Ch. 5 Recursion as a Problem Solving Technique

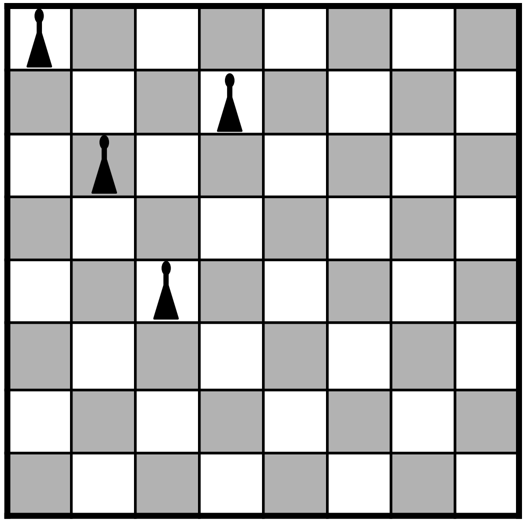
- Chapter 2
  - Basics on recursion
- Chapter 5
  - More on recursion
  - Two useful concepts
    - Backtracking & formal grammars

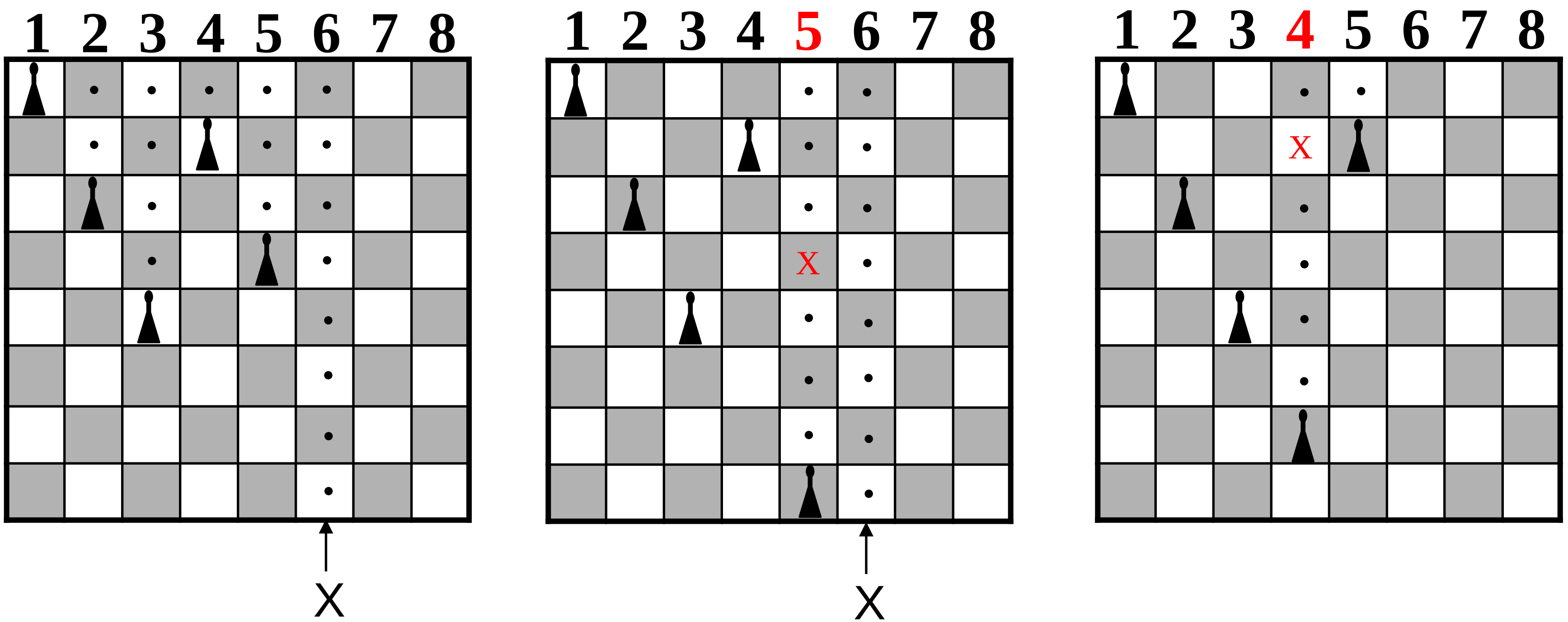
Backtracking

- A search strategy by a seq. of guesses
- Guesses, retraces in reverse order, and tries a new sequence of steps
- Has a strong relationship with recursion and stack

