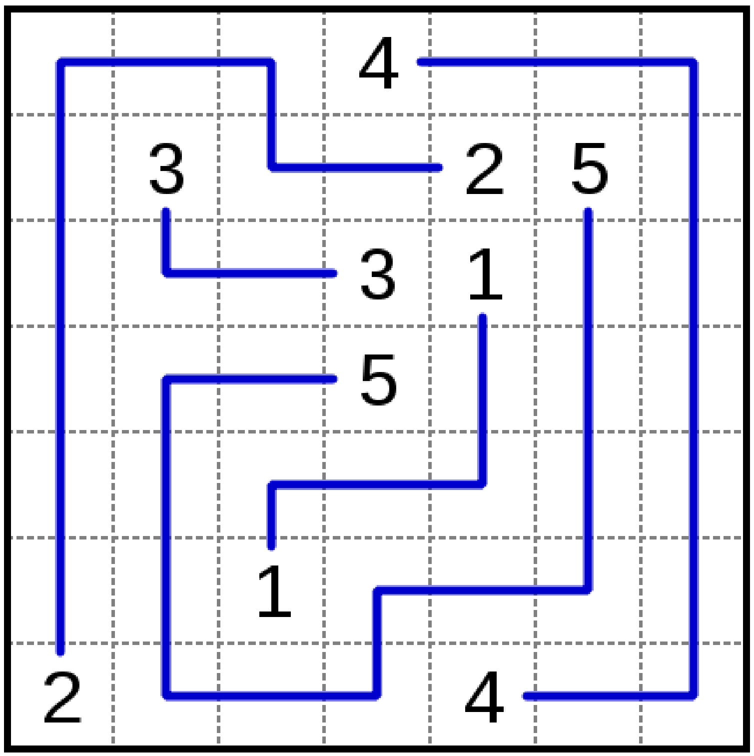
Modeling and Solving AI Problems in Picat

Roman Barták, Neng-Fa Zhou



Pair up all the matching numbers on the grid with single continuous lines (or paths).

- The **lines cannot branch off** or **cross** over each other, and
- the numbers have to fall at the end of each line (i.e., not in the middle).

It is considered all the cells in the grid are filled.

Source: wiki

Numberlink: a hard instance

Solved with the sat module of Picat and the Lingeling solver in 40s.

Numberlink: Picat encoding

```
1 1 1 0 0 0 0
1 0 1 1 1 0 0
1 0 0 0 0 0 0
1 0 0 0 0 0 0
1 0 0 0 0 0 0
1 0 0 0 0 0 0
1 0 0 0 0 0 0
```

```
{{0,0,0,4,0,0,0},
 {0,3,0,0,2,5,0},
 {0,0,0,3,1,0,0},
 {0,0,0,5,0,0,0},
 {0,0,0,0,0,0,0},
 {0,0,1,0,0,0,0},
 {2,0,0,0,4,0,0}}
```

```
import sat.

numberlink(NP,NR,NC,InputM) =>
     M = new_array(NP,NR,NC),
     M :: 0..1,
     % no two numbers occupy the same square
    foreach(J in 1..NR, K in 1..NC)
        sum([M[I,J,K] : I in 1..NP]) #=1
    end,
     % connectivity constraints
    foreach(I in 1..NP, J in 1..NR, K in 1..NC)
        Neibs = [M[I,J1,K1] : (J1,K1) in [(J-1,K),(J+1,K),(J,K-1),(J,K+1)],
                              J1>=1, K1>=1, J1=<NR, K1=<NC],
        (InputM[J,K]==I ->
            M[I,J,K] #=1, sum(Neibs) #= 1
        ;
            M[I,J,K] #=> sum(Neibs) #= 2
        )
    end,
    solve(M).
```

Introduction to Picat

- *Picat's programming constructs*
- *Logic programming*
- *Functional programming*
- *Dynamic programming*

Combinatorial (optimization) problems in Picat

- *A very short introduction to SAT, CP, MIP modules*
- *Sudoku*
- *Golomb ruler*
- *Multi-agent path finding*

Wrap up

Part I:

INTRODUCTION TO PICAT

What is Picat?

Why the name "PICAT"?

- Pattern-matching, Intuitive, Constraints, Actors, Tabling

Core logic programming concepts:

- logic variables (arrays and maps are terms)
- implicit pattern-matching and explicit unification
- explicit non-determinism

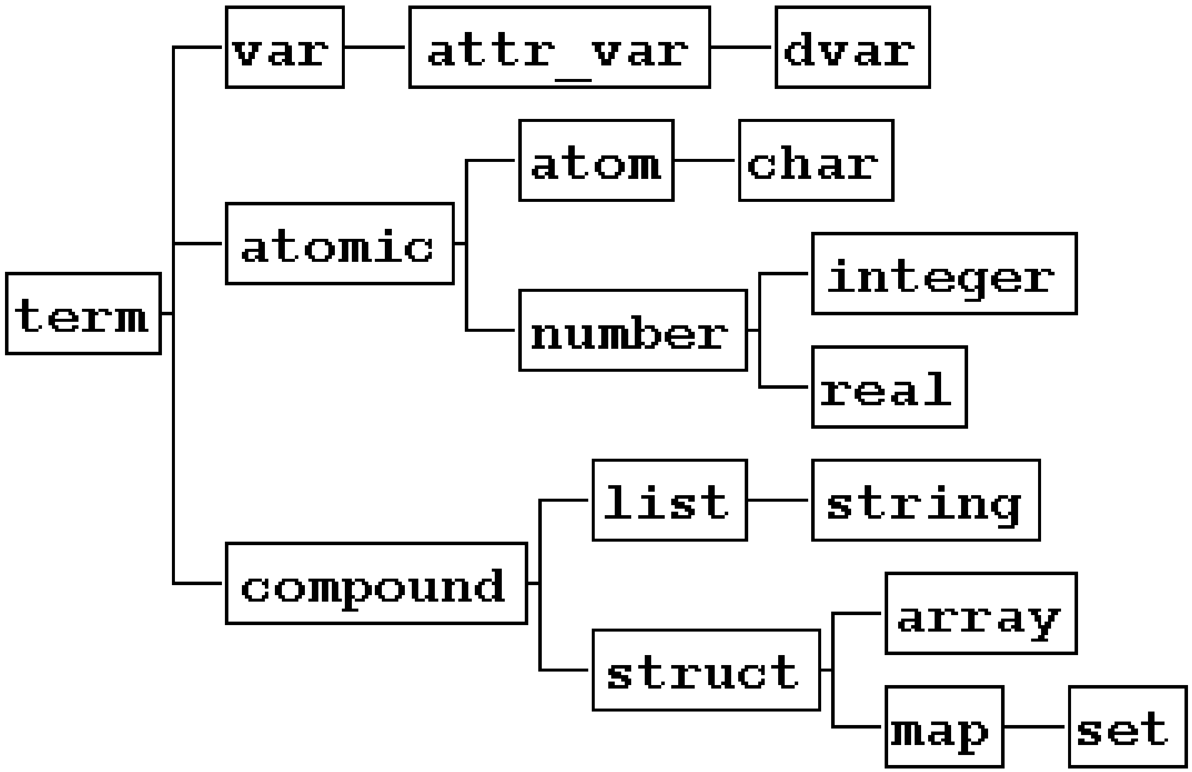
Language constructs for scripting and modeling:

- functions, loops, list and array comprehensions, and assignments

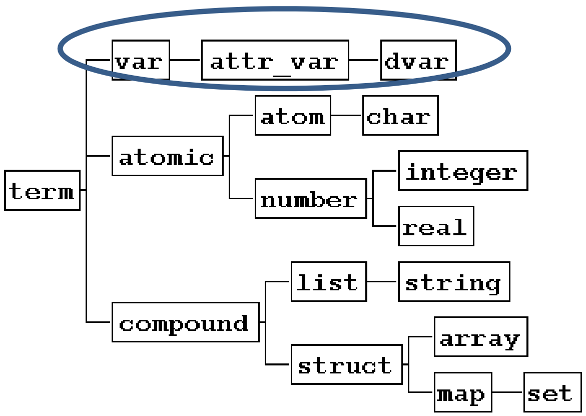
Facilities for combinatorial search:

- tabling for dynamic programming
- the `cp`, `sat`, and `mip` modules for CSPs
- the `planner` module for planning

Picat's Data Types

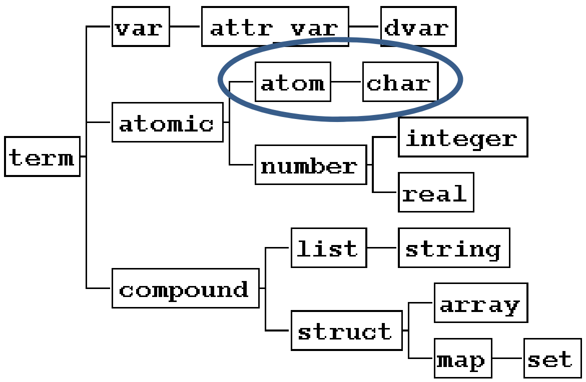


A variable name begins with a capital letter or the underscore.



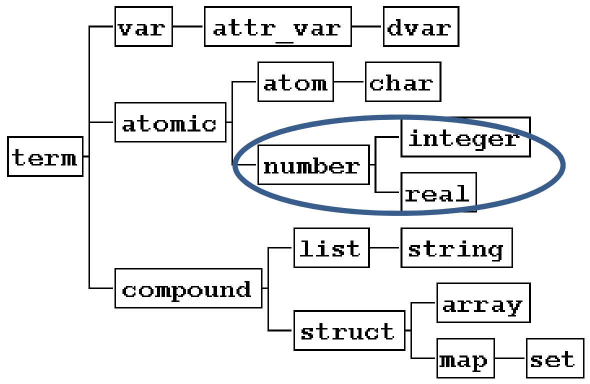

```
Picat> var(X)
yes

Picat> X = a, var(X)
no

Picat> X.put_attr(a,1), attr_var(X)
yes

Picat> X.put_attr(a,1), Val = X.get_attr(a)
Val = 1
yes

Picat> import cp
Picat> X :: 1..10, dvar(X)
X = DV_010b48_1..10
yes
```

An unquoted atom name begins with a lower-case letter.
A character is a single-letter atom.



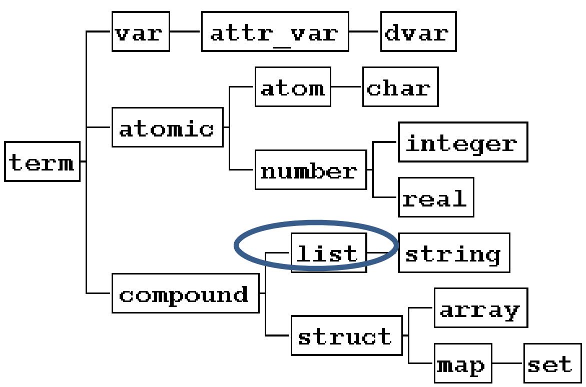

```
Picat> atom(abc)
yes

Picat> atom('_abc')
yes

Picat> char(a)
yes

Picat> Code = ord(a)
Code = 97

Picat> A = chr(97)
A = a
```

Numbers



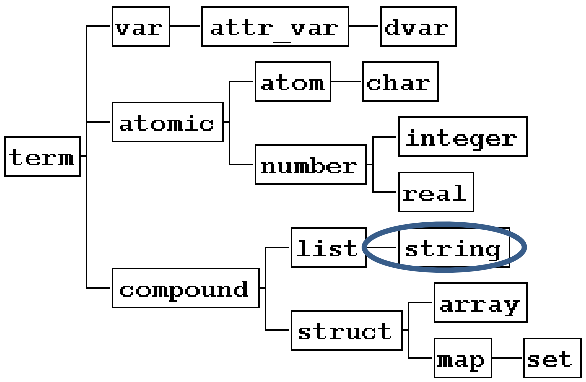

```
Picat> int(123)
yes

Picat> Big = 999999999999999999999
Big = 999999999999999999999

Picat> X = 0b111101
X = 61

Picat> X = 0xff0
X = 4080

Picat> real(1.23)
yes

Picat> X = 1.23e10
X = 12300000000.0
```

Lists are singly-linked lists.



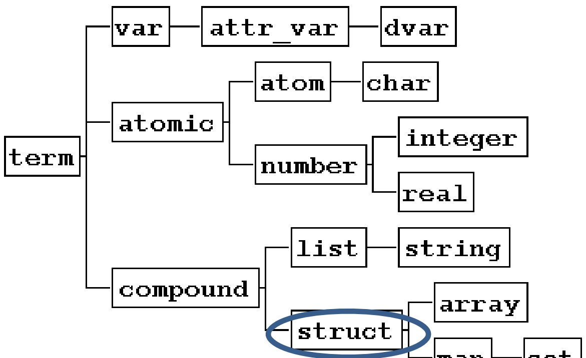

```
Picat> L = [a,b,c], list(L)
L = [a,b,c]
yes

Picat> L = new_list(3)
L = [_101c8,_101d8,_101e8]

Picat> L = 1..2..10
L = [1,3,5,7,9]

Picat> L = [X : X in 1..10, even(X)]
L = [2,4,6,8,10]

Picat> L = [a,b,c], Len = len(L)
L = [a,b,c]
Len = 3

Picat> L = [a,b] ++ [c,d]
L = [a,b,c,d]
```

Strings are lists of characters.



