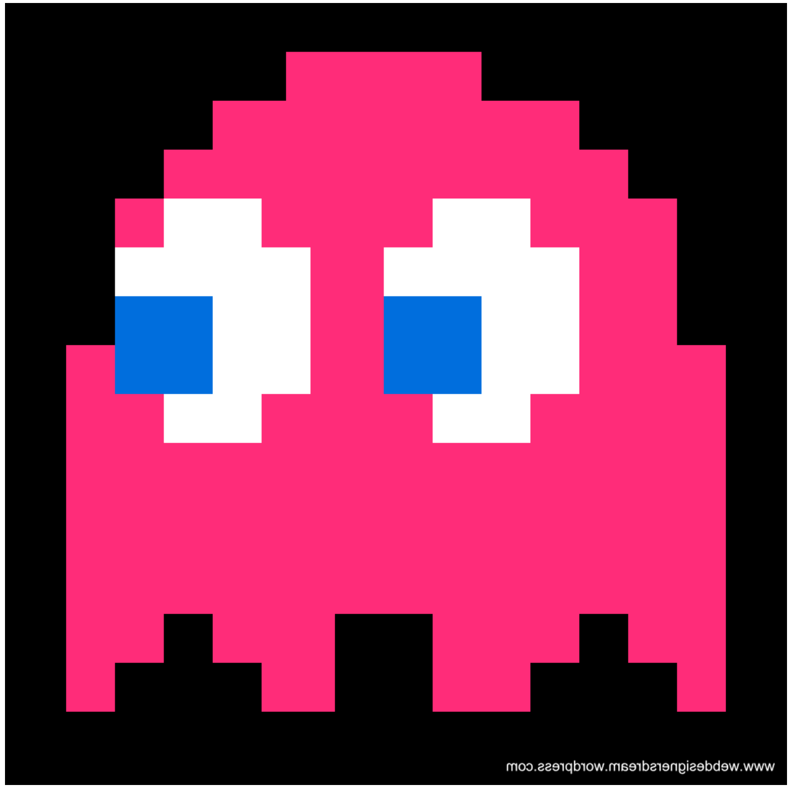
272SM: Artificial Intelligence

Programming with Prolog



Instructor: Tatjana Petrov

University of Trieste, Italy

Today

- Six lessons about problem solving and search with Prolog (https://staff.fnwi.uva.nl/u.endriss/teaching/pss/slides/pss-prolog-slides.pdf)

- Examples:
  - Who is bigger? Who is related? Is Socrates mortal?
  - Lists: length, reversal; Checking if a word is a palindrome
  - Computing a Fibonacci sequence
  - Solving Sudoku, N-Queens

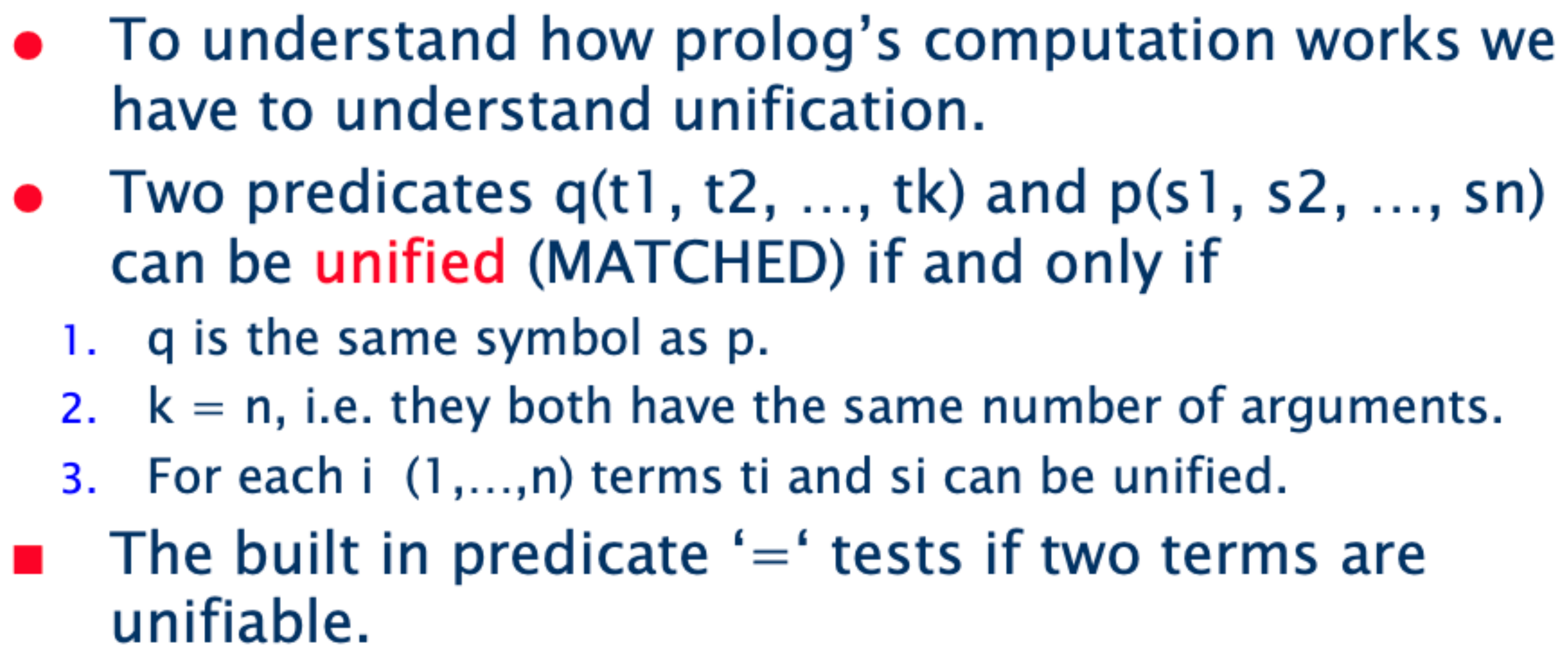
Prolog examples: Who is bigger? (basic syntax)

```
/* Facts (Clauses) */
bigger(elephant, horse).
bigger(horse, donkey).
bigger(donkey, dog).
bigger(donkey, monkey).

/* Rule (Clauses) */
is_bigger(X, Y) :- bigger(X,Y).
is_bigger(X, Y) :- bigger(X,Z), is_bigger(Z,Y).

/* SYNTAX
 * Atoms: elephant, xYZ, a_123, 'How are you today?'
 * Variables: X, Elephant, _G177, MyVariable
 * Compound terms: is_bigger(horse, X), f(g(Alpha, _), 7), 'My Function' (dog)
 * Facts
*/
```

Prolog examples: Family relations (matching)

```
/* Matching using a built-in equality predicate born(mary, yorkshire) = born(mary, X).
 f(a, g(X, Y)) = f(X, Z), Z = g(W, h(X)).
 p(X, 2, 2) = p(1, Y, X).
 p(_, 2, 2) = p(1, Y, _). anonymous variable
*/

mortal(X) :- man(X).
man(socrates).

aunt(X, Z) :-
    sister(X, Y),
    parent(Y, Z).
```

How Prolog works: Unification



- To understand how prolog’s computation works we have to understand unification.
- Two predicates q(t1, t2, ..., tk) and p(s1, s2, ..., sn) can be unified (MATCHED) if and only if
  1. q is the same symbol as p.
  2. k = n, i.e. they both have the same number of arguments.
  3. For each i (1,...,n) terms ti and si can be unified.
- The built in predicate ‘=‘ tests if two terms are unifiable.

How Prolog works: Unification of two terms

- Unifying two terms ti and si
  - If both are constants, then they can be unified only if they are identical.
  - If one is an unbound variable X, then we can unify the two terms by binding X to the other term (i.e., we set X’s value to be the other term).
    - E.g., X = f(a) ➔ yes X is bound to the value f(a).
  - If one or both are bound variables then we have to try to unify the values the variables are bound to
    - E.g. X = f(a), Y = X ➔ first X is bound to the value f(a), then when we try to unify Y with X, X is bound so we must Y with X’s value, so Y also becomes bound to f(a).
  - If ti = f(x1,x2, …, xk) and si = g(y1, y2, …, ym) then ti and si can be unified if and only if
    - f is identical to g.
    - k = m (both functions take the same number of arguments).
    - xi and yi can be recursively unified for all i = 1…m

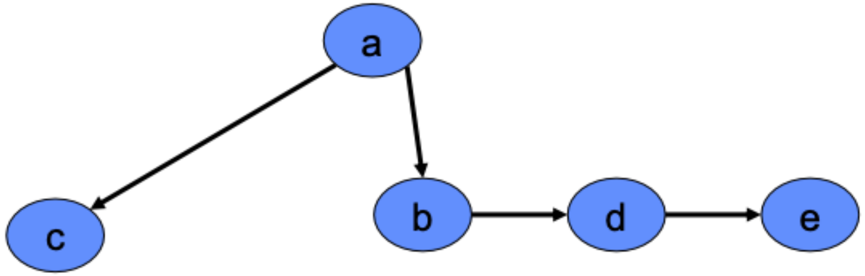
Solving queries

- How Prolog works:
  - Unification
  - Goal–Directed Reasoning
  - Rule–Ordering
  - DFS and backtracking
- When given a query Q = q1, q2, …, qn Prolog performs a search in an attempt to solve this query. The search can be specified as follows

Example

- Route finding in in a directed acyclic graph:

```
edge(a,b).
edge(a,c).
edge(b,d).
edge(d,e).
```