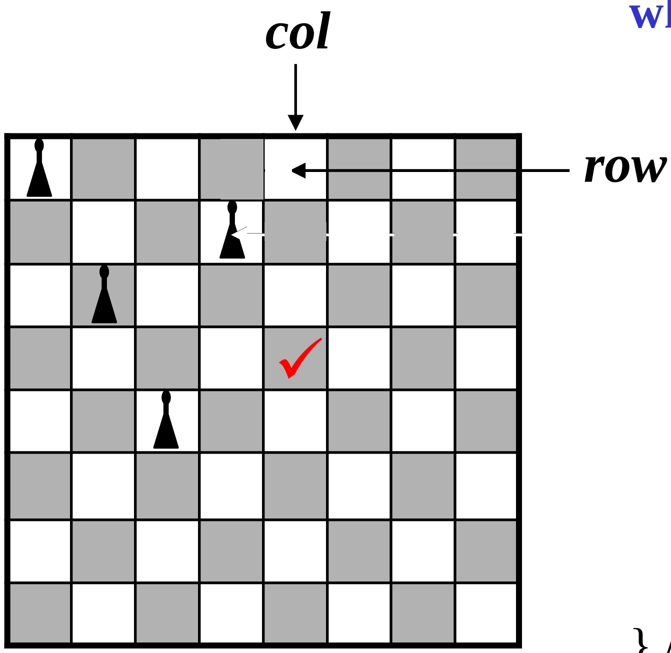
Eight-Queens Problem: An Example

- Chessboard with 64 squares
  - 8 rows and 8 columns
- A queen can attack other pieces
  - within its row
  - within its column
  - along its diagonal
- Want to place eight queens on the chessboard so that no queen can attack any other queen

