

CSCI 256: Theory of Computing

CHAPTER 1 Introduction to Automata and Proof Techniques

Dr. Benjoe A. Juliano

Tel. 530 898-4619 (office)
530 898-6442 (dept)
Fax. 530 898-5995

Juliano@ecst.csuchico.edu
http://www.ecst.csuchico.edu/~juliano

B. Juliano © 2002, based on notes by J. Ullman



1

Why Study Automata Theory?

- Introduction to Finite Automata
- Automata Theory** – study of abstract computing devices, or “machines.”

TURING MACHINES
AREN'T REAL MACHINES...
THEY'RE ABSTRACT
MACHINES, EXISTING ONLY
IN THEORY...



If you dream it... we'll build it!

- In the 1930's, **Alan Turing** studied and described precisely the boundary between what a(n) (*abstract*) computing machine could do and what it could not do.

B. Juliano © 2002, based on notes by J. Ullman



2

Why Study Automata Theory?

- Introduction to Finite Automata



- In the late 1950's, the linguist **Noam Chomsky** began the study of formal “grammars.”



- In 1969, **Stephen Cook** defined “intractable” or “NP-hard” problems – problems that can in principle be solved, but in practice take so much time that computers are useless for all but very small instances of the problem.

B. Juliano © 2002, based on notes by J. Ullman



3

Structural Representations

- Grammars**
 - Useful models when designing software that processes data with a recursive structure.
 - Example: parser for a compiler.
- Regular Expressions**
 - Denote the structure of data; in particular, text strings.
 - Example: Unix-style regular expressions

' ([A-Z][a-z]*[])*[A-Z][A-Z]'

B. Juliano © 2002, based on notes by J. Ullman



4

Automata and Complexity

- **Decidability** (see Chapter 9)
 - What can a computer do at all?
 - Problems that can be solved by computer are called “decidable.”
- **Intractability** (see Chapter 10)
 - What can a computer do efficiently?
 - Problems that can be solved by a computer using no more time than some slowly growing (polynomial) function of the size of the input are called “tractable.”

B. Juliano © 2002, based on notes by J. Ullman

5

Introduction to Formal Proof

- **Deductive Proofs**
 - **Form:** From some initial statement H , called the *hypothesis* or the *given statement(s)*, provide a sequence of statements whose truth leads to a *conclusion* statement C ; C is *deduced* from H .
 - Typically given as a theorem of the form, “If H then C ,” as in
 - Theorem: If $x \geq 4$, then $2^x \geq x^2$.
 - Perhaps, the most common type of proof ...

B. Juliano © 2002, based on notes by J. Ullman

6

Introduction to Formal Proof

- **Reduction to Definitions**
 - It is sometimes helpful to convert all terms in the hypothesis to their definitions ...
- **Note:** Given the statement “If H then C ,”
 - *contrapositive*: “If not C then not H ”
 - *converse*: “If C then H ”
 - *contradiction*: “ H and not C implies false”
- **Note:** To prove that a statement S is not a theorem, it suffices to show a *counterexample*.

B. Juliano © 2002, based on notes by J. Ullman

7

Proof by Mathematical Induction

- **Form:** Prove a statement $S(X)$ about a family of objects X (e.g. integers, trees) in three parts:
 - **Basis:** Show $S(X)$ holds for one or several small values of X directly.
 - **Inductive Hypothesis:** Assume $S(Y)$ holds for values $Y \leq n$.
 - **Inductive Step:** Show that $S(n+1)$ holds using the inductive hypothesis.

B. Juliano © 2002, based on notes by J. Ullman

8

Proof by Mathematical Induction

- **Example:**

- Prove that a binary tree with n leaves has $2n-1$ nodes.
 - Formally, $S(T)$: If T is a binary tree with n leaves, then T has $2n-1$ nodes.
 - Induction is on the number of nodes in T .

Basis: If T is a one-node tree, then has only one leaf; $1 = 2 \times 1 - 1$, so OK.

Inductive Hypothesis: Assume $S(U)$ holds for all binary trees U with at most k leaves
Hence, if U is a binary tree with k leaves, U has $2k-1$ nodes.

B. Juliano © 2002, based on notes by J. Ullman



9

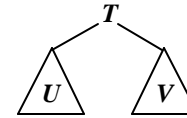
Proof by Mathematical Induction

- **Example (continued):**

- Prove that a binary tree with n leaves has $2n-1$ nodes.

