A constraint satisfaction problem is a triple \((V, D, C)\) where:

1. \(V = \{v_1, v_2, \ldots, v_n\}\) is a finite set of variables;
2. \(D = \{d_1, d_2, \ldots, d_m\}\) is a finite set of values for \(v_i \in V\) \((i = 1, n)\);
3. \(C = \{c_1, c_2, \ldots, c_j\}\) is a finite set of constraints on the values that can be assigned to different variables at the same time.

The solution of the constraint satisfaction problem consists of defining substitutions for variables from corresponding sets of possible values so as to satisfy all the constraints in \(C\).

Traditional approach: “generate and test” methods or chronological backtracking. But, these methods only work on small problems, because they have exponential complexity.
The N-Queens example: the constraint satisfaction approach

The most important question that must be addressed with respect to this problem is how to find consistent column placements for each queen. The solution in the book is based on the idea of "choice sets". A choice set is a set of alternative placements. Consider, for example, the following configuration for \( N = 4 \):

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
\hline
0 & Q & Q & \\
1 & Q & \\
2 & Q & Q & \\
3 & Q & &
\end{array}
\]

- choice set 1 = \{ (0,0), (1,0), (2,0), (3,0) \}
- choice set 2 = \{ (0,1), (1,1), (2,1), (3,1) \}
- choice set 3 = \{ (0,2), (1,2), (2,2), (3,2) \}
- choice set 4 = \{ (0,3), (1,3), (2,3), (3,3) \}

Notice that in each choice set, choices are mutually exclusive and exhaustive.
Each solution (legal placement of queens) is a consistent combination of choices - one from each set. To find a solution, we must:

1. Identify choice sets.
2. Use search through the set of choice sets to find a consistent combination of choices (one or all). A possible search strategy, utilizing chronological backtracking is the following one (partial graph shown):

```
Choice set 1
    /
   /   (0,0)
  /
(0,1)

Choice set 2
    /
   /
  X

Choice set 3
    /
   /
  X

Choice set 4
    /
   /
  X
  /
  X
  /
  X
  /
  X
  /
  X
  /
  X
```

Choice set 1 and choice set 2 are valid, while choice set 3 and choice set 4 are inconsistent combinations of choices.
3. Finding solutions to search problems.

Consider the following graph

Assume you want to color the nodes so that every node is red, or green, or yellow, and adjacent nodes are of different colors. Let "1" means "red", "2" means "green", and "3" means "yellow". Then, the following set of constraints describe this problem:

- A1 or A2 or A3
- B1 or B2 or B3
- C1 or C2 or C2
- D1 or D2 or D3
- E1 or E2 or E2
- not (A1 and B1)
- not (A2 and B2)
- not (A3 and B3)
- not (A1 and C1)
- not (A2 and C2)
- not (A3 and C3)
- not (B1 and D1)
- not (B2 and D2)
- not (B3 and D3)
- not (D2 and E2)
- not (D3 and E3)
- not (C1 and E1)
- not (C2 and E2)
- not (C3 and E3)
To find a solution that satisfies all of the constraints, we can use search:

- A is red
  - B is red
    - C is red
      - D is red
        - E is red
  - B is green
  - B is yellow
- A is green
- A is yellow
- A is yellow
A generic procedure for searching through choice sets utilizing chronological backtracking

The following is a generic procedure that searches through choice sets. When an inconsistent choice is detected, it backtracks to the most recent choice looking for an alternative continuation. This strategy is called chronological backtracking.

```
(defun Chrono (choice-sets)
  (if (null choice-sets) (record-solution)
    (dolist (choice (first choice-sets))
      (while-assuming choice
        (if (consistent?)
          (Chrono (rest choice-sets)))))))
```

Notice that when an inconsistent choice is encountered, the algorithm backtracks to the previous choice it made. This algorithm is not efficient because: (1) it is exponential, and (2) it re-invents contradictions. We shall discuss another approach called, dependency-directed backtracking, which handles this type of search problems in a more efficient way.
Example

Choose in sequence:
  – A or B
  – C or D
  – E or F

Given: A and C cannot hold together
Given: B and E cannot hold together

Assume we want all consistent solutions
Assume that we cannot test until every choice has been made
Example Search Space
(global view)
Dependencies can guide backtracking

\[
\begin{align*}
\text{A} & \quad \text{C} \\
\text{E} & \quad \{A, C, E\}
\end{align*}
\]

*Better tactic*

*Bad tactic*
Chronological Backtracking

- Often wastes computation

Example: Suppose D and F together cause lots of work. Popping context loses this work
Chronological Backtracking RedisCOVERS Contradictions

Example: Useless to try B and E together more than once