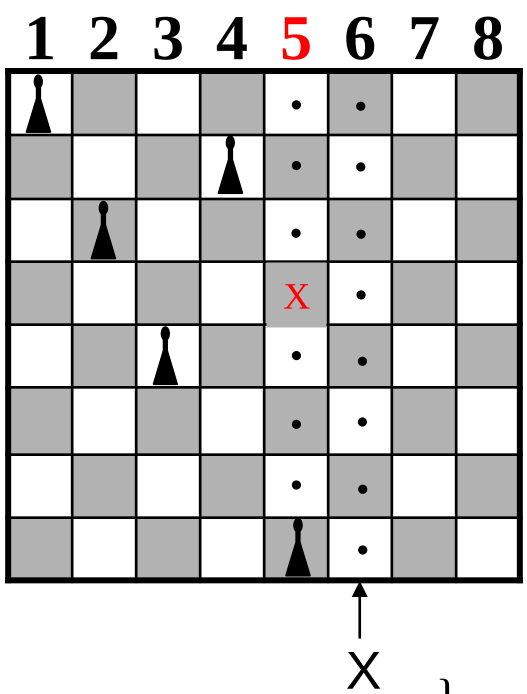



1 2 3 4 5 6 7 8
X
1 2 3 4 5 6 7 8
X
X
1 2 3 4 5 6 7 8
X

```
Public boolean placeQueens(int col) {
// Situation: Queens are placed correctly in columns 1 thru col – 1
// Return true if a solution is found; return false if there is no solution;
        if (col > BOARD_SIZE) {
                return true;
        } else {
                boolean queenPlaced = false;
                int row = 1;  // square id in column
                while (!queenPlaced && (row <= BOARD_SIZE)) {
                        if (isUnderAttack(row, col)) {
                                ++row; // consider next square
                        } else { // found valid square
                                setQueen(row, col);
                                queenPlaced = placeQueens(col+1);
                                if (!queenPlaced) { // failed
                                        removeQueen(row, col);
                                        ++row;
                                }
                        }
                } // end while
                return queenPlaced;
        }
}
```



```
public boolean placeQueens(int col) {
// Situation: Queens are placed correctly in columns 1 thru col – 1
// Return true if a solution is found; return false if there is no solution;
        if (col > BOARD_SIZE) {
                return true;
        } else {
                boolean queenPlaced = false;
                int row = 1;  // square id in column
                while (!queenPlaced && (row <= BOARD_SIZE)) {
                        if (isUnderAttack(row, col)) {
                                ++row; // consider next square
                        } else { // found valid square
                                setQueen(row, col);
                                queenPlaced = placeQueens(col+1);
                                if (!queenPlaced) { // failed
                                        removeQueen(row, col);
                                        ++row;
                                }
                        }
                } // end while
                return queenPlaced;
        }
}
```



Formal Grammars

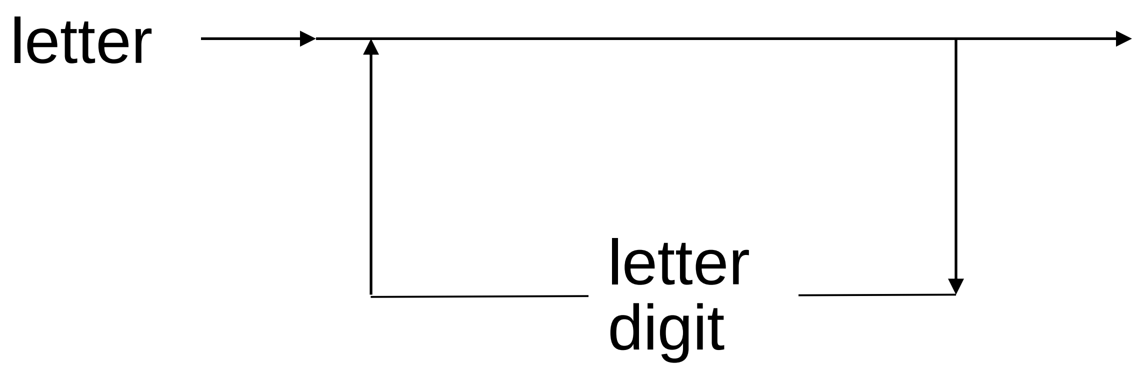
- Basics
  - x|y means x or y
  - xy means x followed by y
  - <*word*> means any instance of *word* that the definition defines

Example – JAVA Identifier

<identifier> = *<letter>* | *<identifier>* *<letter>* | *<identifier>* *<digit>*

<letter> = a | b | · · · | z | A | B | · · · | Z | _ | $

<digit> = 0 | 1 | · · · | 9



- ✓ Valid identifiers
  - ➢ b, tmp3_off, $toy, _33, Zippul, …

Recursive Algorithm for <identifier> Determination

```
isId(w)  // pseudo code
{
    if (w is of length 1) {  // base case
        if (w is a letter) return true;
        else return false;
    } else if (the last character of w is a letter or a digit) {
        return isId(w minus its last character);
    } else {return false}
}
```

<identifier> <letter> | <identifier> <digit>

Example - Palindrome

Palindromes =

{***w*** | ***w*** reads the same left to right as right to left}

<*palindrome*> = empty string | <*ch*> |

a <*palindrome*> a | b <*palindrome*> b |

… | Z <*palindrome*> Z

<*ch*> = a | b | · · · | z | A | B | · · · | Z

- ✓ e.g.
  - ➢ abcba, chainniahc, lioninoil, …

Recursive Algorithm for Palindrome Determination

```
isPalindrome(w) // pseudo code
{
    if (w is empty string or w is of length 1)  return true;
    else if (w's first and last characters are the same)
          return isPalindrome(w minus its first and last characters);
    else return false;
}
```