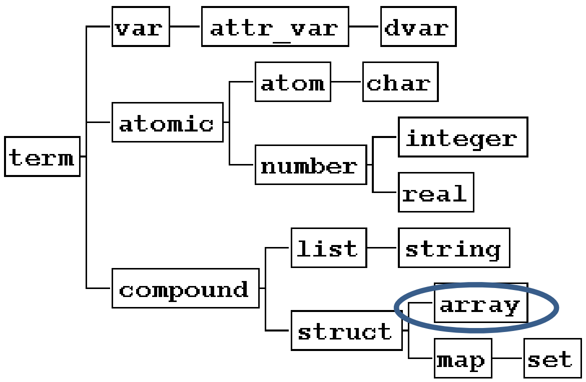

```
Picat> S = "hello"
S = [h,e,l,l,o]

Picat> S = "hello" ++ "Picat"
S = [h,e,l,l,o,'P',i,c,a,t]

Picat> S = to_string(abc)
S = [a,b,c]

Picat> S = to_radix_string(123,16)
S = ['7','B']

Picat> X = to_int("123")
X = 123

Picat> X = parse_term("[1,2,3]")
X = [1,2,3]
```

Structures



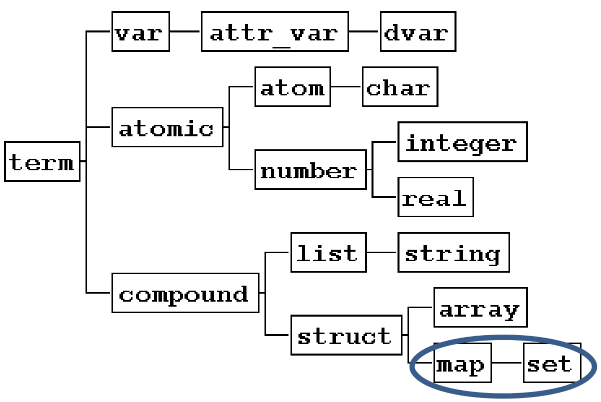

```
Picat> S = $student(mary,cs,3.8)
S = student(mary,cs,3.8)

Picat> S = new_struct(mary,3)
S = mary(_12ad0,_12ad8,_12ae0)

Picat> S = $f(a), A = arity(S), N = name(S)
A = 1
N = f

Picat> And = (a,b)
And = (a,b)

Picat> Or = (a;b)
Or = (a;b)

Picat> Constr = (X #= Y)
Constr = (_10f18 #= _10f20)
```

Arrays




```
Picat> A = {a,b,c}, array(A)
A = {a,b,c}
yes

Picat> A = new_array(3)
A = {_10528,_10530,_10538}

Picat> A = new_array(3,3)
A = {{_fdb0,_fdb8,_fdc0},…}

Picat> A = {X : X in 1..10, even(X)}
A = {2,4,6,8,10}

Picat> L = [a,b,c], A = to_array(L)
L = [a,b,c]
A = {a,b,c}

Picat> A = {a,b} ++ {c,d}
A = {a,b,c,d}
```

Maps and sets are hash tables.




```
Picat> M = new_map([ichi=1, ni=2]), map(M)
M = (map)[ni = 2,ichi = 1]
yes

Picat> M = new_map([ni=2]), Ni = M.get(ni)
Ni = 2

Picat> M = new_map(), M.put(ni,2)
M = (map)[ni = 2]

Picat> M = new_map(), Ni = M.get(ni,unknown)
M = (map)[]
Ni = unknown

Picat> S = new_set([a,b,c])
S = (map)[c,b,a]

Picat> S = new_set([a,b,c]), S.has_key(b)
yes
```

X[I1,…,In] : X references a compound value

Linear-time access of **list** elements.

```
Picat> L = [a,b,c,d], X = L[4]
X = d
```

Constant-time access of **structure** and **array** elements.

```
Picat> S = $student(mary,cs,3.8), GPA = S[3]
GPA = 3.8
```

```
Picat> A = {{1, 2, 3}, {4, 5, 6}}, B = A[2, 3]
B = 6
```

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

```
Picat> L = [X : X in 1..10, even(X)]
L = [2,4,6,8,10]

Picat> L = [(A,I) : A in [a,b], I in 1..2].
L = [(a,1),(a,2),(b,1),(b,2)]

Picat> L = [(A,I) : {A,I} in zip([a,b],1..2)]
L = [(a,1),(b,2)]

Picat> L = [X : I in 1..5]                 % X is local
L = [_bee8,_bef0,_bef8,_bf00,_bf08]

Picat> X = _, L = [X : I in 1..5]     % X is non-local
L = [X,X,X,X,X]
```

OOP Notation

O.f(t1,…,tn)
-- means module qualified call if O is atom
-- means $f(O,t_1,…,t_n)$ otherwise.

```
Picat> Y = 13.to_binary_string()
Y = [‘1’, ‘1’, ‘0’, ‘1’]

Picat> Y = 13.to_binary_string().reverse()
Y = [‘1’, ‘0’, ‘1’, ‘1’]

% X becomes an attributed variable
Picat> X.put_attr(age, 35), X.put_attr(weight, 205), A =
   X.get_attr(age)
A = 35

% X is a map
Picat> X = new_map([age=35, weight=205]), X.put(gender, male)
X = (map)([age=35, weight=205, gender=male])

Picat> S = $point(1.0, 2.0), Name = S.name, Arity = S.len
Name = point
Arity = 2

Picat> Pi = math.pi       % module qualifier
Pi = 3.14159
```

The `foreach` Loop

`foreach`(E_1 `in` D_1, Cond$_1$,…, E_n `in` D_n, Cond$_n$)
Goal
end

Variables that occur within a loop but not before in its outer scope are local to each iteration

```
Picat> A = new_array(5), foreach(I in 1..5) A[I] = X end
A = {_15bd0,_15bd8,_15be0,_15be8,_15bf0}

Picat> X = _, A = new_array(5), foreach(I in 1..5) A[I] = X end
A = {X,X,X,X,X}
```

Non-backtrackable

`Head, Cond => Body.`

Backtrackable

`Head, Cond ?=> Body.`

```
member(X,L) ?=> L = [X|_].
member(X,L) => L = [_|LR], member(X,LR).

membchk(X,[X|_] => true.
membchk(X,[_|L]) => membchk(X,L).
```

- Pattern-matching rules
  - No laziness or freeze
    The call `membchk(X,_)` fails
  - Facilitates indexing
- Explicit unification
- Explicit non-determinism

```
Head = Exp, Cond => Body.
```

```
fib(0) = 1.
fib(1) = 1.
fib(N) = fib(N-1)+fib(N-2).

power_set([]) = [[]].
power_set([H|T]) = P1++P2 =>
    P1 = power_set(T),
    P2 = [[H|S] : S in P1].

qsort([]) = [].
qsort([H|T]) = qsort([E : E in T, E=<H])++
               [H]++
               qsort([E : E in T, E>H]).
```

Dynamically typed

List and array comprehensions

Strict (not lazy)

Higher-order functions

Function calls cannot occur in head patterns.

Index notations, ranges, dot notations, and comprehensions cannot occur in head patterns.

As-patterns:

```
merge([],Ys) = Ys.
merge(Xs,[]) = Xs.
merge([X|Xs],Ys@[Y|_]) = [X|Zs], X<Y =>
    Zs = merge(Xs,Ys).
merge(Xs,[Y|Ys]) = [Y|Zs] =>
    Zs=merge(Xs,Ys).
```

```
main =>
    print("enter an integer:"),
    N = read_int(),
    foreach(I in 0..N)
        Num := 1,
        printf("%*s", N-I, ""),    % print N-I spaces
        foreach(K in 0..I)
            printf("%d ", Num),
            Num := Num*(I-K) div (K+1)
        end,
        nl
    end.
```

SSA (Static Single Assignment)

Loops

```
$ picat pascal
enter an integer:5
     1
    1 1
   1 2 1
  1 3 3 1
 1 4 6 4 1
1 5 10 10 5 1
```

Dynamic Programming in Picat

```
table
fib(0) = 0.
fib(1) = 1.
fib(N) = fib(N-1)+fib(N-2).
```

- Linear tabling
- Mode-directed tabling
- Term sharing

Dynamic Programming: Binomial Coefficient

$$\binom{n}{0} = \binom{n}{n} = 1 \quad \text{for all integers } n \geq 0,$$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \quad \text{for all integers } n, k : 1 \leq k \leq n-1,$$

```
table
c(_, 0) = 1.
c(N, N) = 1.
c(N,K) = c(N-1, K-1) + c(N-1, K).
```

