The Eight Queens Problem

- **Problem**
  - Place eight queens on the chessboard so that no queen can attack any other queen
- **Strategy: guess at a solution**
  - There are 4,426,165,368 ways to arrange 8 queens on a chessboard of 64 squares

- **Providing organization for the guessing strategy**
  - Place queens one column at a time
  - If you reach an impasse, backtrack to the previous column

- **Backtracking**
  - A strategy for guessing at a solution and backing up when an impasse is reached
  - Recursion and backtracking can be combined to solve problems

- **An observation that eliminates many arrangements from consideration**
  - No queen can reside in a row or a column that contains another queen
  - Now: only 40,320 arrangements of queens to be checked for attacks along diagonals

Figure 6-1

a) Five queens that cannot attack each other, but that can attack all of column 6; b) backtracking to column 5 to try another square for the queen; c) backtracking to column 4 to try another square for the queen and then considering column 5 again
The Eight Queens Problem

- A recursive algorithm that places a queen in a column
  - Base case
    - If there are no more columns to consider
      - You are finished
  - Recursive step
    - If you successfully place a queen in the current column
      - Consider the next column
    - If you cannot place a queen in the current column
      - You need to backtrack

Defining Languages

- A language
  - A set of strings of symbols
  - Examples: English, Java
  - If a Java program is one long string of characters, the language Java Programs is defined as
    \[ \text{Java Programs} = \{ \text{strings w : w is a syntactically correct Java program} \} \]

Defining Languages

- A language does not have to be a programming or a communication language
  - Example
    - The set of algebraic expressions
      \[ \text{Algebraic Expressions} = \{ w : w \text{ is an algebraic expression} \} \]

The Basics of Grammars

- Grammar
  - States the rules for forming the strings in a language
- Benefit of recursive grammars
  - Ease of writing a recognition algorithm for the language
    - A recognition algorithm determines whether a given string is in the language
- Symbols used in grammars
  - \( x \mid y \) means \( x \) or \( y \)
  - \( x \circ y \) means \( x \) followed by \( y \)
  - In \( x \circ y \), the symbol \( \circ \) means concatenate, or append
  - \( < \text{ word } > \) means any instance of word that the definition defines
The Basics of Grammars

- **Java identifiers**
  - A Java identifier begins with a letter and is followed by zero or more letters and digits

```
Letter → Letter Digit
```

![Figure 6-3](image)
A syntax diagram for Java identifiers

The Basics of Grammars

- **Recognition algorithm**
  - `isId(w)` if (w is of length 1) {
    - 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
  }

Two Simple Languages:
Palindromes

- A string that reads the same from left to right as it does from right to left
- Examples: radar, deed
- **Language**
  Palindromes = \{ w : w reads the same left to right as right to left \}

Palindromes

- **Grammar**
  - `<pal>` = empty string | `<ch>` | a `<pal>` a | b `<pal>` b | ... | Z `<pal>` Z
  - `<ch>` = a | b | ... | z | A | B | ... | Z

```
<pal> = empty string | <ch> | a <pal> a | b <pal> b | ... | Z <pal> Z
<ch> = a | b | ... | z | A | B | ... | Z
```

Palindromes

- **Recognition algorithm**
  - `isPal(w)` if (w is the empty string or w is of length 1) {
    return true
  } else if (w’s first and last characters are the same letter) {
    return isPal(w minus its first and last characters)
  } else {
    return false
  }
Strings of the form $A^nB^n$

- $A^nB^n$:
  - The string that consists of $n$ consecutive A’s followed by $n$ consecutive B’s
- Language
  $$L = \{w : w \text{ is of the form } A^nB^n \text{ for some } n \geq 0\}$$
- Grammar
  $$\begin{align*}
  \langle \text{legal-word} \rangle &= \text{empty string} \mid A \langle \text{legal-word} \rangle B \\
  
  \end{align*}$$

Algebraic Expressions

- Three languages for algebraic expressions
  - Infix expressions
    - An operator appears between its operands
    - Example: $a + b$
  - Prefix expressions
    - An operator appears before its operands
    - Example: $+ a b$
  - Postfix expressions
    - An operator appears after its operands
    - Example: $a b +$

Recognition algorithm

```java
isAnBn(w) {
  if (the length of w is zero) {
    return true
  } else if (w begins with the character A and ends with the character B) {
    return isAnBn(w minus its first and last characters)
  } else {
    return false
  }
}
```

To convert a fully parenthesized infix expression to a prefix form

- Move each operator to the position marked by its corresponding open parenthesis
- Remove the parentheses
- Example
  - Infix expression: $((a + b) \times c$
  - Prefix expression: $+ a b c$

