ICS 6B
Boolean Algebra & Logic

Lecture Notes for Summer Quarter, 2008

Michele Rousseau

Set 10 – Ch. 12.1, 12.2, 12.3
(Some slides inspired and adapted from Alessandra Pantano)

Announcements

- **Final on Tuesday in class**
- **Comprehensive**
  - Similar to quizzes
  - A little more focus on last lecture
- **Let me know if anyone wants some suggested problems**
- **Let me know if anyone wants to attend a final review on Monday**
  - I will announce if I am able to get a room
- **Quiz #4 regrades due today**

Today's Lecture

- **Chapter 12(12.1, 12.2, 12.3)**
  - Languages and Grammars (12.1)
  - Finite State Machines with Outputs (12.2)
  - Finite State Machines with No Output (12.3)

Chapter 12: Section 12.1

Languages and Grammars

Preliminary Definitions:

- **Vocabulary: a not empty finite set of symbols**
  - Also called an "alphabet"
- **Sentence: a finite string of symbols**
  - Also called a "word"
- **Language: any collection of sentences**

A sentence can be the empty string of symbols.
In this case the sentence is denoted by λ(lambda)

Examples

Example 1
- V={a,B,C,d} is a vocabulary
- aBaaC and dBaB are sentences
- The set {aBaaC, dBaB} is a language

Example 2
- V={English words}
- Any finite sequence of words is a sentence
- The set {"The Blue The", "sky is blue", "goat two"} is a language

Example 3
- V={letters in the alphabet}
- Any sequence of the letters is a sentence
- The set {"blue", "sky", "the"} is a language

Grammar: Informal Definition

- A grammar is a way to construct a language from a given vocabulary

1. Choose a special element S in the vocabulary to begin with (S will be called the "start symbol")
2. Divide your set of symbols (i.e. your vocabulary) into two categories: the "terminal" and the "non-terminal" elements.
3. The non-terminal elements are the ones that can be replaced by other symbols. Strings containing non-terminal elements can be replaced by other strings.
4. Assign a set of rules (called "productions") to replace certain strings by other stings
5. Start from you "start element" S, and apply the rules ("productions") to create the sentences in the language.

Example

- Consider the vocabulary V={a,B,C,d}.
- Say that B is the start element.
- Choose {a,d} to be the terminal elements, and {B,C} to be the non-terminal elements.
- Allow the following replacements:
  - B→aC →aBCd → aBBcd →...
    - add
    - aaCCd →...

A more formal definition

A phrase structure grammar G=(V,T,S,P) consists of:

- A vocabulary v
  - i.e. a not empty set of symbols
- A subset T ⊆ V of terminal elements
  - (i.e. a subset of symbols that cannot be replaced by other symbols)
- A start element S
  - To begin our construction
- A finite set of productions
  - A finite set of rules that specify which strings of symbols can be replaced by other strings

Example 1

Consider the vocabulary V=T UN with

- T={a, the, fat, rabbit, mathematician, hops, eats}
- N={sentence, article, adjective, noun, verb}

- Let "sentence" be the start element.
- Allow the following replacements (productions)

1. sentence → (article) (adjective) (noun) (verb)
2. article → a
3. article → the
4. adjective → fat
5. noun → mathematician
6. noun → rabbit
7. verb → hops
8. verb → eats

Example 1 (2)

If we start from S=sentence and apply a sequence of production (until no further production is applicable), we produce the following valid sentences:

article adjective noun verb

- a fat mathematician hops
- the fat mathematician hops
- a fat rabbit hops
- the fat rabbit hops
- a fat mathematician eats
- the fat mathematician eats
- a fat rabbit eats
- the fat rabbit eats

I am only interested in the sentences that contain "terminal" symbols (those that can't be replaced)

Example 1 (3)

We obtained strings of the following form

S → (article) (adjective) (noun) (verb)

↓

(article) (fat) (noun) (verb)

The article can be replaced by "a" or "the"

The noun can be replaced by "rabbit" or "mathematician"

The verb can be replaced by "eats" or "hops"

Notice that we can't produce the string "a mathematician eats the rabbit"

By applying a sequence of productions?