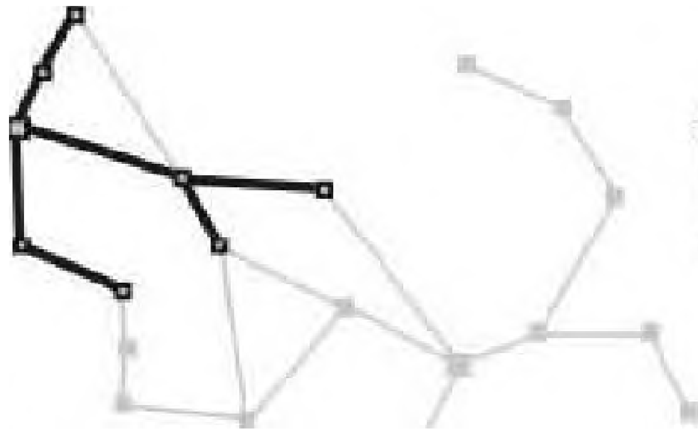
- Tabled parser

```
% E -> E + T | E - T | T
table
ex(Si,So) ?=>
    ex(Si,S1),
    S1 = ['+'|S2],
    term(S2,So).
ex(Si,So) ?=>
    ex(Si,S1),
    S1 = ['-'|S2],
    term(S2,So).
ex(Si,So) =>
    term(Si,So).
```

- Non-tabled parser

```
% E -> T E'
ex(Si,So) =>
    term(Si,S1),
    ex_prime(S1,So).

% E' -> + T E' | - T E' | ε
ex_prime(['+'|Si],So) =>
    term(Si,S1),
    ex_prime(S1,So).
ex_prime(['-'|Si],So) =>
    term(Si,S1),
    ex_prime(S1,So).
ex_prime(Si,So) => So = Si.
```

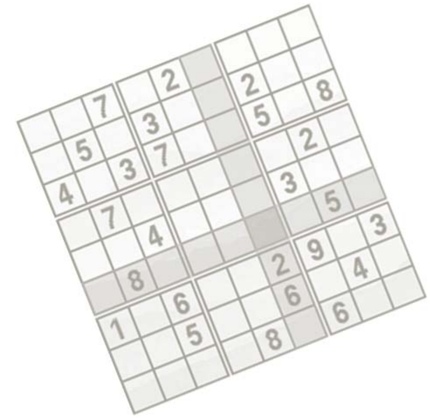
Framework by Pereira and Warren, 1980.



```
table (+,-,min)
path(S,Path,Cost),final(S) =>
    Path=[],Cost=0.
path(S,Path,Cost) =>
    action(S,S1,Action,ActionCost),
    path(S1,Path1,Cost1),
    Path = [Action|Path1],
    Cost = Cost1+ActionCost.
```

```
import planner.

go =>
    S0=[s,s,s,s],
    best_plan(S0,Plan),
    writeln(Plan).

final([n,n,n,n]) => true.

action([F,F,G,C],S1,Action,Cost) ?=>
    Action=farmer_wolf,
    Cost = 1,
    opposite(F,F1),
    S1=[F1,F1,G,C],
    not unsafe(S1).
…
```

- Based on tabling
- Allows use of structures to represent states
- Supports domain knowledge and heuristics
- Provides search predicates
  - Depth-unbounded & depth-bounded
  - IDA & branch-and-bound



Part II.

COMBINATORIAL (OPTIMIZATION) PROBLEMS IN PICAT

`import cp.` **`import sat.`** **`import mip.`**

Constraints:

Domain

```
X :: Domain, X notin Domain
```

Arithmetic

```
(X #= Y), (X #!= Y), (X #> Y), (X #>= Y), …
```

Boolean

```
(X #/\ Y),  (X #\/ Y), (X #<=> Y), (X #=> Y), (X #^ Y),  (#~ X)
```

Table

```
table_in(VarTuple,Tuples),  table_notin(VarTuple,Tuples)
```

Global

```
all_different(L), element(I,L,V), circuit(L), cumulative(…), …
```

Solver invocation:

```
solve(Options,Vars)
```

Example: Send More Money

```
   S E N D          9 5 6 7
 + M O R E   =>   + 1 0 8 5
 ---------        ---------
 M O N E Y        1 0 6 5 2
```

```
import cp.  ——>  import sat.

send_more_money =>
    Vars = [S,E,N,D,M,O,R,Y],
    Vars :: 0..9,
    all_different(Vars),
    S #!= 0,
    M #!= 0,
     1000*S+100*E+10*N+D
    +1000*M+100*O+10*R+E
    #= 10000*M+1000*O+100*N+10*E+Y,
    solve(Vars),                     % label variables
    writeln(Vars).
```

Common interface to CP, SAT, and MIP

Combinatorial puzzle, whose goal is to enter digits 1-9 in cells of 9×9 table in such a way, that no digit appears twice or more in every row, column, and 3×3 sub-grid.

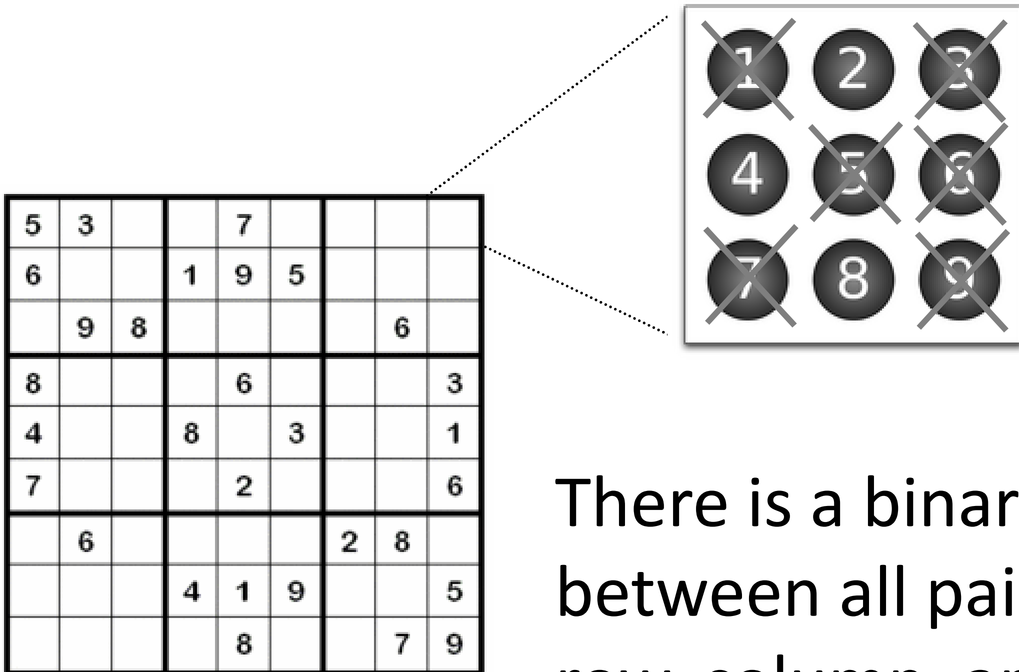
9	6	3	1	7	4	2	5	8
1	7	8	3	2	5	6	4	9
2	5	4	6	8	9	7	3	1
8	2	1	4	3	7	5	9	6
4	9	6	8	5	2	3	1	7
7	3	5	9	6	1	8	2	4
5	8	9	7	1	3	4	6	2
3	1	7	2	4	6	9	8	5
6	4	2	5	9	8	1	7	3

Solving Sudoku

×	×	6		1	3			
3	9	×					1	
2	1	8				4		

Use information that each digit appears exactly once in each row, column and sub-grid.