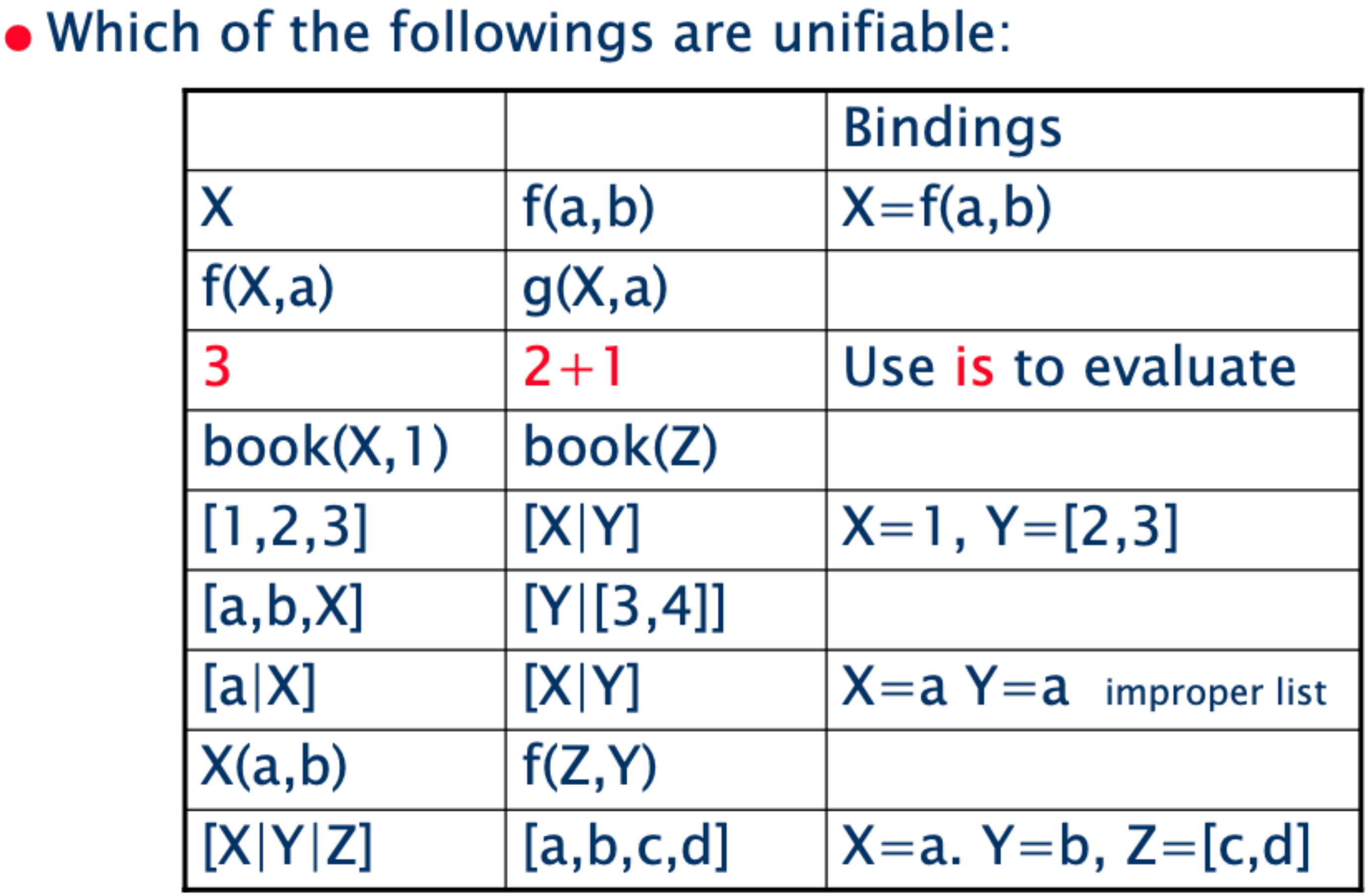
```
path(X,Y) :- path(X,Z), edge(Z,Y).
path(X,Y) :- edge(X,Y).
```

- The above is problematic. Why?
- Here is the correct solution:

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

Hojjat Ghaderi, Fall 2006, University of Toronto

Unification examples



- Which of the followings are unifiable:

		Bindings
X	f(a,b)	X=f(a,b)
f(X,a)	g(X,a)	
3	2+1	Use is to evaluate
book(X,1)	book(Z)	
[1,2,3]	[X|Y]	X=1, Y=[2,3]
[a,b,X]	[Y|[3,4]]	
[a|X]	[X|Y]	X=a Y=a improper list
X(a,b)	f(Z,Y)	
[X|Y|Z]	[a,b,c,d]	X=a. Y=b, Z=[c,d]

Prolog examples: List concatenation

```
/% Lists */
X = [a, b, c].
/* MyList = [1,2,3,4,5], MyList = [1,_|Rest].*/

/* concatenation concat_lists([1, 2, 3, 4], [dog, cow, tiger], L).*/
concat_lists([], List, List).
concat_lists([Elem|List1], List2, [Elem|List3]) :- concat_lists(List1, List2, List3).

length([tiger, donkey, cow, tiger], N).    % built-in operator
member(tiger, [dog, tiger, elephant, horse]).
```

```
concat_lists(Begin, End, [1, 2, 3]).
Begin = []
End = [1, 2, 3] ;
Begin = [1]
End = [2, 3] ;
Begin = [1, 2]
End = [3] ;
Begin = [1, 2, 3]
End = [] ;
No
```

```
show(List) :-
    member(Element, List),
    write(Element),
    write('  ‘), % nl
    false.


show([a,b,c]).
```

```
len([], 0).
len([_ I Tail], N) :-
len(Tail, N1),
N is N1 + 1.
```

```
X is  3+5

Vs.

X = 3+5
```

Fibonacci, factorial

```
fibo(0, 0). fibo(1, 1).
fibo(N, F) :-
        N >= 2, N1 is N - 1, N2 is N - 2,
        fibo(N1, F1), fibo(N2, F2), F is F1 + F2.
```

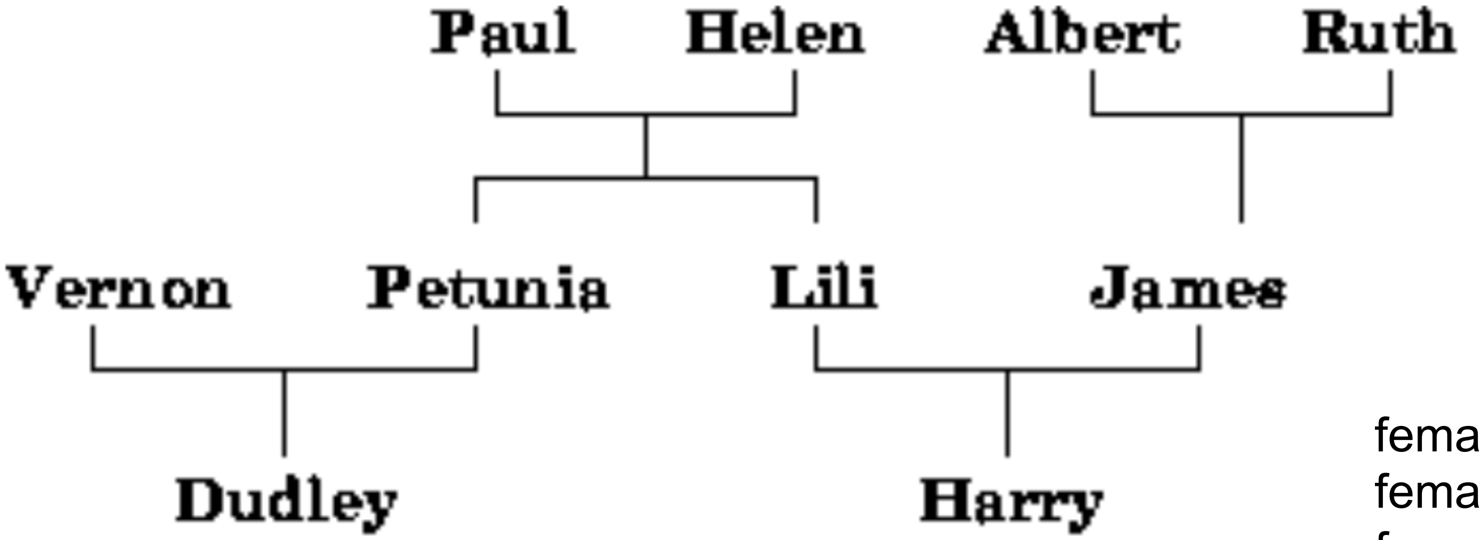
```
fact(0,1). fact(1,1).
fact(N,F) :-
   N>=2, N1 is N-1,
   fact(N1, F1), F is F1*N.
```

```
fibo(0, 0). fibo(1, 1).
fibo(N, F) :-
        N >= 2,
        N1 is N - 1,
        N2 is N - 2,
        fibo(N1, F1),
        fibo(N2, F2),
        F is F1 + F2.
        write(F), nl.
```

What will we get here as output?

(solution: 1-2-1-3-1-2-5, show the tree of calls)

QUIZZ



```
female(helen).
female(ruth).
female(petunia).
female(lili).
male(paul).
male(albert).
male(vernon).
male(james).
male(dudley).
male(harry).
```

Formulate PROLOG rules to capture the relationship of aunt_of(Aunt, Person) :-

Computing the maximum

```
%% Base case: the list has one element.
%% The maximum must be this element as there are no other elements
%% which could be bigger.
max([Max],Max).

%% Two recursive clauses which compute the maximum of the tail and
%% then compare the result to the head.
%% First case: the head is greater than the maximum of the tail. The
%% head is the maximum of the whole list.
max([Head|Tail],Max) :- max(Tail,TailMax),
                        Head > TailMax,
                        Max = Head.

%% Second case: the head is smaller or equal to the maximum of the
%% tail. The maximum of the tail is the maximum of the whole list.
max([Head|Tail],Max) :- max(Tail,TailMax),
                        Head =< TailMax,
                        Max = TailMax.
```

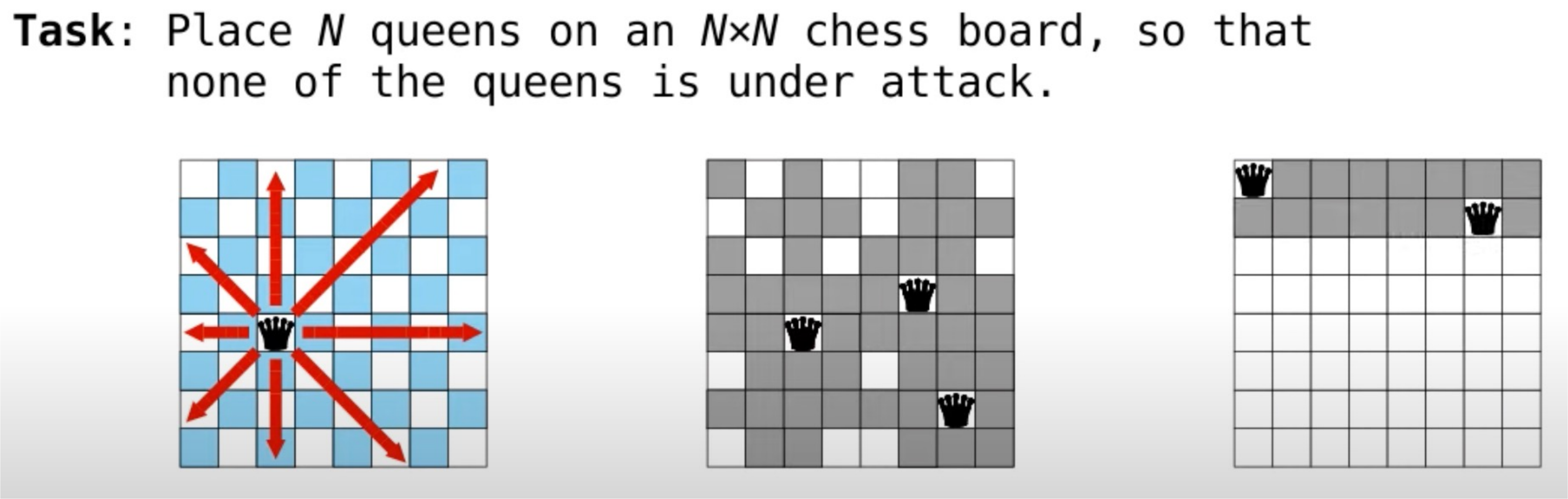
Maze (Computing connections in a graph)

```
connected(1,2).
connected(3,4).
connected(5,6).
connected(7,8).
connected(9,10).
connected(12,13).
connected(13,14).
connected(15,16).
connected(17,18).
connected(19,20).
connected(4,1).
connected(6,3).
connected(4,7).
connected(6,11).
connected(14,9).
connected(11,15).
connected(16,12).
connected(14,17).
connected(16,19).
```

Write a predicate path/2 that tells you from which point in the maze you can get to which other point when chaining together connections given in the above knowledge base.