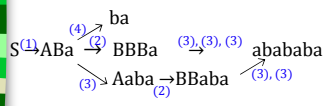
=> This is not a valid string

Example 2

G=(V,T,S,P) with

- V={a,b,A,B,S}
- T={a,b}
- S=the start symbol
- P={S →ABa, A →BB, B →ab, AB →b}
  (1) (2) (3) (4)

Lets see what are all the valid sentences that we can produce (by applying a sequence of productions to S)



Derivations

- **Let G=(V,T,S,P) be a phrase-structure grammar.**
- **Let w_0 and w_1 be strings over V.**
- **We say that w_1 is directly derivable from w_0 if w_1 can be obtained from w_0 by applying a *single production***
  More precisely, we can write w_0 and w_1 as a concatenation of
  $w_0 = l, z_0 r$, $w_1 = l, z_1 r$
  And there exists a production $z_0 \rightarrow z_1$.
  If w_1 is directly derivable from w_0, we write $w_0 \Rightarrow w_1$
- **We say that w_1 is derivable from w_0 if w_1 can be obtained from w_0 by applying a *sequence of productions.***
  More precisely, there exists strings $u_1, u_2, \ldots, u_k$ such that
  $w_0 \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow w_1$
  In this case we write $w_0 \overset{*}{\Rightarrow} w_1$
  The sequence of steps $w_0 \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow w_1$ is called a derivation.

For example

- **In example 2,**
  - Bbaba is derivable from Aba, and a derivation is:
    $Aaba \underset{(3)}{\Rightarrow} Aaba \underset{(2)}{\Rightarrow} BBaba$
  - Another derivation is
    $ABa \underset{(2)}{\Rightarrow} BBBa \underset{(3)}{\Rightarrow} BBaba$

Language of a Grammar

- **Let G=(V, T, S, P) be a grammar. The language of G - denoted L(G) - is the set of all sequences of terminal elements that can be derived from P:**
  $L(G)=\{w \in T^* : S \overset{*}{\Rightarrow} w\}$
- **To construct the language of G, you start from S and apply all possible derivations that lead to sentences which only contain the symbols in T.**
- **In Example 1, L(G) consists of exactly 8 sentences.**
- **In Example 2, L(G) consists of exactly 2 sentences**
  - ("ba" and "abababa")

2 Main Types of Problems:

1 – Given a grammar G, find L(G)

2 – Given a language L, find a grammar G such that L=L(G)

Type 1: Construct the Lang of a Grammar

Example 3

G ={V, T, S, P} with

- V={S, A, a, b}
- T={a, b}
- S= the start element
- P={S⟶aA, S ⟶b, A=aa}

Find L(G)

S ⟶aA ⟶aaa
↓
b

L(G) = {b, aaa}

Type 2: Given a Lang. construct a Grammar

Example:

Give a phrase-structure grammar that generates the set $L=\{0^n 1^n : n \geq 0\}$

We can use $G=(V,T,S,P)$ with

$V=\{0,1,S\}$

$T=\{0,1\}$

S= the start element

$P=\{S \rightarrow \lambda, S \rightarrow 0S1\}$

We get

$S \rightarrow 0S1 \rightarrow 00S11 = 0^2S1^2 \rightarrow 0^3S1^3 \rightarrow 0^4S1^4 \rightarrow \ldots$

$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow$

$\lambda = 0^0 1^0 \quad 01 \quad 0^2 1^2 \quad 0^3 1^3 \quad 0^4 1^4$

It is clear that G generates L

19

Type 2: Trick

We need to construct strings $0^n 1^n$ that contain the same number of 0's and 1's

So, we use the production $S \rightarrow 0S1$.

Notice that applying this production n-times gives $0^n S1^n$. This is almost what we want...

To get $0^n 1^n$ we need to apply the production $S \rightarrow \lambda$, where λ is the empty string.

Different types of Grammars

Type	AKA	Productions Allowed
0		Any production – no restrictions
1	Context – sensitive	Change 1 (non-terminal symbol) between 2 strings AND $S \rightarrow \lambda$ $lAr \rightarrow lwr$ A is a N (non-terminal) symbol lr is a string of T & N symbols (may be λ) w is a string of T & N symbols (can't be empty)
2	Context-free	Change **1** non-terminal symbol $A \rightarrow lr$ A is a single N terminal symbol lr is a string of T & N symbols (may be λ)
3	regular	Change **1** non-terminal symbol → very restricted how it can be changed $A \rightarrow aB$; $A \rightarrow a$ & $S \rightarrow \lambda$ A & B are non terminal, a is terminal

21

Different types of Grammars

- Grammars of type 2 and type 3 only allow you to change a single non-terminal symbol at a time.
- If a grammar is of type 2, a non-terminal symbol could be replaced by any string
- If a grammar is of type 3, a non-terminal symbol could only be replaced by a terminal symbol, or by a terminal symbol followed by a non-terminal symbol.
  - You can send S to the empty string
- If a grammar is of type 0, everything is allowed.

