ECE 641
Advanced Topics in Supervisory Control for Discrete Event Systems

Lecture 4

Associate Prof. Dr. Klaus Schmidt

Department of Mechatronics Engineering – Çankaya University

PhD Course in Electronic and Communication Engineering
Credits (3/0/3)
Course webpage: http://ece641.cankaya.edu.tr/

Algorithm: Basics

Algorithm Definition

- A computable set of steps to achieve a desired result
- Precisely specified using an appropriate mathematical formalism (such as a programming language)

Efficiency of an Algorithm

- Less consumption of computing resources (execution time (CPU cycles), memory)
  $\Rightarrow$ We will focus on time efficiency

Comparison of Algorithms

- Question: Assume two algorithms that accomplish the same task
  $\Rightarrow$ Which one is better?

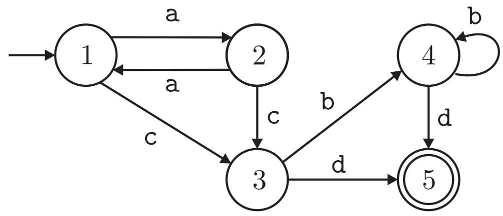
Algorithm: Example

Algorithm

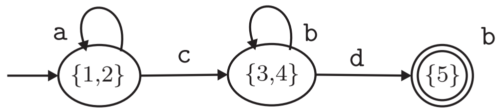
1. Initialize: $q_1 = X_M$ and $q_2 = X - X_m$
2. Formulate table with all state pairs
3. Mark all pairs with one marked and one unmarked state as distinguishable
4. Mark all pairs with different outgoing transitions as distinguishable
5. Mark pairs that lead to distinguishable states with the same event as distinguishable
6. Repeat the previous step until all state pairs are investigated

   $\Rightarrow$ Combine states that are indistinguishable

Automata Graph



2				
3	d	d		
4	d	d		
5	d	d	d	d
	1	2	3	4



Klaus Schmidt
Department of Electronic and Communication Engineering – Çankaya University

Algorithm: Example

State Minimization

Gap 1

Klaus Schmidt
Department of Electronic and Communication Engineering – Çankaya University

Algorithm: Analysis

Resources

- We want to predict the resources that the algorithm requires
- Resources: Memory, processor, other hardware but MOSTLY TIME

General Notions

- Run time: Time until an algorithm produces a result
  $\Rightarrow$ Run time of a given algorithm generally grows by the size of the input
- Denote size as n: number of items to be processed, number of bits to represent the relevant quantities, number of states, etc.
- Growth rate: How quickly the time of an algorithm grows as a function of n

Algorithm: Type of Analysis

Worst Case Analysis

- Largest possible running time of algorithm on input of a given size.
- Provides an upper bound on running time

$\Rightarrow$ Absolute guarantee for the longest run-time for any input

Best Case Analysis

- Provides a lower bound on run-time
- Input is the one for which the algorithm runs the fastest

Average Case Analysis

- Obtain bound on run-time of algorithm on random input as a function of input size
- Hard (or impossible) to accurately model real instances by random distributions (Algorithm tuned for a certain distribution may perform poorly on other inputs)

Klaus Schmidt
Department of Electronic and Communication Engineering – Çankaya University

Algorithm: Sequential Search

Illustration

Gap 2

Algorithm: Binary Search

Illustration

Gap 3

Algorithm: Discussion

Illustration

Gap 4

Klaus Schmidt
Department of Electronic and Communication Engineering – Çankaya University

Algorithm: Conclusions

Facts

- Run-time depends on the input size n
- Usually run-time is fixed for a certain n
- Different algorithms might have different run-times
- Some algorithms might be better for some inputs and worse for others
- Exact run-time depends on the algorithm but also on the processor where it runs

General Analysis

- We will look at the trend in run-time versus input size rather than exact time

$\Rightarrow$ Computational Complexity

Klaus Schmidt
Department of Electronic and Communication Engineering – Çankaya University

Computational Complexity: Basics

Task

- Compares growth of two functions
- Independent of constant multipliers and lower-order effects

Metrics

- Big-O Notation: $O(\bullet)$
- Big-Omega Notation: $\Omega(\bullet)$
- Big-Theta Notation: $\Theta(\bullet)$

Properties

- Allow us to evaluate algorithms
- Has precise mathematical definition
- Used in a sense to put algorithms into families
- May often be determined by inspection of an algorithm

Klaus Schmidt
Department of Electronic and Communication Engineering – Çankaya University

Computational Complexity: Big-O Notation

Definition

Function $f(n)$ is denoted as $O(g(n))$ for a function $g(n)$ if there exists a constant K and some value n_0 such that for all $n \geq n_0$

$$f(n) \leq K \cdot g(n).$$

This means, as $n \to \infty$, $f(n)$ is upper-bounded by $K \cdot g(n)$.