Sudoku - a model idea



We can see every cell as a **variable** with possible values from **domain** {1,…,9}.

There is a binary inequality **constraint** between all pairs of variables in every row, column, and sub-grid.

Such formulation of the problem is called a **constraint satisfaction problem.**

```
import cp.

sudoku(Board) =>
   N = Board.length,
   N1 = ceiling(sqrt(N)),
   Board :: 1..N,
   foreach(R in 1..N)
       all_different([Board[R,C] :
                              C in 1..N])
   end,
   foreach(C in 1
       all_differ
   end,
   foreach(R in 1
       all_differ

   end,
   solve(Board).
```

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

```
board(Board) =>
  Board = {{_, 6, _, 1, _, 4, _, 5, _},
           {_, _, 8, 3, _, 5, 6, _, _},
           {2, _, _, _, _, _, _, _, 1},
           {8, _, _, 4, _, 7, _, _, 6},
           {_, _, 6, _, _, _, 3, _, _},
           {7, _, _, 9, _, 1, _, _, 4},
           {5, _, _, _, _, _, _, _, 2},
           {_, _, 7, 2, _, 6, 9, _, _},
           {_, 4, _, 5, _, 8, _, 7, _}}.
```

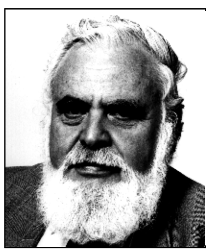
RB1

A **ruler with M marks** such that **distances** between any two marks are **different**.

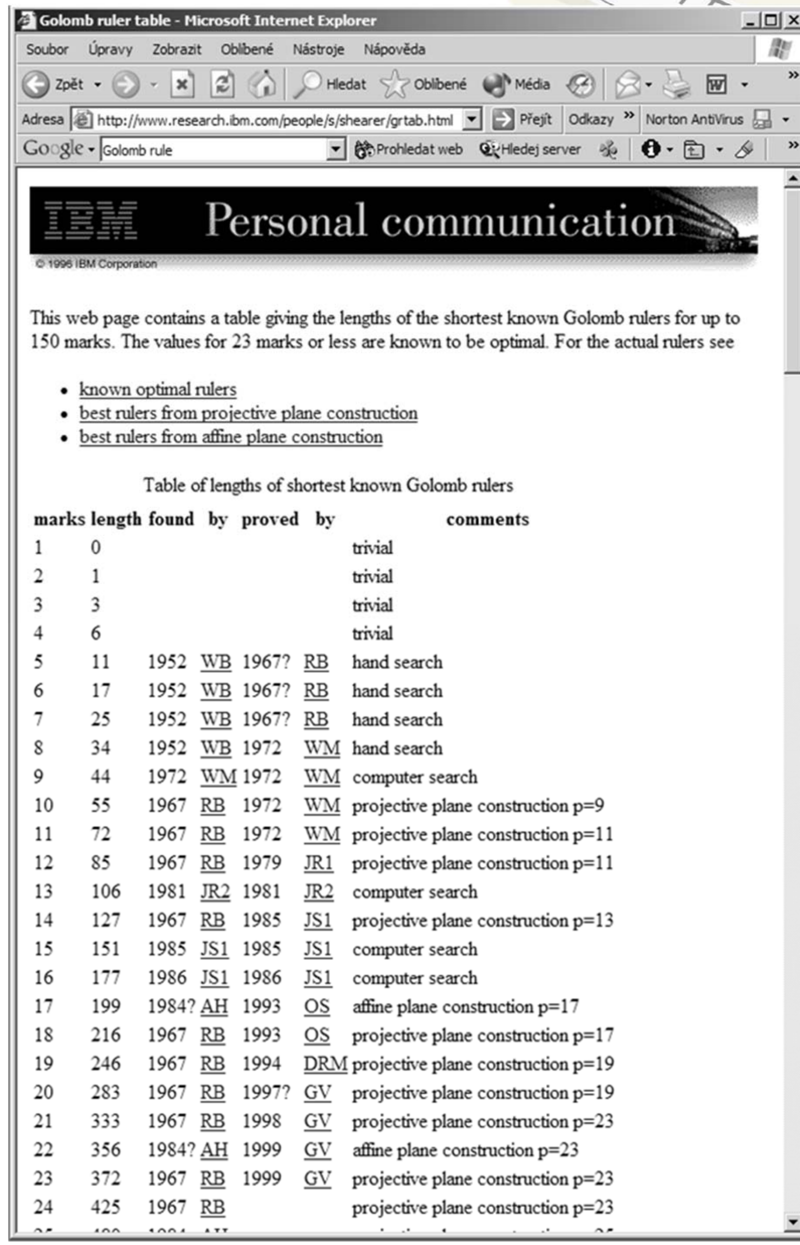
The **shortest ruler** is the optimal ruler.

Hard for M≥16, no exact algorithm for M ≥ 24!

Applied in **radioastronomy**.



Solomon W. Golomb
Professor
University of Southern California
http://csi.usc.edu/faculty/golomb.html



Golomb ruler table - Microsoft Internet Explorer

Soubor Úpravy Zobrazit Oblíbené Nástroje Nápověda

Adresa http://www.research.ibm.com/people/s/shearer/grtab.html

IBM Personal communication

© 1996 IBM Corporation

This web page contains a table giving the lengths of the shortest known Golomb rulers for up to 150 marks. The values for 23 marks or less are known to be optimal. For the actual rulers see

- known optimal rulers
- best rulers from projective plane construction
- best rulers from affine plane construction

Table of lengths of shortest known Golomb rulers

marks	length	found	by	proved	by	comments
1	0					trivial
2	1					trivial
3	3					trivial
4	6					trivial
5	11	1952	WB	1967?	RB	hand search
6	17	1952	WB	1967?	RB	hand search
7	25	1952	WB	1967?	RB	hand search
8	34	1952	WB	1972	WM	hand search
9	44	1972	WM	1972	WM	computer search
10	55	1967	RB	1972	WM	projective plane construction p=9
11	72	1967	RB	1972	WM	projective plane construction p=11
12	85	1967	RB	1979	JR1	projective plane construction p=11
13	106	1981	JR2	1981	JR2	computer search
14	127	1967	RB	1985	JS1	projective plane construction p=13
15	151	1985	JS1	1985	JS1	computer search
16	177	1986	JS1	1986	JS1	computer search
17	199	1984?	AH	1993	OS	affine plane construction p=17
18	216	1967	RB	1993	OS	projective plane construction p=17
19	246	1967	RB	1994	DRM	projective plane construction p=19
20	283	1967	RB	1997?	GV	projective plane construction p=19
21	333	1967	RB	1998	GV	projective plane construction p=23
22	356	1984?	AH	1999	GV	affine plane construction p=23
23	372	1967	RB	1999	GV	projective plane construction p=23
24	425	1967	RB			projective plane construction p=23

