

- B-Prolog version 7.4
  - CLP(FD)+ CGLIB
  - www.probp.com/examples.htm



Constraint Systems

- CLP systems
  - B-Prolog
  - BNR-Prolog
  - CHIP
  - CLP(R)
  - ECLiPSe - CISCO
  - GNU-Prolog
  - IF/Prolog
  - Prolog-IV
  - SICStus
- Other systems
  - 2LP
  - ILOG solver
  - OPL
  - Oz
  - Gcode
  - Choco
- More information
  - Languages & compilers
  - Logic programming
  - Constraint programming

Major References

- B-Prolog virtual machine
  - N.F. Zhou: Parameter Passing and Control Stack Management in Prolog Implementation Revisited, *ACM TOPLAS*, 1996.
  - N.F. Zhou: The Language Features and Architecture of B-Prolog, *TPLP special issue*, 2011.
- Action rules and constraint solving
  - N.F. Zhou: Programming Finite-Domain Constraint Propagators in Action Rules, *TPLP*, 2006.
  - N.F. Zhou: Encoding Table Constraints in CLP(FD) Based on Pair-wise AC, *ICLP*, 2009.
- Tabling
  - N.F. Zhou: T. Sato and Y.D. Shen: Linear Tabling Strategies and Optimizations, *TPLP*, 2008.
  - N.F. Zhou: Y. Kameya and T. Sato, Mode-directed Tabling for …, *Tools for Artificial Intelligence*, 2010. (submitted)