Note: if a grammar is of type r, then it is also of type r+1

Example

- V={S,A,B,a,b}
- T={a,b}
- P=
  - (1) $S \rightarrow Aba$, $AB \rightarrow a$,
  - (2) $S \rightarrow \lambda$, $aAb \rightarrow aab$
  - (3) $S \rightarrow \lambda$, $A \rightarrow aa$
  - (4) $S \rightarrow \lambda$, $A \rightarrow b$
- G1) type 0, but not type 1
  - AB is 2 N symbols
- G2) type 1, but not type 2
  - Not a single N symbol (has some T symbols too)
- G3) type 2, but not type 3
  - Goes to two T & N strings

Type	AKA	Productions Allowed
0		no restrictions
1	Context – sensitive	$S \rightarrow \lambda$ $lAr \rightarrow lwr$
2	Context-free	$A \rightarrow lr$
3	regular	$A \rightarrow aB$; $A \rightarrow a$ & $S \rightarrow \lambda$

G4) type 3

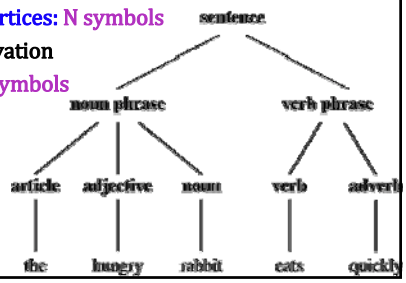
23

Context-free (Type 2) Grammars

- Most programming languages are in defined using context-free or regular grammars.
- For the remainder of this section we will focus on type 2 grammars.
- Type 2: $A \rightarrow lr$
  - (A is N and lr is a string of N & T)

Derivation trees

- A derivation generated by a context-free grammar can be represented graphically using an ordered rooted tree → a derivation tree.
- Internal vertices: N symbols in the derivation
- Leaves: T symbols



Example

- Let G= (V,T,S,P) with
  - V={a,b,c,A,B,C,S}
  - T={a,b,c}
  - P:

S⟶AB

1. A ⟶Ca
2. B ⟶Ba
3. B ⟶Cb
4. B ⟶b
5. C ⟶cb
6. C ⟶b

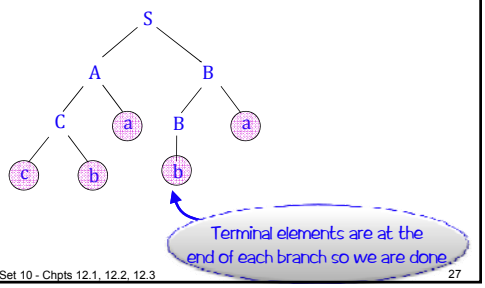
Consider the string cbaba, which is obtained from S by applying the following set of productions

S ⟶ AB ⟶CaBa ⟶cbaba

We want to represent the derivation S⇒ cbaba, using a tree

Example: S ⟶ AB ⟶CaBa ⟶cbaba

- On the top we start with the symbol S. S will be replaced by the string AB. We put a brand for each of these two symbols:



Backus-Naur Form

Alternative way to list productions in a type 2 grammar

1. Collect all the productions that give replacements of the same symbol
   (e.g. B ⟶Ba, B ⟶Cb, B ⟶b)
2. Use := instead of ⟶
   list the RHS of all the productions in the same group
   (e.g. B := Ba | Cb | b)
3. Enclose each N element in brackets (e.g. <B>)

In our example <B>:=<B>a|<C>b|b

Example

- V={a,b,c,A,B,C,S}, T={a,b,c} and P:

1. S⟶AB
2. A ⟶Ca
3. B ⟶Ba
4. B ⟶Cb
5. B ⟶b
6. C ⟶cb
7. C ⟶b

- In Backus=Naur Form
  - <S>:= <A><B>
  - <A>:=<C>a
  - <B>:=<B>a|<C>b|b
  - <C>:=cb|b