Inductive Step: Consider a binary tree T with $k+1$ leaves ...

- T must have two subtrees U and V .
- If U and V have u and v leaves, respectively, then T has $u + v = k+1$ leaves.



B. Juliano © 2002, based on notes by J. Ullman



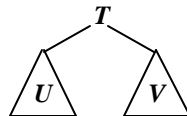
10

Proof by Mathematical Induction

- **Example (continued):**

Inductive Step: Consider a binary tree T with $k+1$ leaves ...

- By the inductive hypothesis, U and V have $2u-1$ and $2v-1$ nodes, respectively.
- Then T has $1 + (2u-1) + (2v-1)$ nodes.
 - T has $2(u+v)-1$ nodes.
 - T has $2(k+1)-1$ nodes, proving the inductive step.



B. Juliano © 2002, based on notes by J. Ullman



11

Proof by Equivalence

- **Form:** Prove “ X if and only if Y .”
- The proof has to be done in two steps:
 - Prove the **if-part**: Assume Y and prove X .
 - Prove the **only-if-part**: Assume X and prove Y .
- Example: Equivalence of sets S and T can be shown when x is in S if and only if x is in T .
 - Assume x is in S ; prove x is in T .
 - Assume x is in T ; prove x is in S .

B. Juliano © 2002, based on notes by J. Ullman



12

Proof by Equivalence

- **Form:** Prove “ X if and only if Y .”
- Remember:
 - The *if-part* and *only-if-part* are *converses* of each other.
 - One part, say “if X then Y ,” says nothing about whether Y is true when X is false.
 - An alternate, equivalent form of “if X then Y ” is “if not Y then not X ” – the latter is a *contrapositive* of the former.

B. Juliano © 2002, based on notes by J. Ullman

13

Proof by Equivalence

- Example: Balanced Parentheses
- Two ways to define “balanced parentheses”:
1. **Grammatically (GB):**
 - a) The empty string, ϵ , is balanced.
 - b) If w is balanced, then (w) is balanced.
 - c) If w and x are balanced, then so is wx .
 2. **By Scanning (SB):** w is balanced if and only if
 - a) w has an equal number of left and right parentheses
 - b) Every prefix of w has at least as many left as right parentheses

B. Juliano © 2002, based on notes by J. Ullman

14

Proof by Equivalence

- **Theorem:** A string of parentheses w is **GB** if and only if w is **SB**.
- Proof (*if-part*):
 - Assume w is **SB**; prove it is **GB** by induction on $|w|$, the length of string w .

Basis: If $|w|=0$ (i.e. w is ϵ), then w is **GB** by **GB** rule a.

Inductive Hypothesis: Suppose the statement “if **SB** then **GB**” is true for all w where $|w|<n$.

B. Juliano © 2002, based on notes by J. Ullman

15

Proof by Equivalence

- Inductive Step:** Show that the statement “if **SB** then **GB**” is true for all w where $|w|\geq n$.
- **Case 1:** w is not ϵ , but has no nonempty prefix that has an equal number of (and). Then, w must begin with (and end with); i.e. $w=(x)$.
 - x must be **SB** (why?).
 - x is **GB** by the *Inductive Hypothesis*.
 - (x) is **GB** by **GB** rule b; but $(x)=w$, so w is **GB**.

B. Juliano © 2002, based on notes by J. Ullman

16

Proof by Equivalence

Inductive Step: Show that the statement “if **SB** then **GB**” is true for all w where $|w| \geq n$.

- Case 2: $w=xy$, where x is the shortest, nonempty prefix of w with an equal number of (and), and $y \neq \epsilon$.
 - x and y are both **SB** (why?).
 - x and y are both **GB** by the *Inductive Hypothesis*.
 - w is **GB** by **GB** rule c.

Proof by Equivalence

- *Theorem:* A string of parentheses w is **GB** if and only if w is **SB**.

- Proof (*only-if-part*):

- Assume w is **GB**; prove it is **SB** by induction on $|w|$, the length of string w .

Basis: If $|w|=0$ (i.e. w is ϵ), then clearly w obeys the conditions for being **SB**.