Prefix and postfix expressions

- Never need
  - Precendence rules
  - Association rules
  - Parentheses
- Have
  - Simple grammar expressions
  - Straightforward recognition and evaluation algorithms
Prefix Expressions

- **Grammar**
  < prefix > = < identifier > | < operator > < prefix >
  < operator > = + | - | * | /
  < identifier > = a | b | ... | z

- **A recognition algorithm**
  ```plaintext
  isPre()
  size = length of expression strExp
  lastChar = endPre(0, size - 1)
  if (lastChar >= 0 and lastChar == size-1) {return true}
  else if (ch is an identifier) {
    return value of the identifier
  } else if (ch is an operator named op) {
    operand1 = evaluatePrefix(strExp)
    operand2 = evaluatePrefix(strExp)
    return operand1 op operand2
  }
  ```

Postfix Expressions

- **Grammar**
  < postfix > = < identifier > | < postfix > < postfix > < operator>
  < operator > = + | - | * | /
  < identifier > = a | b | ... | z

- **At high-level, an algorithm that converts a prefix expression to postfix form**
  ```plaintext
  if (exp is a single letter) {
    return exp
  } else {
    return postfix(prefix1) + postfix(prefix2) + operator
  }
  ```

Fully Parenthesized Expressions

- **To avoid ambiguity, infix notation normally requires**
  - Precedence rules
  - Rules for association
  - Parentheses

- **Fully parenthesized expressions do not require**
  - Precedence rules
  - Rules for association

- **An algorithm that evaluates a prefix expression**
  ```plaintext
  evaluatePrefix(strExp)
  ch = first character of expression strExp
  Delete first character from strExp
  if (ch is an identifier) {
    return value of the identifier
  } else if (ch is an operator named op) {
    operand1 = evaluatePrefix(strExp)
    operand2 = evaluatePrefix(strExp)
    return operand1 op operand2
  }
  ```

- **A recursive algorithm that converts a prefix expression to postfix form**
  ```plaintext
  convert(pre)
  ch = first character of pre
  Delete first character of pre
  if (ch is a lowercase letter) {
    return ch as a string
  } else {
    postfix1 = convert(pre)
    postfix2 = convert(pre)
    return postfix1 + postfix2 + ch
  }
  ```

- **Fully parenthesized expressions**
  - A simple grammar
    < infix > = < identifier > | ( < infix > < operator > < infix > )
    < operator > = + | - | * | /
    < identifier > = a | b | ... | z
  - Inconvenient for programmers
The Relationship Between Recursion and Mathematical Induction

- A strong relationship exists between recursion and mathematical induction.
- Induction can be used to
  - Prove properties about recursive algorithms.
  - Prove that a recursive algorithm performs a certain amount of work.

The Correctness of the Recursive Factorial Method

- Pseudocode for a recursive method that computes the factorial of a nonnegative integer n:

  ```java
  fact(n) {
    if (n is 0) {
      return 1
    } else {
      return n * fact(n - 1)
    }
  }
  ``

- Solution to the Towers of Hanoi problem:

  ```java
  solveTowers(count, source, destination, spare) {
    if (count is 1) {
      Move a disk directly from source to destination
    } else {
      solveTowers(count-1, source, spare, destination)
      solveTowers(1, source, destination, spare)
      solveTowers(count-1, spare, destination, source)
    }
  }
  ``

The Cost of Towers of Hanoi

- Question
  - If you begin with N disks, how many moves does `solveTowers` make to solve the problem?

- Let
  - `moves(N)` be the number of moves made starting with N disks.

- When N = 1
  - `moves(1) = 1`
The Cost of Towers of Hanoi

- A closed-form formula for the number of moves that `solveTowers` requires for N disks
  \[ \text{moves}(N) = 2^N - 1, \text{for all } N \geq 1 \]
- Induction on N can provide the proof that
  \[ \text{moves}(N) = 2^N - 1 \]

Summary

- Backtracking is a solution strategy that involves both recursion and a sequence of guesses that ultimately lead to a solution
- A grammar is a device for defining a language
  - A language is a set of strings of symbols
  - A recognition algorithm for a language can often be based directly on the grammar of the language
  - Grammars are frequently recursive

Summary

- Different languages of algebraic expressions have their relative advantages and disadvantages
  - Prefix expressions
  - Postfix expressions
  - Infix expressions
- A close relationship exists between mathematical induction and recursion
  - Induction can be used to prove properties about a recursive algorithm