Chapter 12: Section 12.2

Finite State Machines with Outputs

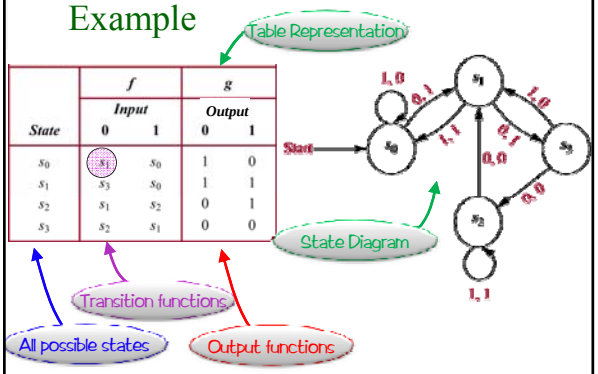
Finite State Machines with Outputs

- **A finite state machine with outputs:**

  $M=(S, I, O, f, g, s_0)$ consists of:

  1) S → a finite set of states
  2) I → a finite set of inputs
  3) O → a finite set of outputs
  4) f → a transition function
     => assigns each pair (state, input) a new state
  5) g → an output function
     => assigns each pair (state, input) an output and an initial state (s_0)

Example



Example: Vending Machine

- **Accepts nickels (5c)**
- **Sells orange juice (OJ) and apple juice (AJ)**
- **The price of an OJ or an AJ is 15c**
- **If you deposit more than 15c, the machine will immediately return the change**
- **The customer can press an "O button" for OJ or the "A" button for apple juice**

Vending Maching I/O and States

Possible inputs

- Nickel (5)
- O-button (O)
- A-button (A)

Possible outputs

- nothing (N)
- Nickel (5)
- OJ
- AJ

Possible states

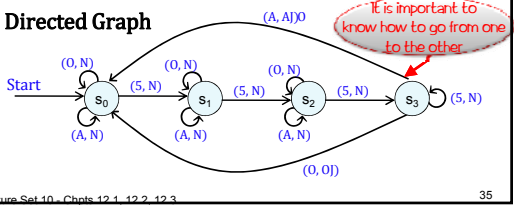
- initial state s_0 (no money in the machine)
- state s_1 (5c in the machine)
- state s_2 (10c in the machine)
- state s_3(15 in the machine)

Vending machine (2)

- **Table**

	States	Transition – f(s,i) 5	O	A	Output – g(s,i) 5	O	A
0	s_0	s_1	s_0	s_0	N	N	N
5c	s_1	s_2	s_1	s_1	N	N	N
10c	s_2	s_3	s_2	s_2	N	N	N
15c	s_3	s_3	s_0	s_0	5	OJ	AJ

- **Directed Graph**



Chapter 12: Section 12.3

Finite State Machines with No Output

Preliminaries on Strings

Definition:

If x and y are strings of symbols in a vocabulary V (i.e. $x \in V^*$ and $y \in V^*$), the concatenation of x and y is the string xy

- Eg. If x=011 and y=11 then xy=0111
- Notice that yx=11011 is **NOT** the same as xy

Definition:

If $A \subseteq V^*$ and $B \subseteq V^*$ are sets of strings the concatenation of A and B is the set

$AB=\{xy : x \in A,\ y \in B\}$

- Eg. A= {0, 11}
- B= {110, 10, 1}
- => AB={0110,010,01,11110,1110,111}

Prelims on Strings (2)

Similarly, one can define the concatenation of A with itself n-Times: $A^n = \underbrace{AA\ldots A}_{n \text{ times}}$

This is defined recursively by setting $A^0 = \{\lambda\}$ and $A^n = A^{n-1}A$, for all $n \geq 1$

So, if A={1,00}, A^2 ={11, 100, 011, 0000} and A^3={111, 1001, 0011, 00001, 1100, 10000, 00100, 000000}

The Kleen closer of a if the set $A^* = \bigcup_{n \geq 0} A^n$

Elements of A^* are concatenations of arbitraily many strings from A.

Examples

- $A=\{0\} \longrightarrow A^* = \{0^n : n \geq 0\}$
- $A=\{0,1\} \longrightarrow A^* = \{\text{all possible bit-strings}\}$
- $A=\{1,1\} \longrightarrow A^* = \{(11)^n : n \geq 0\} = \{1^{2m} : m \geq 0\}$

FSM with Output
AKA Finite State Automata

A finite state automata is similar to a FSM with outputs except that

- There are no outputs
  - In particular, there is no output function
- There are some final states.