Now ask some queries. Can you get from point 5 to point 10? Which other point can you get to when starting in point 1? And which points can be reached from point 13?

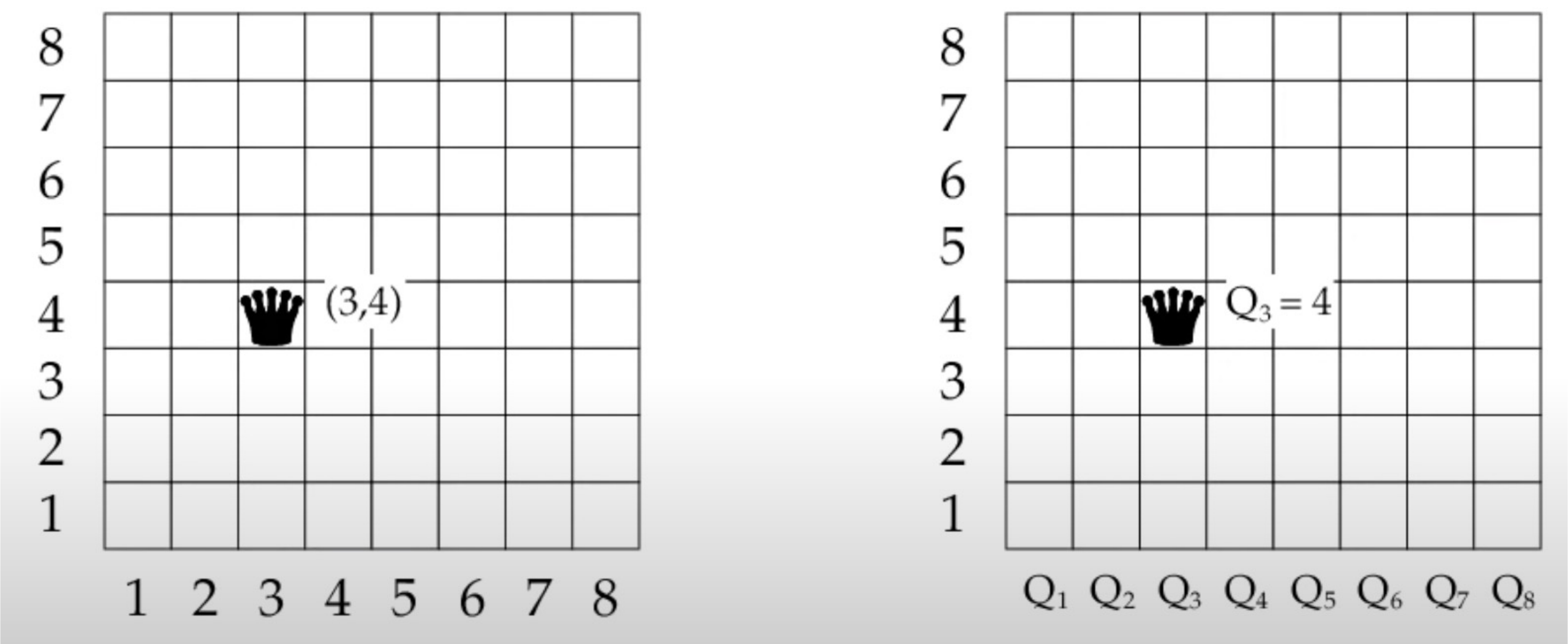
Solving N-Queens with Prolog



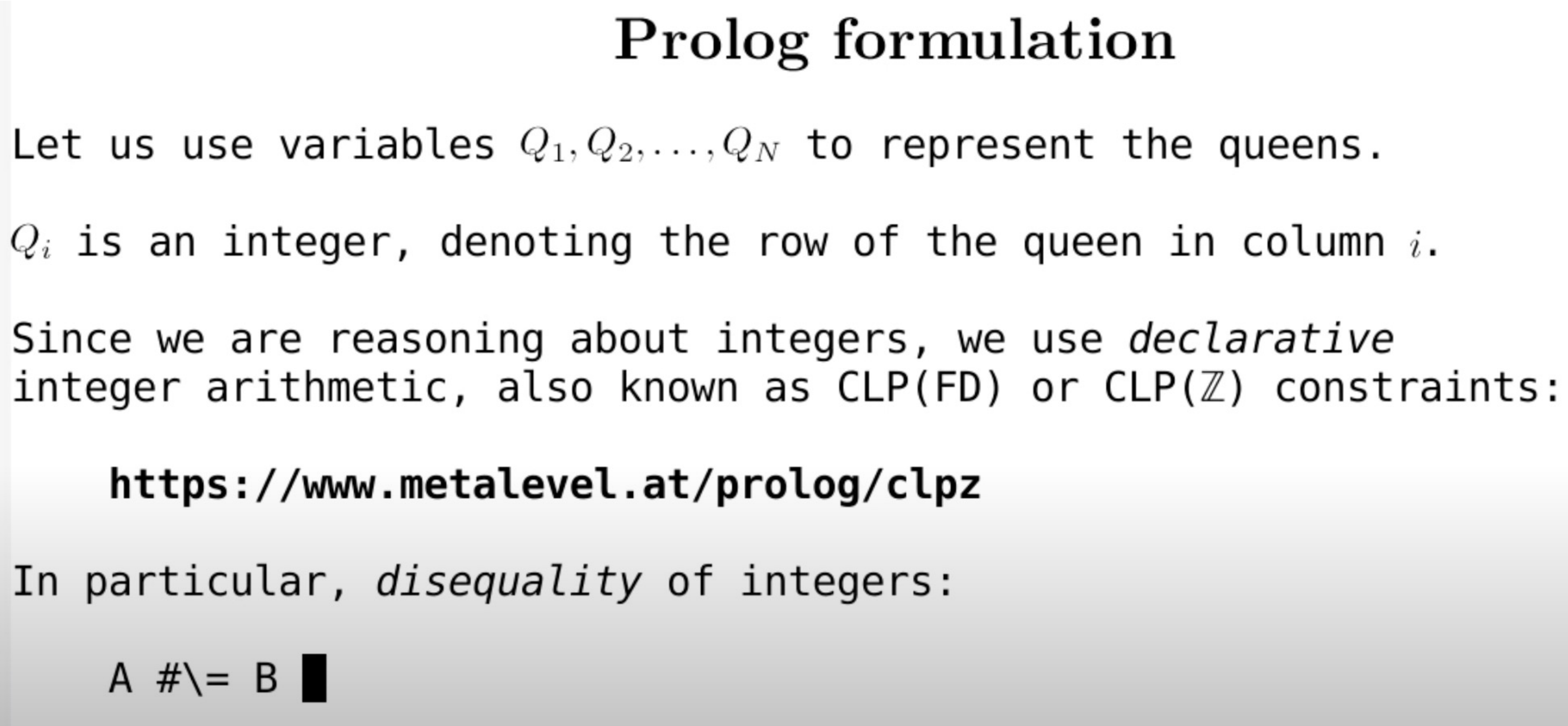
Task: Place *N* queens on an *N*×*N* chess board, so that none of the queens is under attack.

Solving N-Queens with Prolog

Representations



Solving N-Queens with Prolog



Prolog formulation

Let us use variables $Q_1, Q_2, \ldots, Q_N$ to represent the queens.

Q_i is an integer, denoting the row of the queen in column i.

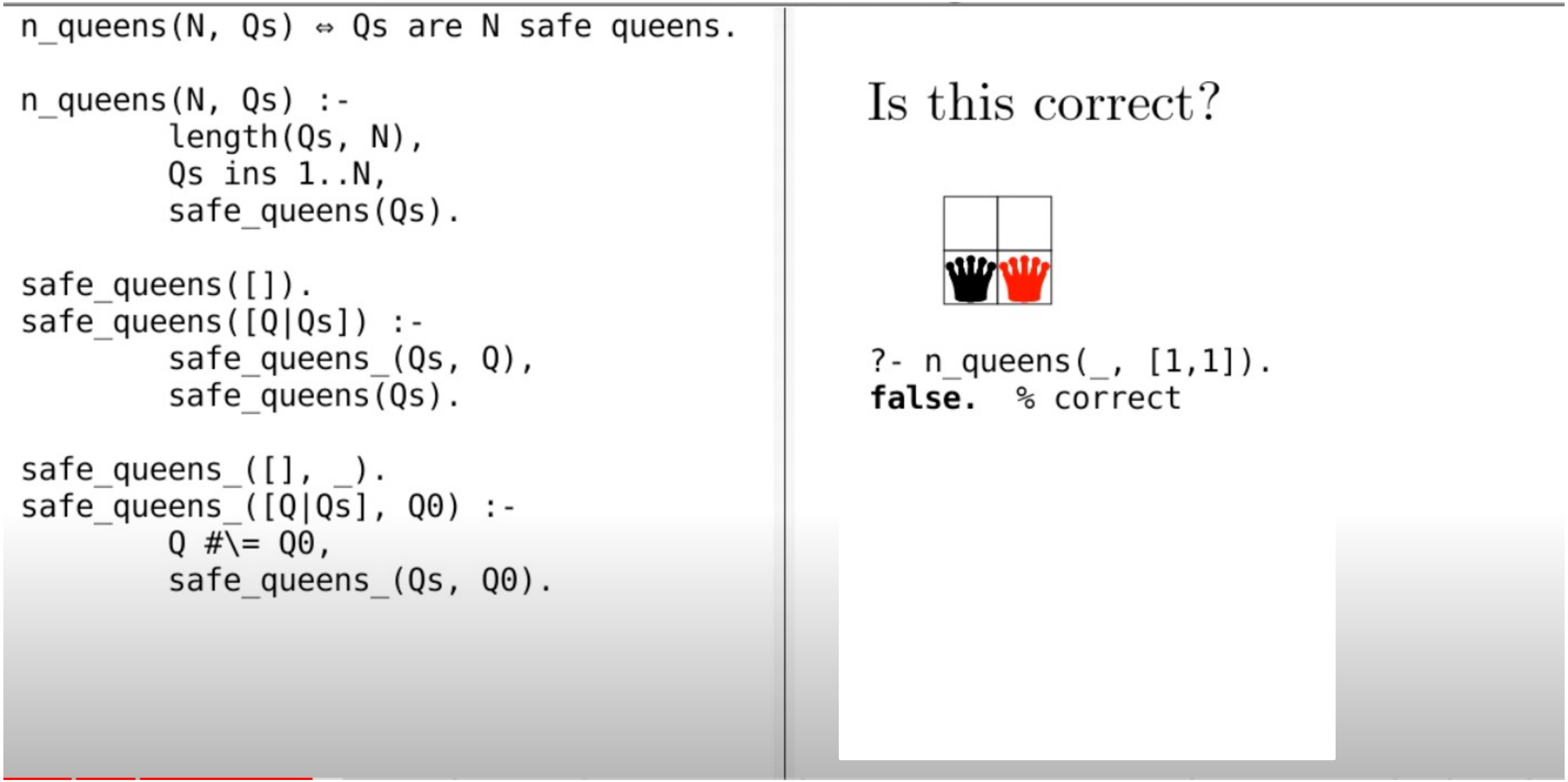
Since we are reasoning about integers, we use *declarative* integer arithmetic, also known as CLP(FD) or CLP($\mathbb{Z}$) constraints:

https://www.metalevel.at/prolog/clpz

In particular, *disequality* of integers:

```
A #\= B
```

Solving N-Queens with Prolog

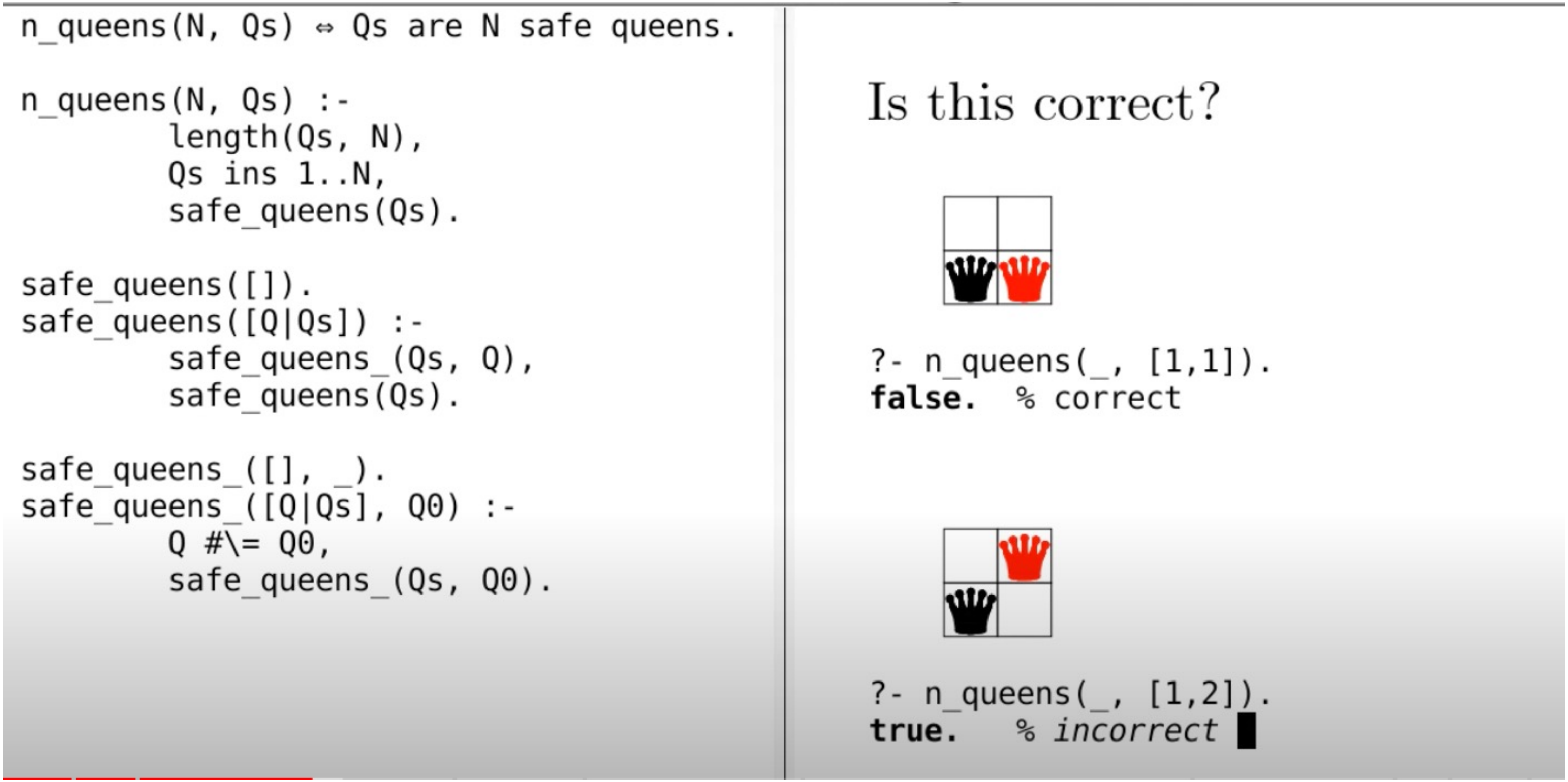


```
n_queens(N, Qs) ⇔ Qs are N safe queens.

n_queens(N, Qs) :-
        length(Qs, N),
        Qs ins 1..N,
        safe_queens(Qs).

safe_queens([]).
safe_queens([Q|Qs]) :-
        safe_queens_(Qs, Q),
        safe_queens(Qs).

safe_queens_([], _).
safe_queens_([Q|Qs], Q0) :-
        Q #\= Q0,
        safe_queens_(Qs, Q0).
```

Is this correct?

```
?- n_queens(_, [1,1]).
false.  % correct
```

Solving N-Queens with Prolog



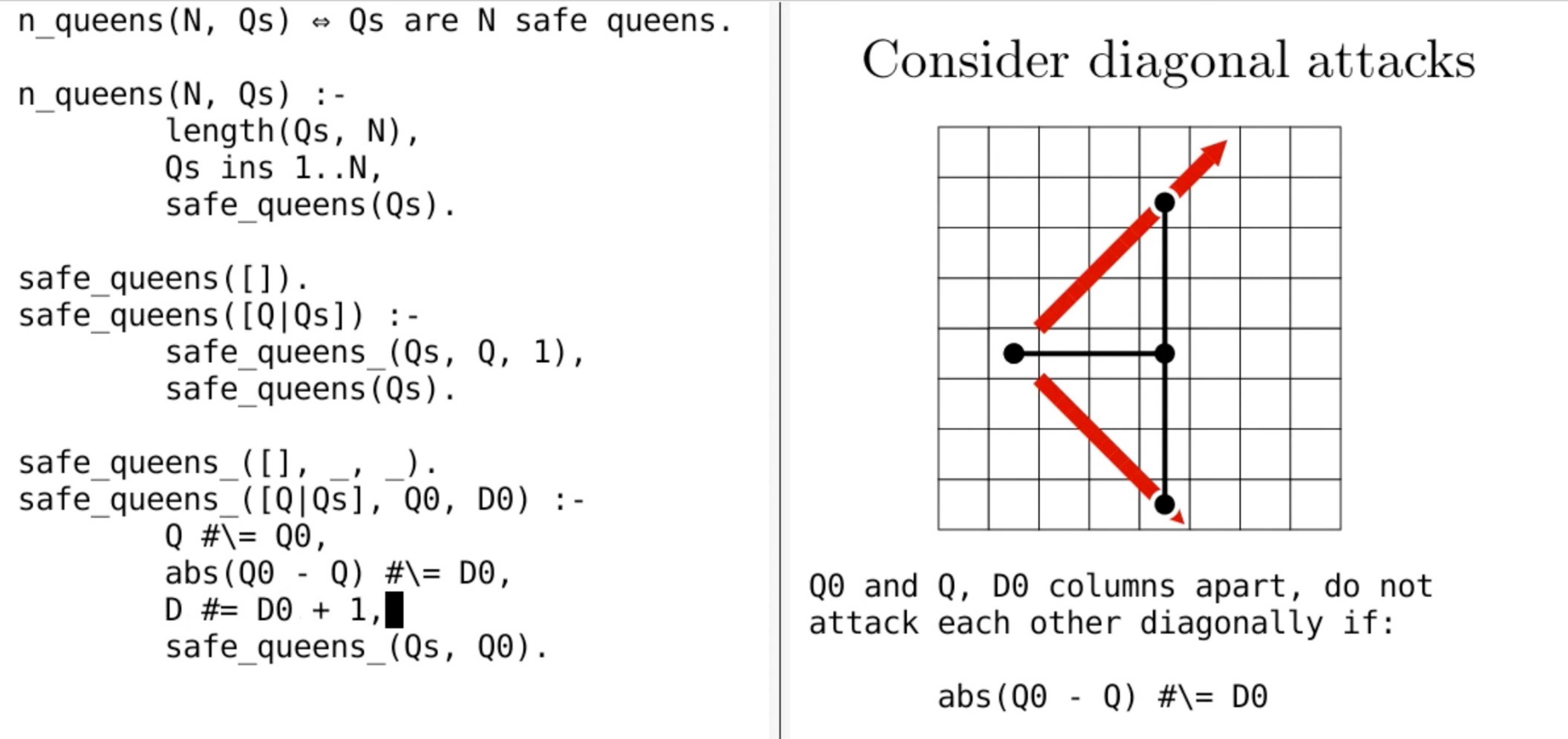
```
n_queens(N, Qs) ⇔ Qs are N safe queens.

n_queens(N, Qs) :-
        length(Qs, N),
        Qs ins 1..N,
        safe_queens(Qs).

safe_queens([]).
safe_queens([Q|Qs]) :-
        safe_queens_(Qs, Q),
        safe_queens(Qs).

safe_queens_([], _).
safe_queens_([Q|Qs], Q0) :-
        Q #\= Q0,
        safe_queens_(Qs, Q0).
```

Is this correct?

```
?- n_queens(_, [1,1]).
false.   % correct
```

```
?- n_queens(_, [1,2]).
true.    % incorrect
```