Slide 36

RB1 distributed.net vyřešili optimáně problémy 24-27 pomocí masivního paralelismu

Roman Barták, 10/15/2016

Golomb ruler – abstract model

A base model:

Variables $X_1, \ldots, X_M$ with the domain 0..M*M

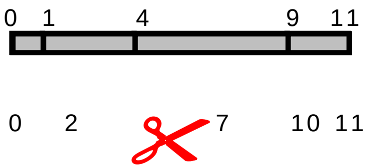
$\mathbf{X_1 = 0}$ ***ruler start***

$\mathbf{X_1 < X_2 < \ldots < X_M}$ ***no permutations of variables***

$\forall \mathbf{i<j\ D_{i,j} = X_j - X_i}$ ***difference variables***

all_different$(\{\mathbf{D_{1,2}, D_{1,3}, \ldots D_{1,M}, D_{2,3}, \ldots D_{M-1,M}}\})$

Model extensions:



$\mathbf{D_{1,2} < D_{M-1,M}}$ ***symmetry breaking***

better bounds (**implied constraints**) for $D_{i,j}$

$D_{i,j} = D_{i,i+1} + D_{i+1,i+2} + \ldots + D_{j-1,j}$

so $\mathbf{D_{i,j} \geq \Sigma_{j-i} = (j-i)*(j-i+1)/2}$ ***lower bound***

$X_M = X_M - X_1 = D_{1,M} = D_{1,2} + D_{2,3} + \ldots D_{i-1,i} + D_{i,j} + D_{j,j+1} + \ldots + D_{M-1,M}$

$D_{i,j} = X_M - (D_{1,2} + \ldots D_{i-1,i} + D_{j,j+1} + \ldots + D_{M-1,M})$

so $\mathbf{D_{i,j} \leq X_M - (M-1-j+i)*(M-j+i)/2}$ ***upper bound***

```
import cp.
golomb(M,X) =>
   X = new_list(M),
   X :: 0..(M*M),                               % domains for marks
   X[1] = 0,

   foreach(I in 1..(M-1))
      X[I] #< X[I+1]                            % no permutaions
   end,

   D = new_array(M,M),                          % distances
   foreach(I in 1..(M-1),J in (I+1)..M)
      D[I,J] #= X[J] - X[I],
      D[I,J] #>= (J-I)*(J-I+1)/2,               % bounds
      D[I,J] #=< X[M] - (M-1-J+I)*(M-J+I)/2
   end,

   D[1,2] #< D[M-1,M],                          % symmetry breaking
   all_different([D[I,J] : I in 1..(M-1),
                           J in (I+1)..M]),
   solve($[min(X[M])],X).
```

Golomb ruler - some results

What is the effect of different constraint models?

size	base model	base model + symmetry	base model + symmetry + implied constraints
7	12	7	4
8	94	44	21
9	860	353	143
10	7 494	3 212	1 091
11	147 748	57 573	23 851

time in milliseconds on 1,7 GHz Intel Core i7, Picat 1.9#6

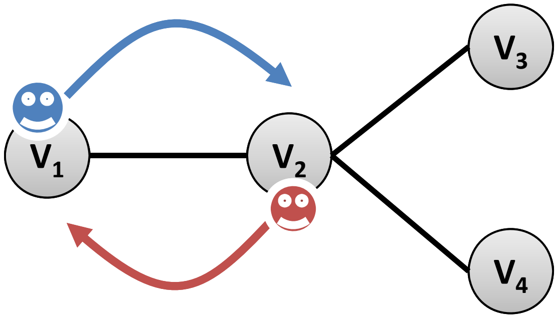
What is the effect of different search strategies?

size	fail first		leftmost first	
	enum	*split*	*enum*	*split*
7	9	9	5	4
8	67	68	23	21
9	537	537	170	143
10	4 834	4 721	1 217	1 091
11	134 071	132 046	26 981	23 851

time in milliseconds on 1,7 GHz Intel Core i7, Picat 1.9#6

Setting:

- a **graph** (directed or undirected)
- a set of **agents**, each agent is assigned to two locations (nodes) in the graph (start, destination)



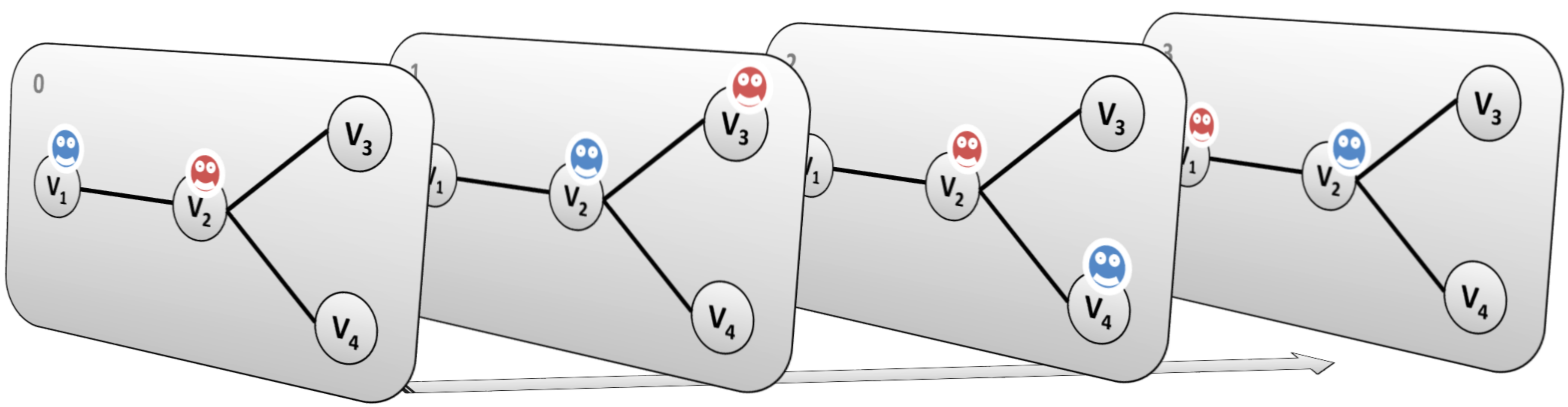
MAPF problem:

Find a **collision-free** plan (path) for each agent.

In MAPF, we do not know the lengths of plans (due to possible re-visits of nodes)!

We can encode plans of a known length using a **layered graph** (temporally extended graph).

- Each layer corresponds to one time slice and indicates positions of agents at that time.