Example - $A^n B^n$

L = {$\boldsymbol{w}$ | $\boldsymbol{w}$ is the form $A^n B^n$ for some n >= 0 }

<*L*> = empty string | A <*L*> B

- ✓ e.g.
  - ➢ AB, AAABBB, AAAAAABBBBBB, …

Recursive Algorithm for A^nB^n Determination

```
isAnBn(w) // pseudo code
{
    if (w is empty string) return true;
    else if (w begins w/ the character A and
                    ends w/ the character B)
        return isAnBn(w minus its first and last characters);
    else return false;
}
```

Example – Infix, Prefix, Postfix Expression

- Infix expression
  - The operator locates in the middle of the operands
  - The most popular
  - A + B * C – 2
- Prefix expression
  - The operator proceeds the operands
  - – + A * B C 2
- Postfix expression
  - The opeartor follows the operands
  - A B C * + 2 –

<infix> = *<identifier>* | *<infix>* *<operator>* *<infix>*
<operator> = + | – | * | /
<identifier> = a | b | … | z

<prefix> = *<identifier>* | *<operator>* *<prefix>* *<prefix>*
<operator> = + | – | * | /
<identifier> = a | b | … | z

<postfix> = *<identifier>* | *<postfix>* *<postfix>* *<operator>*
<operator> = + | – | * | /
<identifier> = a | b | … | z

Recursive Algorithm for Determination of Prefix

isPre(A, 1, n)

{ // return true if the string A[1…*n*] is in prefix form

// otherwise return false

lastChar = endPre(A, 1, *n*);

if (lastChar == *n*) **return true**;

else return false;

}

endPre(A, *first*, …): A[*first*]에서 시작하는 prefix expression이 여기서 끝난다는 걸 알아내기

```
endPre(A, first, last)
{   // input: A[first…last], e.g., +*ab+cd,  b, *bc
    // return the position of the end of the prefix expression
    //          beginning at A[first], if one exists
        if (first > last) return –1;
        if (A[first] is an identifier) return first ;
        else if (A[first] is an operator) {
                firstEnd = endPre(A, first+1, last);
                if (firstEnd = –1) return –1;
                else return endPre(A, firstEnd+1, last);
        } else return –1;
}
```

base case

끝

+ <*prefix*> <*prefix*>

핵심부

first

a

endPre(A, *first*, *last*)
{

base case

if (A[*first*] is an identifier) **return** *first* ;

else { // A[*first*] is an operator

firstEnd = endPre(A, *first*+1, *last*);

else return endPre(A, firstEnd+1, *last*);

}

}

last index

+ <*prefix*> <*prefix*>

first

\+ . . .

생각 훈련: Conversion from Prefix to Postfix

```
convert(pre)
{   // pre: a valid prefix expression
    // return the equivalent postfix expression
        ch = the 1st character of pre;
        Delete the 1st character from pre;
        if (ch is an identifier) return ch;
        else {  // ch is an operator
                postfix1 = convert(pre);
                postfix2 = convert(pre);
                return postfix1•postfix2•ch;   // concatenation
        }
}
```

<postfix> <postfix> +

pre[ ] + <prefix> <prefix>

pre[ ] <prefix> <prefix>

Recursion & Math.cal Induction

- Cost of Hanoi Tower: An example
  - Recursive relation for the # of moves
    - moves(N) = 2 • moves(N–1) + 1
    - N : # of discs

```
move(N, A, B, C)
{
        ….
        move(N –1, A, C, B)
        move(1, A, B, C)
        move(N –1, C, B, A);
}
```

Fact: moves(N) = $2^N - 1$

<Proof>

Basis:

moves(1) = 1 = $2^1 - 1$

Inductive hypothesis: Assume moves(k) = $2^k - 1$.

Inductive conclusion:

$$\begin{aligned} \text{moves}(k+1) &= 2\bullet\text{moves}(k) + 1 \\ &= 2(2^k - 1) + 1 \\ &= 2^{k+1} - 1 \quad \blacksquare \end{aligned}$$