Solving N-Queens with Prolog



```
n_queens(N, Qs) ⇔ Qs are N safe queens.

n_queens(N, Qs) :-
        length(Qs, N),
        Qs ins 1..N,
        safe_queens(Qs).

safe_queens([]).
safe_queens([Q|Qs]) :-
        safe_queens_(Qs, Q, 1),
        safe_queens(Qs).

safe_queens_([], _, _).
safe_queens_([Q|Qs], Q0, D0) :-
        Q #\= Q0,
        abs(Q0 - Q) #\= D0,
        D #= D0 + 1,
        safe_queens_(Qs, Q0).
```

Consider diagonal attacks

Q0 and Q, D0 columns apart, do not attack each other diagonally if:

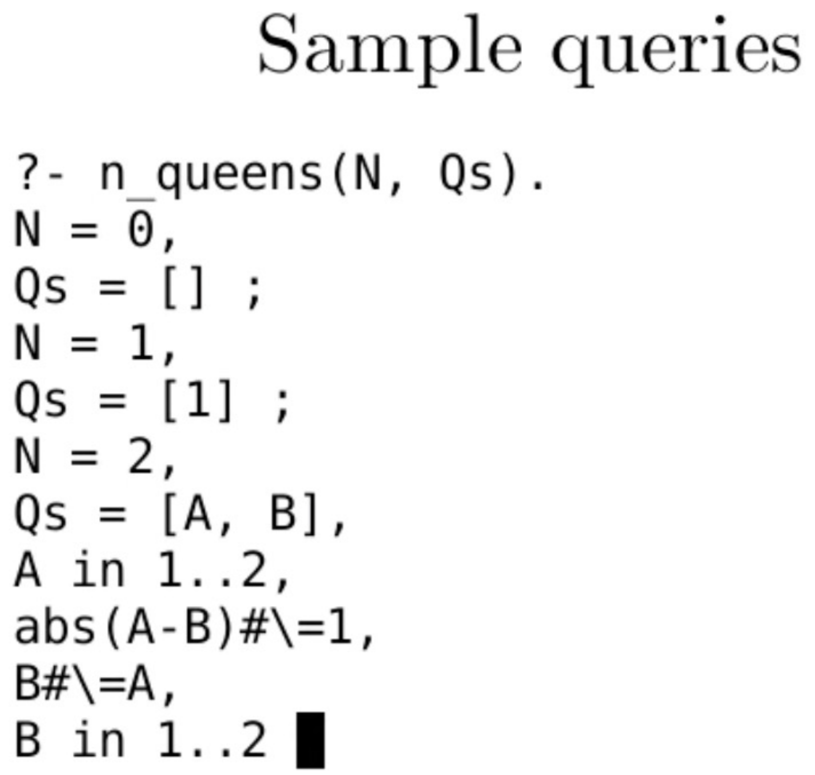
```
abs(Q0 - Q) #\= D0
```

Solving N-Queens with Prolog

`n_queens(N, Qs) ⇔ Qs are N safe queens.`

```
n_queens(N, Qs) :-
        length(Qs, N),
        Qs ins 1..N,
        safe_queens(Qs).

safe_queens([]).
safe_queens([Q|Qs]) :-
        safe_queens_(Qs, Q, 1),
        safe_queens(Qs).

safe_queens_([], _, _).
safe_queens_([Q|Qs], Q0, D0) :-
        Q #\= Q0,
        abs(Q0 - Q) #\= D0,
        D #= D0 + 1,
        safe_queens_(Qs, Q0, D).
```

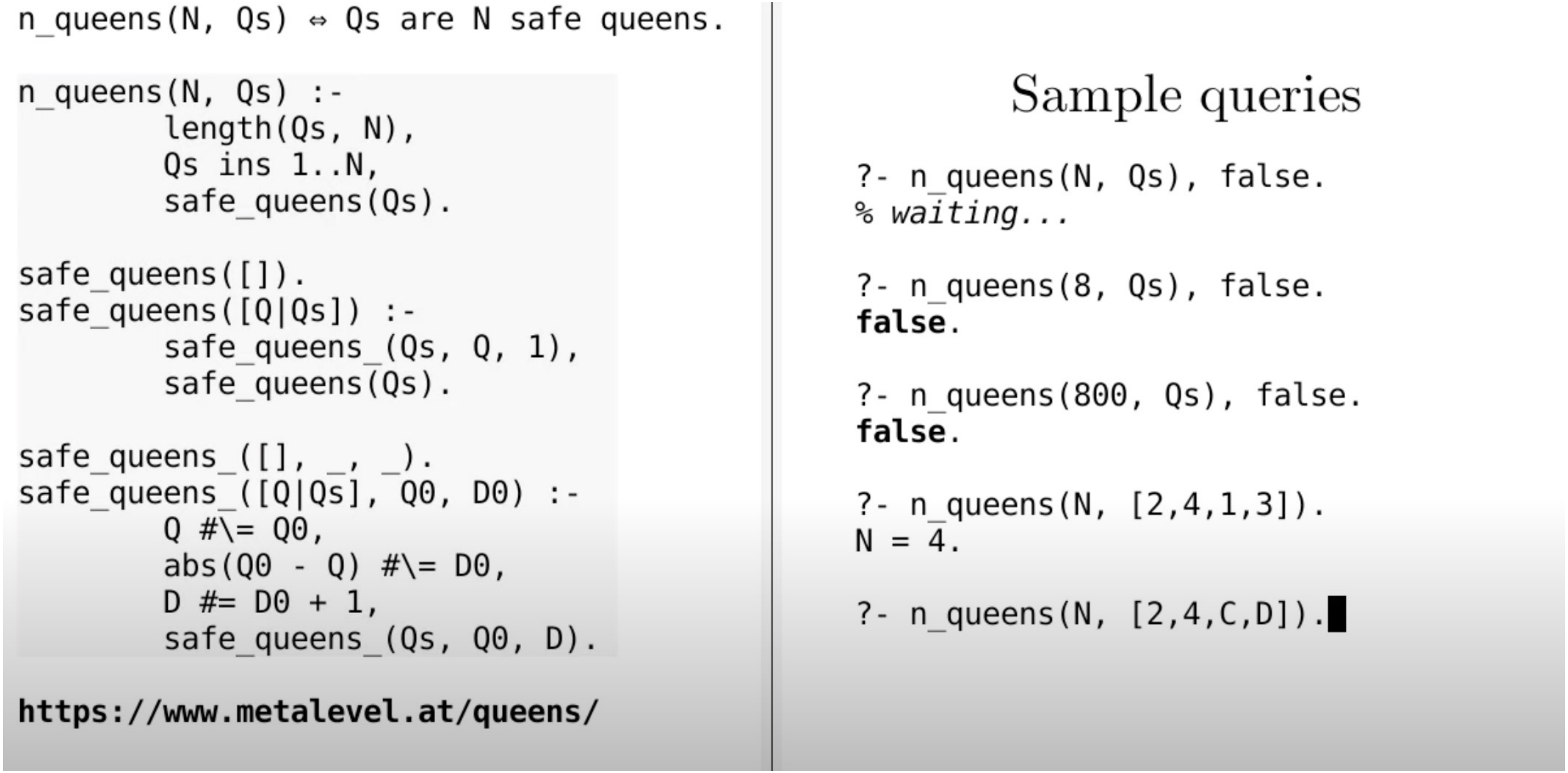
https://www.metalevel.at/queens/



Sample queries

```
?- n_queens(N, Qs).
N = 0,
Qs = [] ;
N = 1,
Qs = [1] ;
N = 2,
Qs = [A, B],
A in 1..2,
abs(A-B)#\=1,
B#\=A,
B in 1..2 █
```

Solving N-Queens with Prolog



```
n_queens(N, Qs) ⇔ Qs are N safe queens.

n_queens(N, Qs) :-
        length(Qs, N),
        Qs ins 1..N,
        safe_queens(Qs).

safe_queens([]).
safe_queens([Q|Qs]) :-
        safe_queens_(Qs, Q, 1),
        safe_queens(Qs).

safe_queens_([], _, _).
safe_queens_([Q|Qs], Q0, D0) :-
        Q #\= Q0,
        abs(Q0 - Q) #\= D0,
        D #= D0 + 1,
        safe_queens_(Qs, Q0, D).
```

https://www.metalevel.at/queens/

Sample queries

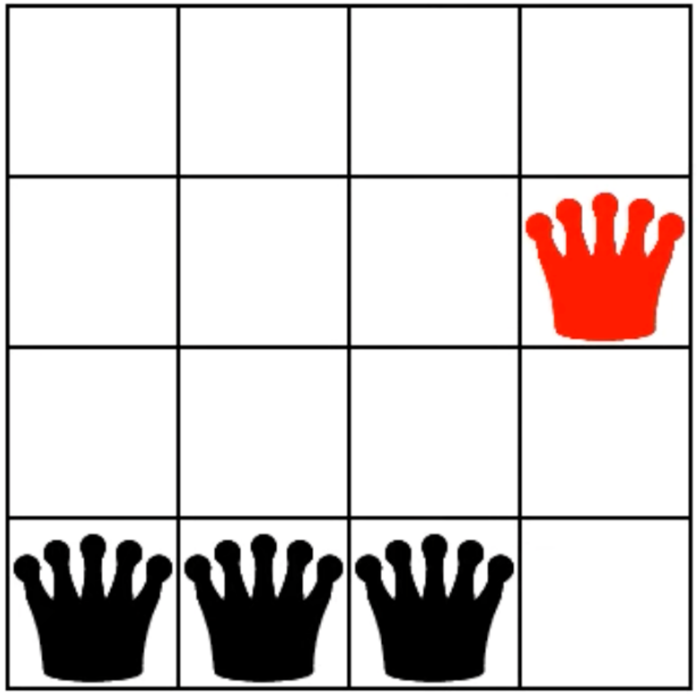
```
?- n_queens(N, Qs), false.
% waiting...

?- n_queens(8, Qs), false.
false.

?- n_queens(800, Qs), false.
false.

?- n_queens(N, [2,4,1,3]).
N = 4.

?- n_queens(N, [2,4,C,D]).
```