MAPF – abstract model

Using **layered graph** describing agent positions at each time step

B_{tav} : agent *a* occupies vertex *v* at time *t*

Constraints:

- each agent occupies exactly one vertex at each time.

$$\Sigma_{v=1}^{n} B_{tav} = 1 \text{ for } t = 0, \ldots, m, \text{ and } a = 1, \ldots, k.$$

- no two agents occupy the same vertex at any time.

$$\Sigma_{a=1}^{k} B_{tav} \leq 1 \text{ for } t = 0, \ldots, m, \text{ and } v = 1, \ldots, n.$$

- if agent *a* occupies vertex *v* at time *t*, then *a* occupies a neighboring vertex or stay at *v* at time *t* + 1.

$$B_{tav} = 1 \Rightarrow \Sigma_{u \in neibs(v)} (B_{(t+1)au}) \geq 1$$

Preprocessing:

B_{tav} = 0 if agent *a* cannot reach vertex *v* at time *t* or
a cannot reach the destination being at *v* at time *t*

```
import sat.

path(N,As) =>
    K = len(As),
    lower_upper_bounds(As,LB,UB),
    between(LB,UB,M),
    B = new_array(M+1,K,N),
    B :: 0..1,

    % Initialize the first and last states
    foreach (A in 1..K)
        (V,FV) = As[A],
        B[1,A,V] = 1,
        B[M+1,A,FV] = 1
    end,

    % Each agent occupies exactly one vertex
    foreach (T in 1..M+1, A in 1..K)
        sum([B[T,A,V] : V in 1..N]) #= 1
    end,

    % No two agents occupy the same vertex
    foreach (T in 1..M+1, V in 1..N)
        sum([B[T,A,V] : A in 1..K]) #=< 1
    end,

    % Every transition is valid
    foreach (T in 1..M, A in 1..K, V in 1..N)
        neibs(V,Neibs),
        B[T,A,V] #=>
        sum([B[T+1,A,U] : U in Neibs]) #>= 1
    end,

    solve(B),
    output_plan(B).
```

Incremental generation of layers

Setting the initial and destination locations

Agent occupies one vertex at any time

No conflict between agents

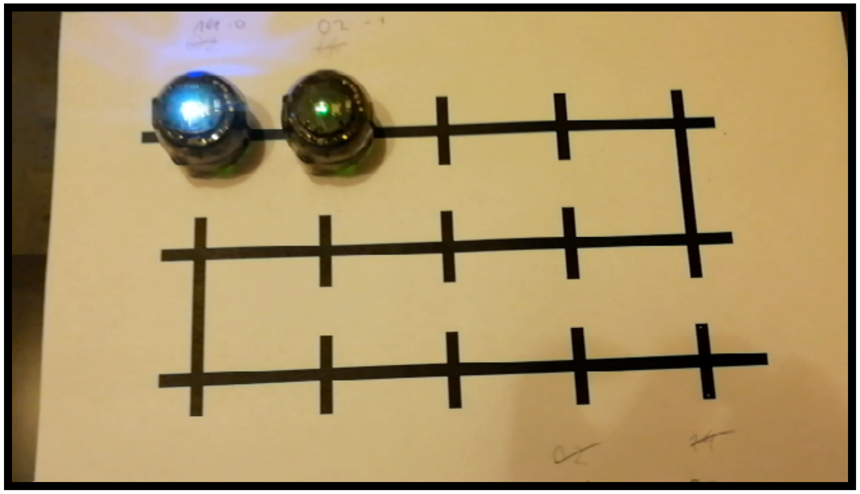
Agent moves to a neighboring vertex

```
foreach(T in 1..M1, A in 1..K, V in 1..N)
    B[T,A,V] #=> sum([B[Prev,A2,V] :
                      A2 in 1..K, A2!=A,
                      Prev in max(1,T-L)..T]) #= 0
end
```

L-robustness

If any agent is delayed then trains may cause collisions during execution.

To prevent such collisions we may introduce more space between agents.



MAPF – some results

Instance	Makespan			Sum of costs		
	Picat	MDD	ASP	Picat	MDD	ICBS
g16_p10_a05	0.27	**0.02**	10.86	5.68	**0.01**	**0.01**
g16_p10_a10	1.37	**0.14**	9.58	35.82	**0.01**	**0.01**
g16_p10_a20	2.76	**0.76**	26.06	143.35	**0.01**	**0.01**
g16_p10_a30	3.11	**0.79**	>600	495.04	0.52	**0.02**
g16_p10_a40	8.25	**4.71**	>600	>600	**107.95**	>600
g16_p20_a05	1.01	**0.16**	5.96	16.2	**0.01**	**0.01**
g16_p20_a10	1.5	**0.31**	18.59	92.16	1.58	**0.16**
g16_p20_a20	2.12	**0.46**	20.71	209.74	0.6	**0.05**
g16_p20_a30	4.37	**1.45**	>600	>600	>600	>600
g16_p20_a40	3.48	**1.15**	>600	>600	>600	>600
g32_p10_a05	1.98	**0.53**	12.93	29.91	**0.01**	**0.01**
g32_p10_a10	3.08	**1.21**	31.34	84.92	**0.01**	**0.01**
g32_p10_a20	8.71	**6.8**	105.47	586.71	0.03	**0.01**
g32_p10_a30	**34.48**	40.13	274.11	>600	0.22	**0.02**
g32_p10_a40	34.95	**24.87**	>600	>600	1.81	**0.34**
g32_p20_a05	5.75	**2.77**	11.99	58.27	**0.01**	**0.01**
g32_p20_a10	2.97	**1.11**	33.22	112.2	0.09	**0.01**
g32_p20_a20	16.93	**13.73**	101.84	>600	2.5	**0.22**
g32_p20_a30	12.98	**4.54**	199.69	>600	1.78	**0.05**
g32_p20_a40	16.51	**8.17**	418.56	>600	3.24	**0.13**
Total solved	20	20	15	12	18	17