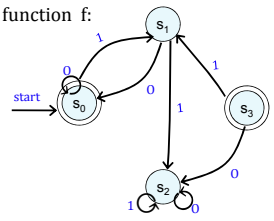
To define an automaton, we give:

- A finite set of states S
- A finite set of inputs I
- A transition function f: SxI $\longrightarrow$S
  - associates each (state, input) pair a new input.
- An initial state s_0
- A finite set of final states F

Example

- Consider the automaton (S, I, f, s_0, F) with
  - $S=\{s_0, s_1, s_2, s_3\}$, $I=\{0, 1\}$, $F=\{s_0, s_3\}$
  - And the transition function f:

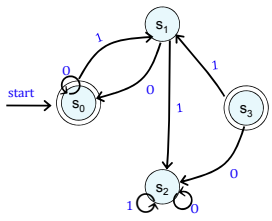
States	Inputs 0	1
s_0	s_0	s_1
s_1	s_0	s_2
s_2	s_2	s_2
s_3	s_2	s_1



Note that we can extend the transition function. It accepts a string of inputs(instead of a s state and a single input), and returns a new state

Example

- $f(s_0, 101) = s_1$
- $f(s_2, 01001) = s_2$

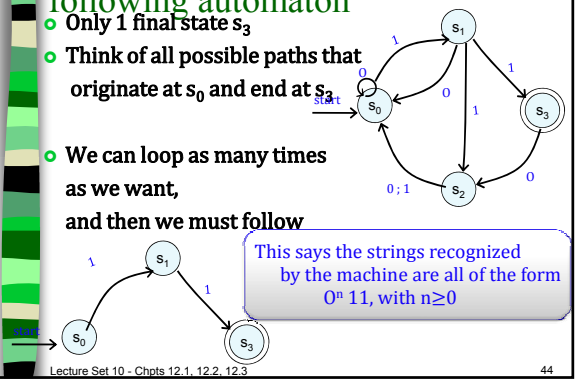


Language Recognition

- Let M=(S,I,f, s_0, F) be an automaton.
- Let $x \in I^*$ a string of inputs.
- We say that f is recognized (or accepted) bythe machine M if x brings the initial state s_0 into a final state.
- More formally, $x \in I^*$ is recognized by M if
  $f(s_0, x) \in F$ (F is the set of final states)
- Informally, x gives a path in the state diagram that originates at s_0 and ends at a state which is final

What is the language for the following automaton

- Only 1 final state s_3
- Think of all possible paths that originate at s_0 and end at s_3
- We can loop as many times as we want,
  and then we must follow



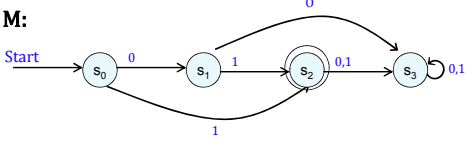
This says the strings recognized by the machine are all of the form $0^n 11$, with $n \geq 0$

Definition

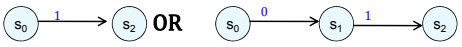
- The language recognized by the machine M, denoted by L(M) is the set of all strings that are recognized by M.
- Two finite state automata are called equivalent if they recognize the same language

Example: Given M determine L(M)

- M:



- There's only 1 final state (s_2). So we look for paths from s_0 to s



- => L(M) ={0,01}

Non Deterministic Finite State Automata

So far we've discussed "deterministic" automata. The sense that for each choice of stat S and input i there was a uniquely determined next stat.

In a "non-deterministic" automaton, f(s,i) is allowed to be any finite subset of states.

Then we can think of the transition of function f as a map

$f: S \times I \rightarrow P(S)$

Where P(S) is the set of all subsets of S.

Notice that f(s,i) can also be the empty set of states.

Non-deterministic Automaton

Formally:

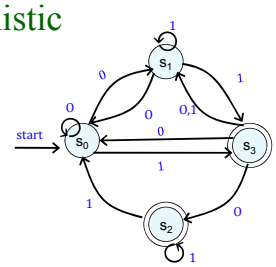
A non-deterministic automaton M=(S,I,f, s_0 , F) consists of:

- A set S of states
- A set I of inputs
- A transition function $f: S \times I \rightarrow P(S)$
- A starting state s_0
- A set F of final states ($F \subseteq S$)

Non-deterministic

- **Final states: s_2, s_3**

States	Inputs 0	1
s_0	s_0, s_2	s_3
s_1	s_0	s_1, s_3
s_2		s_0, s_2
s_3	s_0, s_2, s_2	s_1



Notice that the transition function with state s_2 and input 0 produces the empty set of states: $f(s_2, 0)$ does not give any next state

Languages and Non D FSA

- **If the Language is recognized by a nondeterministic finite state automata then it is also recognized by a deterministic automaton**