Solving N-Queens with Prolog

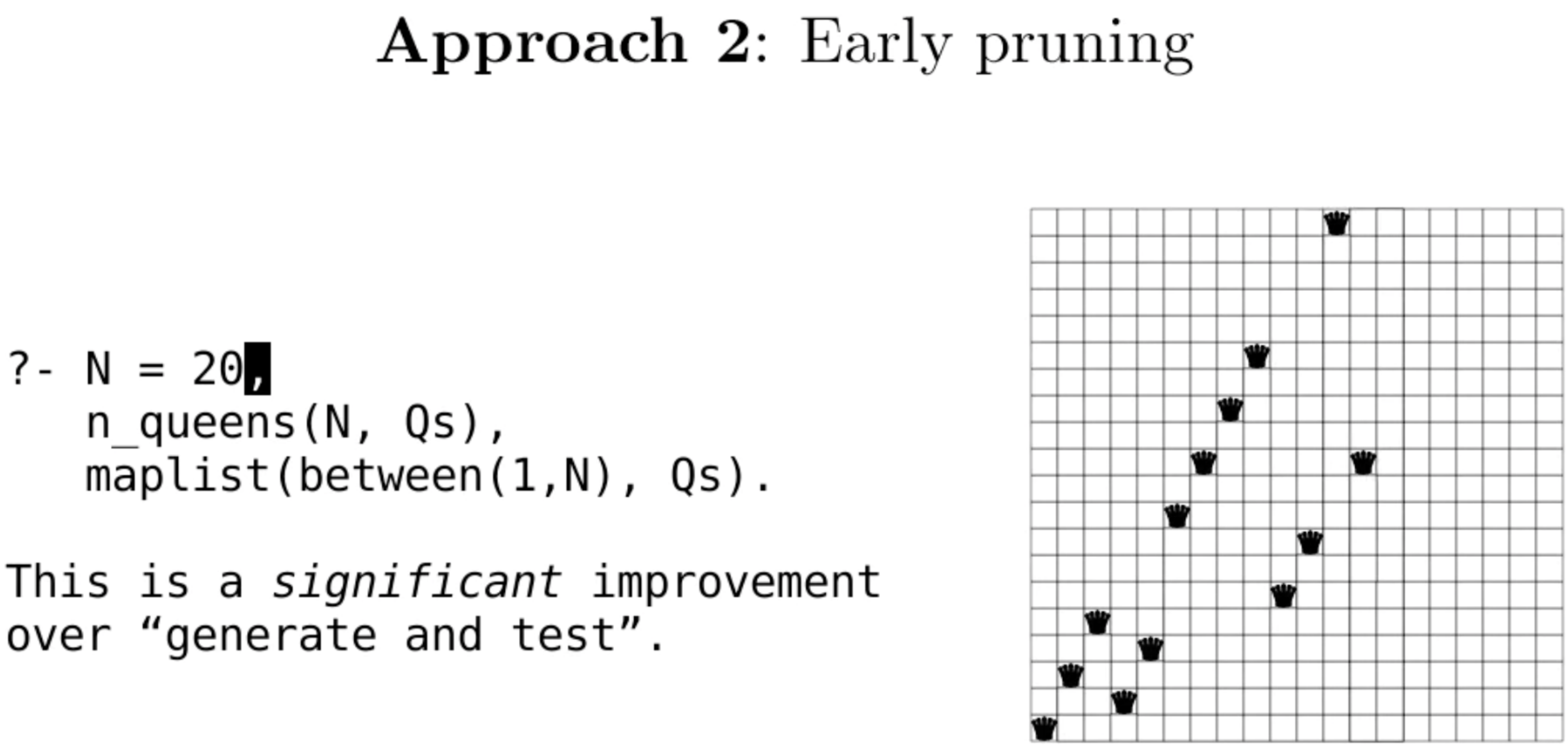
Approach 1: "Generate and test"

```
?- N = 4,
   length(Qs, N),
   maplist(between(1,N), Qs),
   n_queens(N, Qs).
```

extremely inefficient, $\mathcal{O}(N^N)$



Solving N-Queens with Prolog

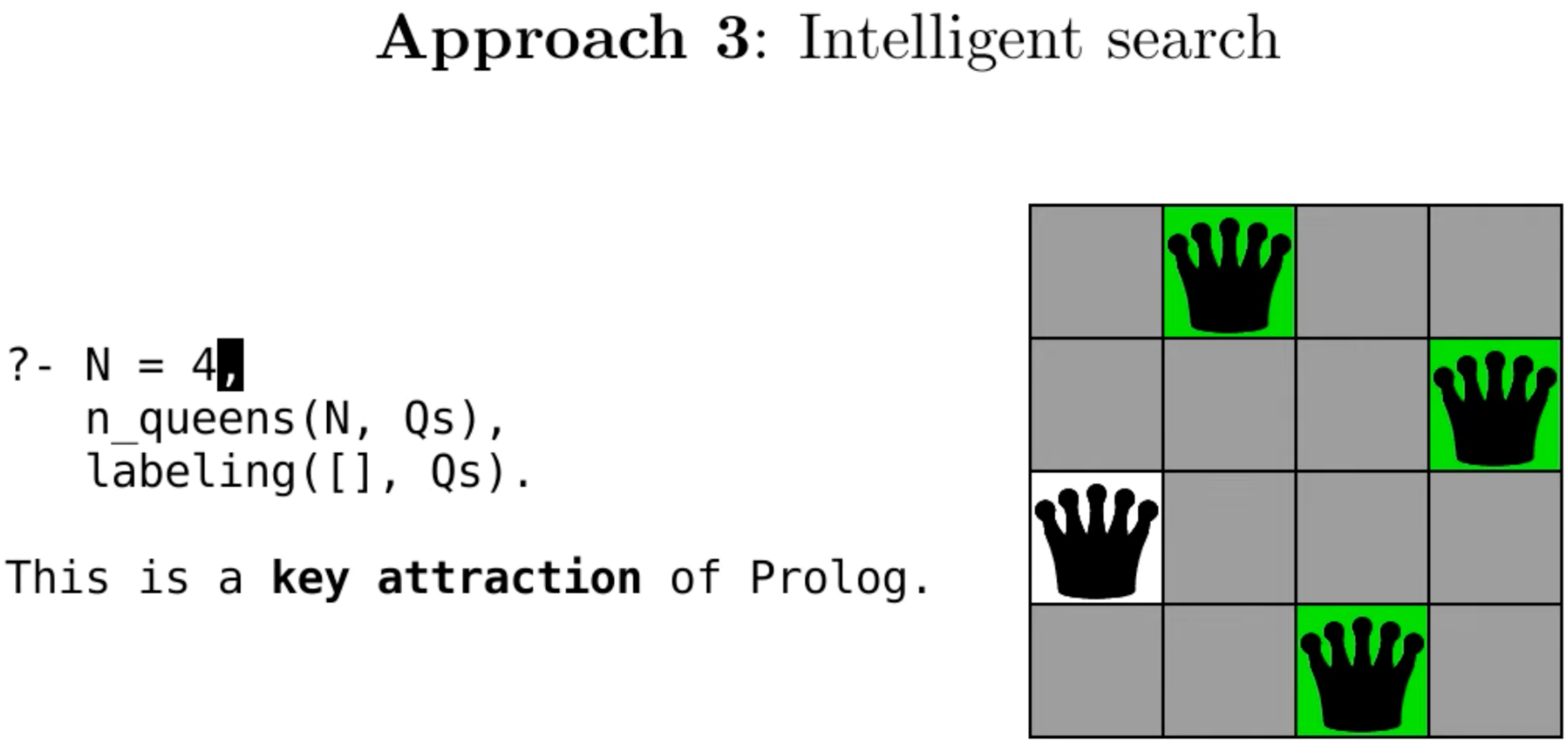


Approach 2: Early pruning

```
?- N = 20,
   n_queens(N, Qs),
   maplist(between(1,N), Qs).
```

This is a *significant* improvement over “generate and test”.

Solving N-Queens with Prolog

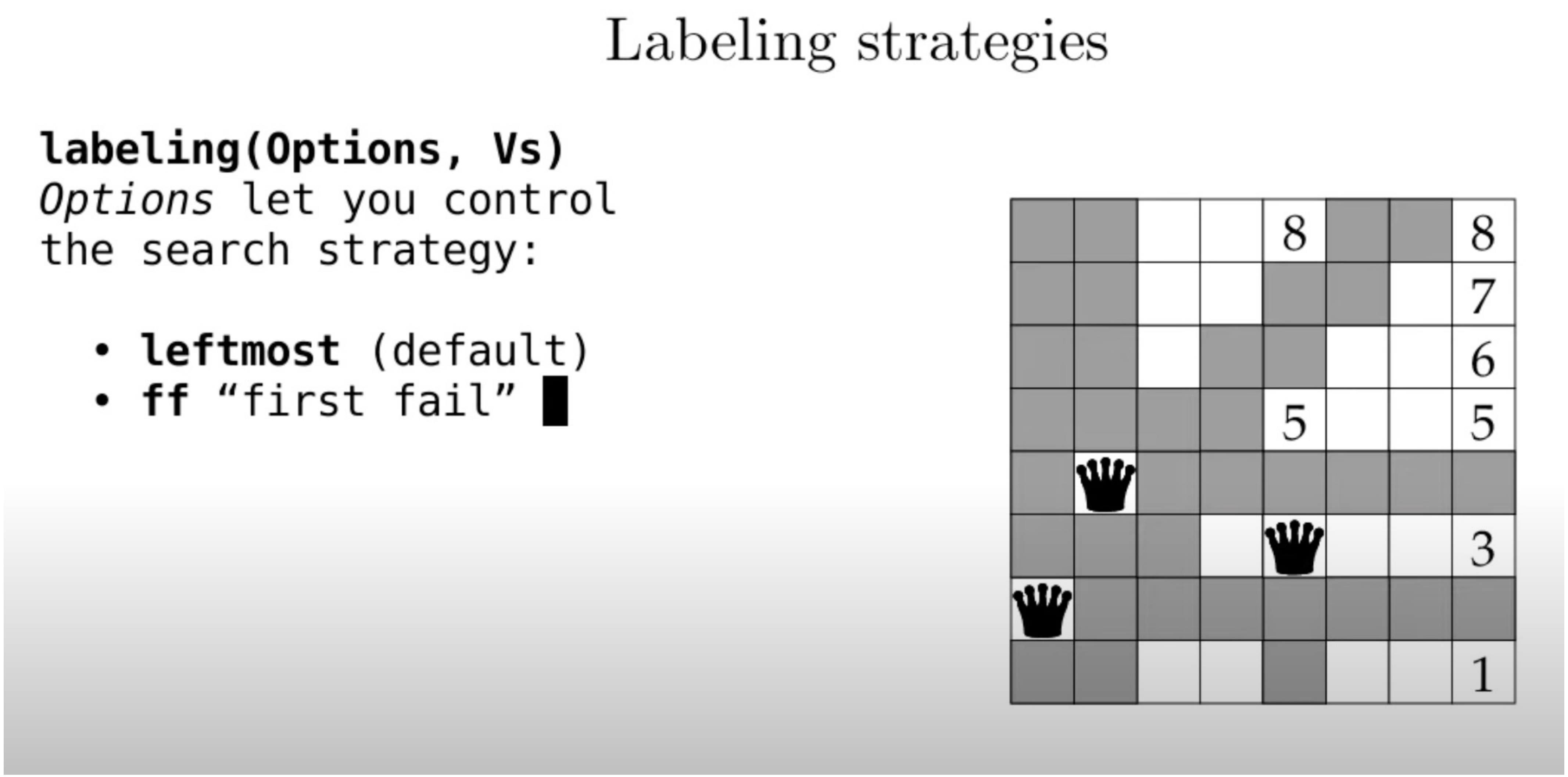


Approach 3: Intelligent search

```
?- N = 4,
   n_queens(N, Qs),
   labeling([], Qs).
```

This is a **key attraction** of Prolog.