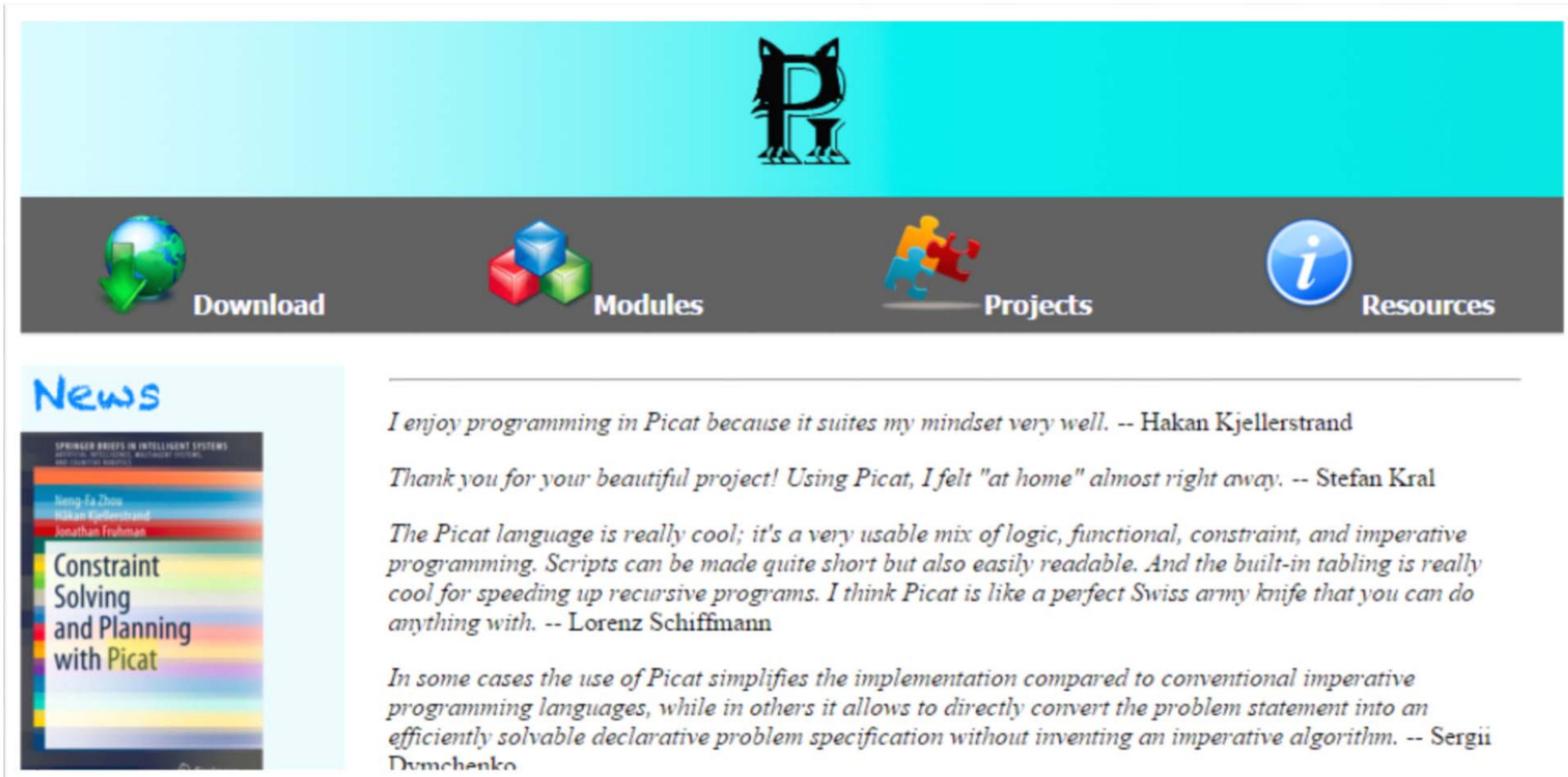
Runtime in seconds

WRAP UP

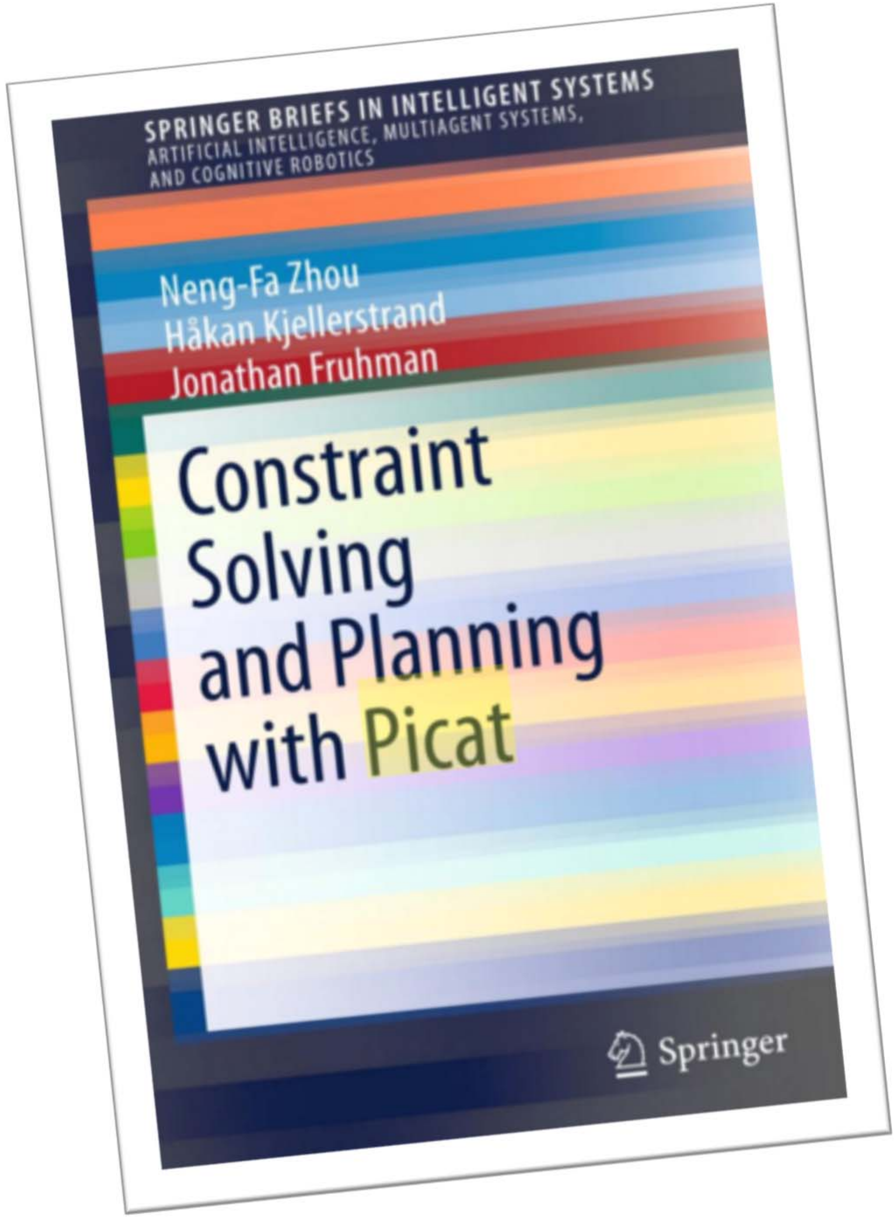
Picat is a logic-based multi-paradigm language that integrates logic programming, functional programming, constraint programming, and scripting.

- logic variables, unification, backtracking, pattern-matching rules, functions, list/array comprehensions, loops, assignments
- tabling for dynamic programming and planning
- **constraint solving** with CP (constraint programming), SAT (satisfiability), and MIP (mixed integer programming).

picat-lang.org



Picat book


SPRINGER BRIEFS IN INTELLIGENT SYSTEMS
ARTIFICIAL INTELLIGENCE, MULTIAGENT SYSTEMS,
AND COGNITIVE ROBOTICS
Neng-Fa Zhou
Håkan Kjellerstrand
Jonathan Fruhman
Constraint Solving and Planning with Picat
Springer

References

1. H. Kjellerstrand: **Picat: A Logic-based Multi-paradigm Language**, ALP Newsletter, 2014.
2. R. Barták and N.-F. Zhou: **Using Tabled Logic Programming to Solve the Petrobras Planning Problem**, TPLP 2014.
3. R. Barták, A. Dovier, and N.-F. Zhou: **On Modeling Planning Problems in Tabled Logic Programming**, PPDP 2015.
4. S. Dymchenko and M. Mykhailova: **Declaratively Solving Google Code Jam Problems with Picat**, PADL 2015.
5. S. Dymchenko: **An Introduction to Tabled Logic Programming with Picat,** Linux Journal, August, 2015.
6. N.-F. Zhou: **Combinatorial Search With Picat**, ICLP invited talk, 2014.
7. N.-F. Zhou, R. Barták, and A. Dovier: **Planning as Tabled Logic Programming,** TPLP 2015.
8. N.-F. Zhou, H. Kjellerstrand, and J. Fruhman: **Constraint Solving and Planning with Picat**, Springer, 2015.
9. N.-F. Zhou, H. Kjellerstrand: **The Picat-SAT Compiler**, PADL 2016.
10. N.-F. Zhou, H. Kjellerstrand: **Optimizing SAT Encodings for Arithmetic Constraints**, CP 2017.

Modeling and Solving AI Problems in Picat

Roman Barták, Neng-Fa Zhou