Inductive Hypothesis: Suppose the statement “**SB** only if **GB**” is true for all $w \neq \epsilon$ where $|w| < n$.

Proof by Equivalence

Inductive Step: Show that the statement “**SB** only if **GB**” is true for all w where $|w| \geq n$.

- Case 1: w is **GB** by **GB** rule b; i.e. $w=(x)$ and x is **GB**.
 - x is **SB** by *Inductive Hypothesis*.
 - Since x has equal number of (and), so does (x) .
 - Since x has no prefix with more (than), then so does (x) .

Proof by Equivalence

Inductive Step: Show that the statement “**SB** only if **GB**” is true for all w where $|w| \geq n$.

- Case 2: $w \neq \epsilon$ is **GB** by **GB** rule c; i.e. $w=xy$ and both x and y are **GB**.
 - x and y are **SB** by *Inductive Hypothesis*.
 - (*Aside*) Trickier than it looks: we have to argue that neither $x \neq \epsilon$ nor $y \neq \epsilon$, because if one were, the other would be w , and this rule application could not be the one that first shows w to be **GB**.
 - Since each of x and y have equal number of (and), so does xy .

Proof by Equivalence

Inductive Step: Show that the statement “**SB** only if **GB**” is true for all w where $|w| \geq n$.

- Case 2: *continued ...*
 - If w had a prefix with more) than (, that prefix would either be a prefix of x (contradicting the fact that x has no such prefix) or it would be x followed by a prefix of y (contradicting the fact that y also has no such prefix).
 - (*Aside*) Above is an example of *proof by contradiction* – we assumed our conclusion about w was false and showed it would imply something that we know is false.

B. Juliano © 2002, based on notes by J. Ullman

21

Languages

- **Alphabet** = finite set of symbols
 - Examples: $\{0,1\}$ (the *binary* alphabet)
 - $\{a,b,c,\dots,z\}$
- **String** = finite sequence of symbols chosen from some alphabet
 - Examples: **01101**
 - abracadabra*
- **Language**
 - = set of strings chosen from some alphabet

B. Juliano © 2002, based on notes by J. Ullman

22

Languages

- **Powers of an alphabet**
 - If Σ is an alphabet, define S^k to be the set of strings of length k , consisting of symbols in Σ .
 - The set of *all* strings over Σ is denoted S^* ; and,

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots$$
 - The set of *nonempty* strings over Σ is denoted S^+ ; further,

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \dots$$

$$\Sigma^* = \Sigma^+ \cup \{\epsilon\}$$

B. Juliano © 2002, based on notes by J. Ullman

23

Languages

- **Note:**
 - A language may be infinite, but there is some finite set of symbols of which all its strings are composed from.
- **Examples:**
 - The set of all *binary* strings consisting of some number of 0's followed by an equal number of 1's; that is, $\{\epsilon, 01, 0011, 000111, \dots\}$.
 - C (the set of compilable C programs)
 - English

B. Juliano © 2002, based on notes by J. Ullman

24

Languages

- **More Abstract Examples:**
 - The set of binary numbers whose value is *prime*; that is, $\{10, 11, 101, 111, 1011, \dots\}$
 - Σ^* is a language for any alphabet Σ
 - \emptyset , the empty language, is a language over any alphabet.
 - $\{\epsilon\}$, the language consisting of only the empty string, is also a language over any alphabet.

Problems

- In automata theory, a *problem* is the question of deciding whether a given string is a member of some particular language.
- Hence, if Σ is an alphabet, and L is a language over Σ , then the problem L is:

Given a string $w \in \Sigma^*$, decide if $w \in L$.

Copyright and Intellectual Property Notice

This document and its contents are the *Intellectual Property* (IP) of Dr. Benjoe A. Juliano of the Department of Computer Science at California State University, Chico (CSUC). Dr. Juliano claims exclusive moral rights of ownership under current *Copyright Laws* (Title 17 of the United States Code and 1998 Digital Millennium Copyright Act) and *IP Policies/Guidelines* (CSUC EM83-08, EM97-07, and Article 39 of the CFA/CSU Contract) including, but not limited to:

- the exclusive right to copy, reproduce, and/or distribute this document;
- the right to be identified as the creator of this work (the right of attribution);
- the right to take action against false attribution; and
- the right to object to derogatory treatment of this work (the right of integrity).