Solving N-Queens with Prolog

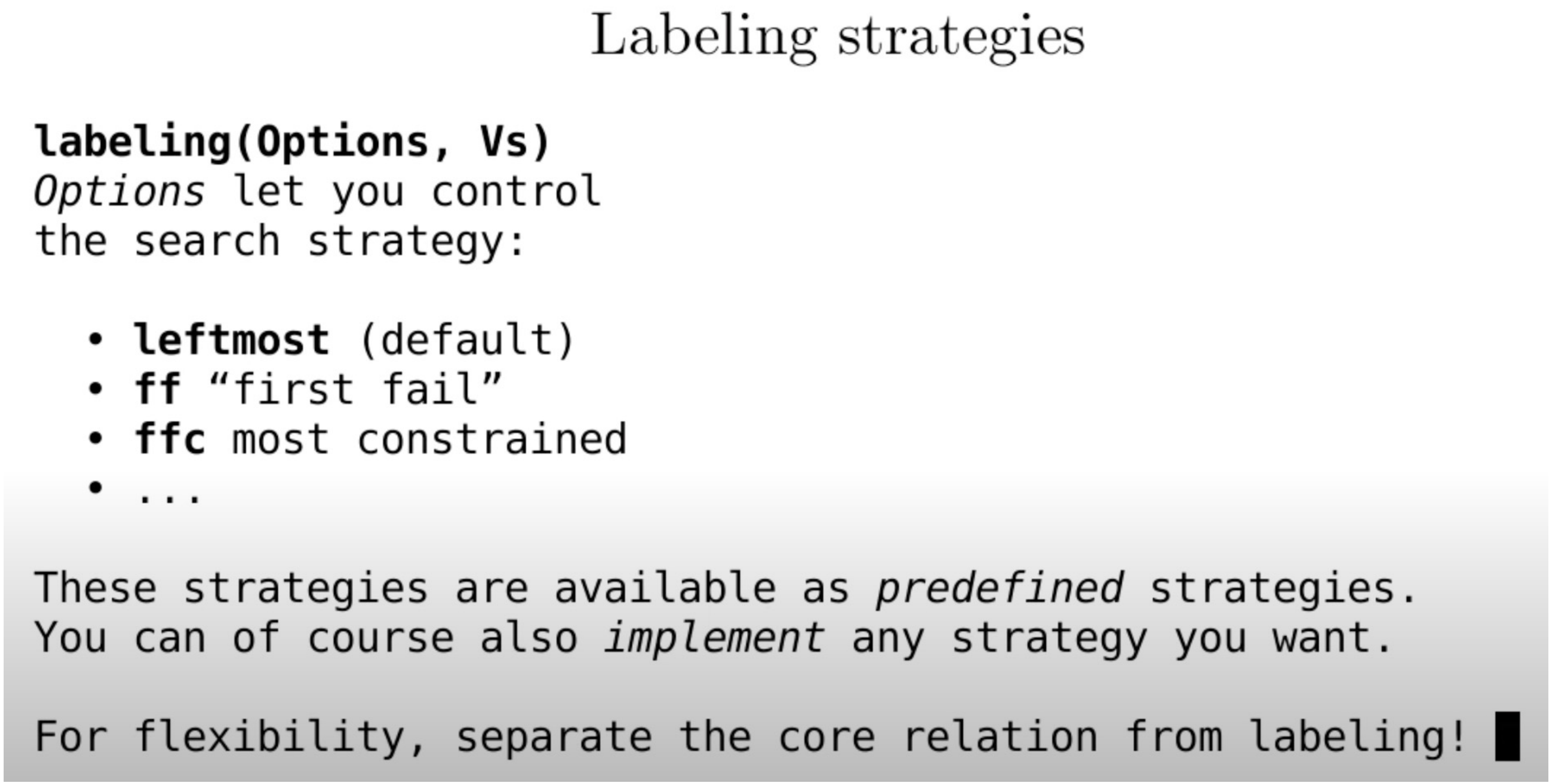


Labeling strategies

`labeling(Options, Vs)`
Options let you control the search strategy:

- **leftmost** (default)
- **ff** “first fail”

Solving N-Queens with Prolog



Labeling strategies

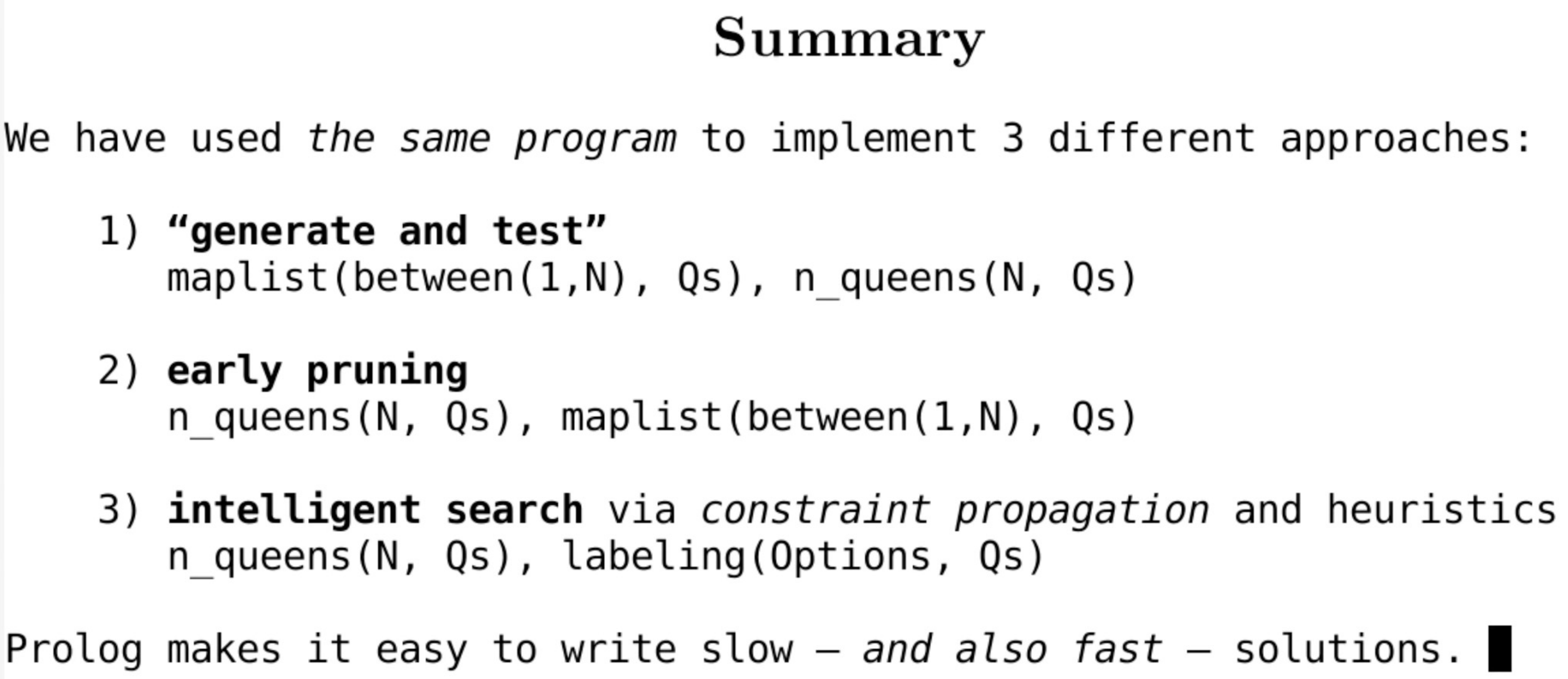
labeling(Options, Vs)
Options let you control
the search strategy:

- **leftmost** (default)
- **ff** “first fail”
- **ffc** most constrained
- ...

These strategies are available as *predefined* strategies.
You can of course also *implement* any strategy you want.

For flexibility, separate the core relation from labeling!

Solving N-Queens with Prolog



Summary

We have used *the same program* to implement 3 different approaches:

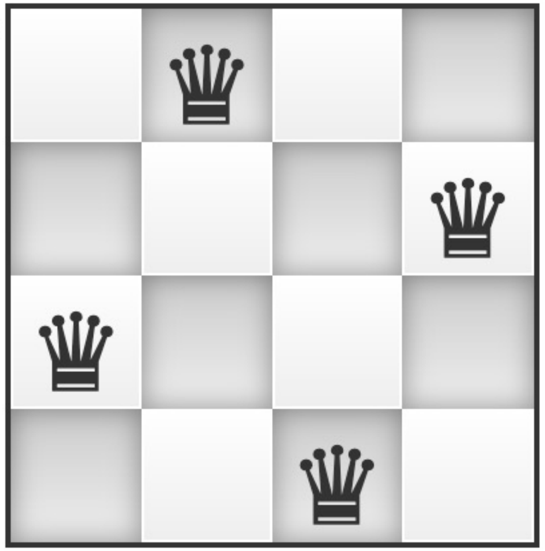
1) **“generate and test”**
   ```
   maplist(between(1,N), Qs), n_queens(N, Qs)
   ```

2) **early pruning**
   ```
   n_queens(N, Qs), maplist(between(1,N), Qs)
   ```

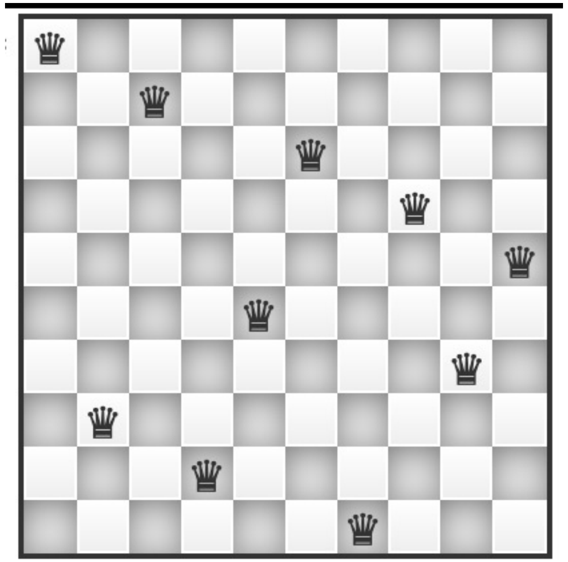
3) **intelligent search** via *constraint propagation* and heuristics
   ```
   n_queens(N, Qs), labeling(Options, Qs)
   ```

Prolog makes it easy to write slow – *and also fast* – solutions.