Useful Choices for $g(n)$

- $\log n$ (recall that $\log_a n) = k \cdot \log_b n$ for any $a, b \in \mathbb{R}$)
- n^k for $k \in \mathbb{N}_0$ (polynomial)
- k^n for some $k \in \mathbb{R}$ (exponential)

Properties

- Big-O-Notation establishes the worst-case performance
- Helps compare and see which algorithm has better performance

Klaus Schmidt
Department of Electronic and Communication Engineering – Çankaya University

Computational Complexity: Big-O Notation

Illustration

Gap 5

Klaus Schmidt
Department of Electronic and Communication Engineering – Çankaya University

Computational Complexity: Big-O Notation

State Minimization

Gap 6

Klaus Schmidt
Department of Electronic and Communication Engineering – Çankaya University

Computational Complexity: Hopcroft's Algorithm

Initialization: $P := \{X_{\rm m}, X \setminus X_{\rm m}\}$; *Waiting* $= \{X_{\rm m}\}$
while *Waiting* $\neq \emptyset$
    Choose and remove a set A from *Waiting*
    for each $\sigma \in \Sigma$
        Let Q be the state set for which a transition on σ leads to A
        for each set $Y \in P$ for which $Q \cap Y \neq \emptyset$
            Replace Y in P by the two sets $Q \cap Y$ and $Y \setminus Q$
            if Y belongs to *Waiting*
                Replace Y in *Waiting* by $Q \cap Y$ and $Y \setminus Q$
            else
                if $|Q \cap Y| \leq |Y \setminus Q|$
                    add $Q \cap Y$ to *Waiting*
                else
                    add $Y \setminus Q$ to *Waiting*
Result: Set P contains the sets of equivalent states

Klaus Schmidt
Department of Electronic and Communication Engineering – Çankaya University

Computational Complexity: Big-O Notation

State Minimization

Gap 7

Klaus Schmidt
Department of Electronic and Communication Engineering – Çankaya University

Computational Complexity: Big-Omega/Theta Notation

Definition

Function $f(n)$ is $\Omega(g(n))$ if there exists a constant K and some n_0 such that for all $n \geq n_0$

$$K \cdot g(n) \leq f(n).$$

That is, as $n \to \infty$, $f(n)$ is lower-bounded by $K \cdot g(n)$.

Definition

Function $f(n)$ is $\Theta(g(n))$ if there exist constants K_1 and K_2 and some n_0 such that for all $n \geq n_0$

$$K_1 \cdot g(n) \leq f(n) \leq K_2 \cdot g(n).$$

That is, as $n \to \infty$, $f(n)$ is upper and lower bounded by some constants times $g(n)$.

Computational Complexity: Big-Omega/Theta Notatioin

Illustration

Gap 8

Computational Complexity: Common Asymptotic Bounds

Examples

- Polynomials: $a_0 + a_1\,n + \cdots + a_d\,n^d$ is $\theta(n^d)$ if $a_d > 0$
  $\Rightarrow$ Polynomial time: run-time is $O(n^d)$ for some constant d independent of the input size n
- Logarithms: $O(\log_a n) = O(\log_b n)$ for any constants $a, b > 0$
- Logarithm grows slower than every polynomial
  $\Rightarrow$ for each $d > 0$, $\log n = O(n^d)$
- Exponential time: run-time is $O(k^n)$ for some constant k
- Every exponential grows faster than every polynomial
  $\Rightarrow$ For every $k > 1$ and every $d > 0$, $n^d = O(k^n)$

Computational Complexity: Examples

Synchronous Composition

Gap 9

Natural Projection

Gap 10

Diagnosability

Gap 11