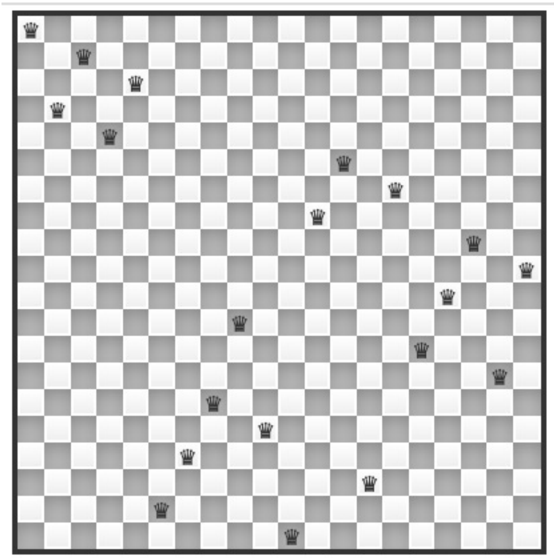
Solving N-Queens with Prolog



N=4



N=10

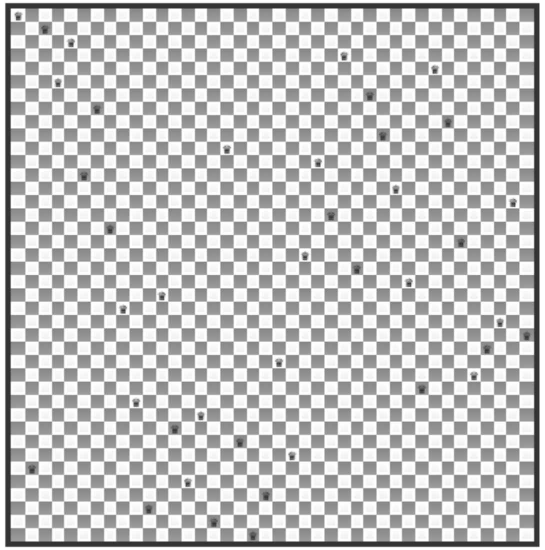


N=20,

2.144 CPUtime

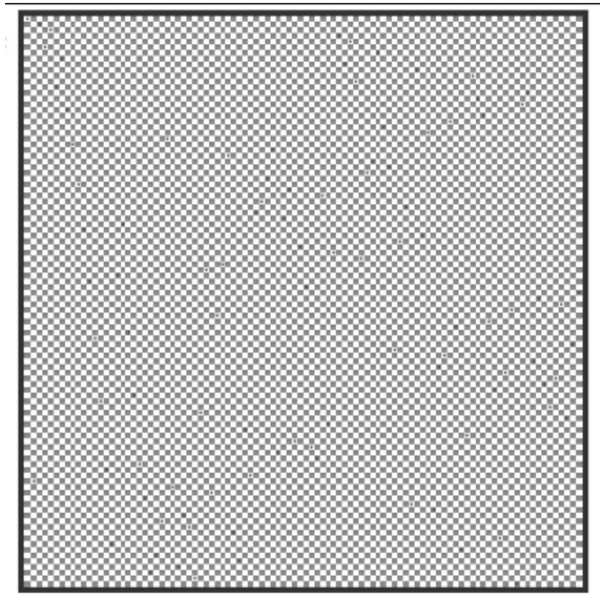
(default labeling)

Solving N-Queens with Prolog: with labelling



N=40, 0.23CPU time

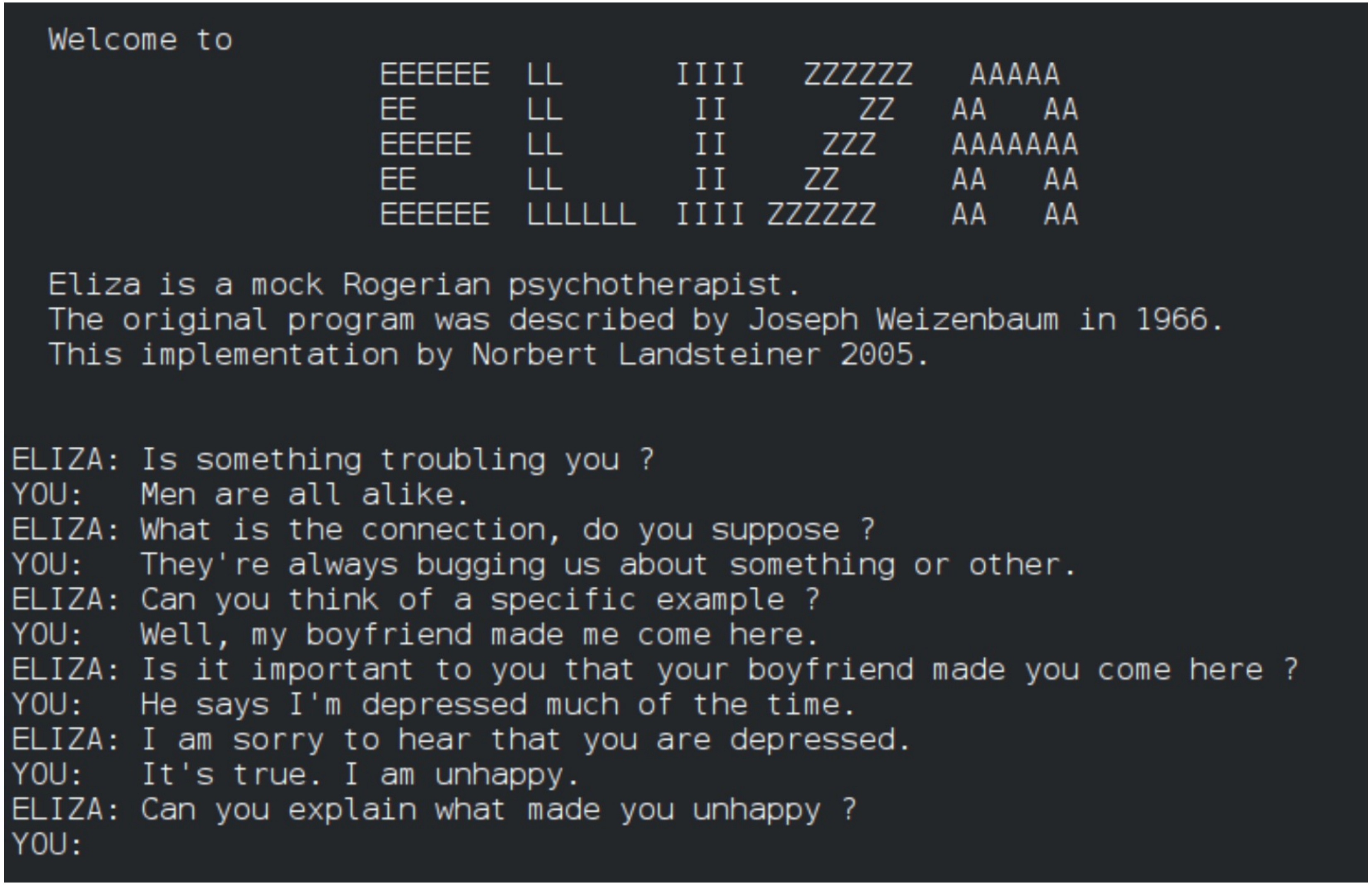
(smart labeling)



N=100, 1.363CPU time

(smart labeling)

Implementing Eliza (1964 ChatGPT ;-)) with Prolog



```
  Welcome to
                    EEEEEE  LL      IIII   ZZZZZZ   AAAAA
                    EE      LL       II       ZZ   AA   AA
                    EEEEE   LL       II     ZZZ    AAAAAAA
                    EE      LL       II    ZZ      AA   AA
                    EEEEEE  LLLLLL  IIII ZZZZZZ    AA   AA

  Eliza is a mock Rogerian psychotherapist.
  The original program was described by Joseph Weizenbaum in 1966.
  This implementation by Norbert Landsteiner 2005.


ELIZA: Is something troubling you ?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:
```

Eliza-like program with Prolog

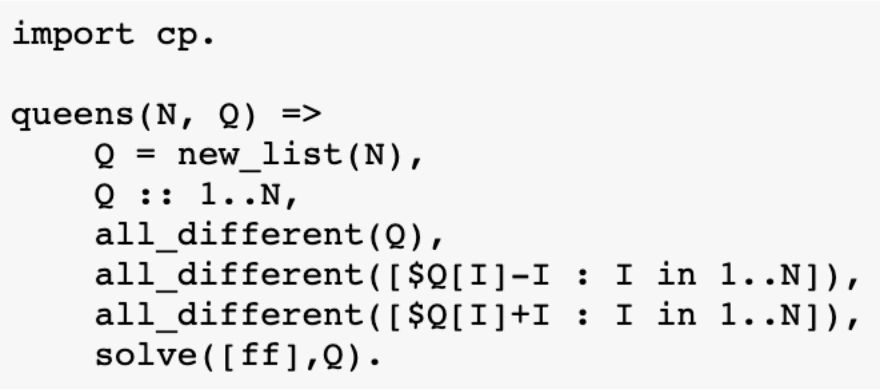
```
eliza(Stimuli, Response) :-
    template(InternalStimuli, InternalResponse),
    match(InternalStimuli, Stimuli),
    match(InternalResponse, Response),
    !.

template([s([i,am]),s(X)], [s([why,are,you]),s(X),w('?')]).
template([w(i),s(X),w(you)], [s([why,do,you]),s(X),w(me),w('?')]).

match([],[]).
match([Item|Items],[Word|Words]) :-
    match(Item, Items, Word, Words).

match(w(Word), Items, Word, Words) :-
    match(Items, Words).
match(s([Word|Seg]), Items, Word, Words0) :-
    append(Seg, Words1, Words0),
    match(Items, Words1).
```

```
eliza(+Stimuli, -Response) is det.
@param  Stimuli is a list of atoms (words).
@author Richard A. O'Keefe (The Craft of Prolog)
```

```
/** <examples>

?- eliza([i, am, very, hungry], Response).
?- eliza([i, love, you], Response).

*/
```

Resources

- https://www.metalevel.at/prolog/introduction (https://www.youtube.com/watch?v=l_tbL9RjFdo for N-Queens)
- Sethi, Ravi. *Programming languages concepts and constructs*. Addison Wesley Longman Publishing Co., Inc., 1996. (Chapter 11)
- https://cs.union.edu/~striegnk/courses/esslli04prolog/
- https://swish.swi-prolog.org/example/examples.swinb
- Prolog for data science https://emiruz.com/post/2023-04-30-prolog-for-data-science/
- Picat-lang.org



```
import cp.

queens(N, Q) =>
    Q = new_list(N),
    Q :: 1..N,
    all_different(Q),
    all_different([$Q[I]-I : I in 1..N]),
    all_different([$Q[I]+I : I in 1..N]),
    solve([ff],Q).
```