

CECS 444 Compiler Constructions

Seminar Notes

August 28, 2018

Syllabus

Things to cover:

- Treewalking (binary)

Textbook:

Fisher, Cytron, Leblanc

- Crafting a Compiler (2009 ~720pg)

Grading:

Cumulative Exams

20% Exams I

20% Exams II

33% Final

20% Projects (Will build on each other)

7% Quiz, Paper, Participation

MGR Types: (Manager Types)

Good: 10% - Super people

Bad: 80% - Need people to do the job

- They buy programmers “By the Yard”

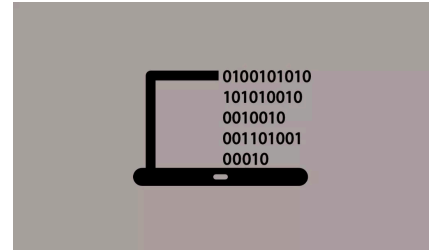
Ugly: 10% - Backstab

Mini- SWE (Software Engineering) Rules

** Reasonable Person STD (Standard)

- Due Diligence (Everybody has their own view)

- Pace yourself



- AIO: (Adapt, Improvise, and Overcome)

** “Smart” Person STD

- Always be ready to show your work (Show your progress)

★ Most Important Things in SW(Software): **MORALE**

Rules:

0. Get to working Software Fast!

(Go ugly early)

Why!



1. You can see it work

* 2. Users can see it & tell you it sucks

- Get users feedback faster

(MVP = Minimum Viable Product)

1. Never Pre-Optimize (Usually 1% of code is too slow)

- Change this 1% and program increases more in speed

*** Optimize ONLY when proven needed

2. No “BUG HUNTS”

I. Compile-Time Errors \leq 5 mins to fix

II. Usually 90% of DEV Time spend on Run-Time Bugs

- How to get rid of it?

• Force all bugs into small box (look there!)

★ Use “Add-A-Trick”

- Add 1-N Lines, Compile, then Test

3. EIO (Expected Input/Output)

*** Build Before Coding (Slice it into Itty-Bitty Stepping Stones)

- It focus design on what is important

*** Avoid “Gold-Plating”



Continued on August 30,
but placed here since it
continue --->

- Making things look nice with nothing to functionality

4. Clean The Page. (~ 50 to n lines of code per page)

- Usually one page for a Function so easy to read

August 30, 2018

Homework: Read Fischer

Chapter 1 Intro - 30pg

Chapter 2 Compiler Parts - 25pg

Chapter 3 Scanner/Lexer - 50pg

Mini Study Rules:

1. Textual Mean

- Build/Use "Flash-Cards" (3x5)

2. Visual Memory

IE: Charts, Graphs, etc

- Draw it twice, looking

- Draw it Blind

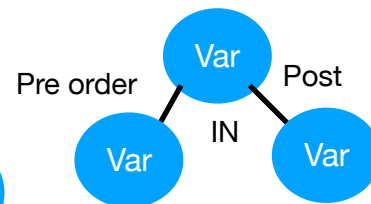
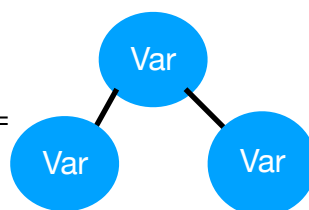
• win 3x

• include labels

TreeWalking:

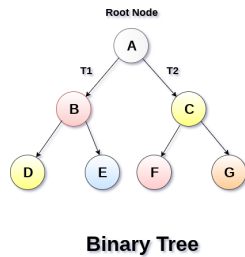
- Consist of:

Left / Right / Lollypop =



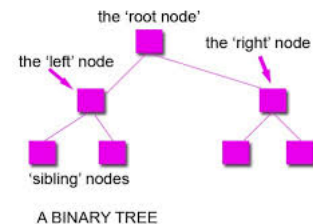
CLASS Node

```
{  
    INT VAL;  
    NODE LKid;  
    NODE RKid;  
}
```



- To Do For TreeWalking:
1. Header
 2. Basic Step
 - Do manually
 3. Left/Right Recur
 4. Deal with Lollypop
 5. Glue

```
Void printTree(NODE root)
{
    # Basic Step
    If (NULL == root)
    {
        RGT; #Abbr. for returning nothing
    }
    # Left Recur
    printTree(root.LKid);
    # Right Recur
    printTree(root.RKid);
    # Deal with LollyPOP
    System.out.println(root.VAL);
    # GLUE
    // None
}
```



(c)www.teach-ict.com

```
Void countTree(NODE RP)
{
    # Basic Step
    If (NULL == root)
    {
        RGT; #Abbr. for returning nothing
    }
    # Left Recur
    INT Lx = countTree(RP.LKid);
    # Right Recur
    INT Rx = countTree(RP.RKid);
    # Deal with LollyPOP
```

```

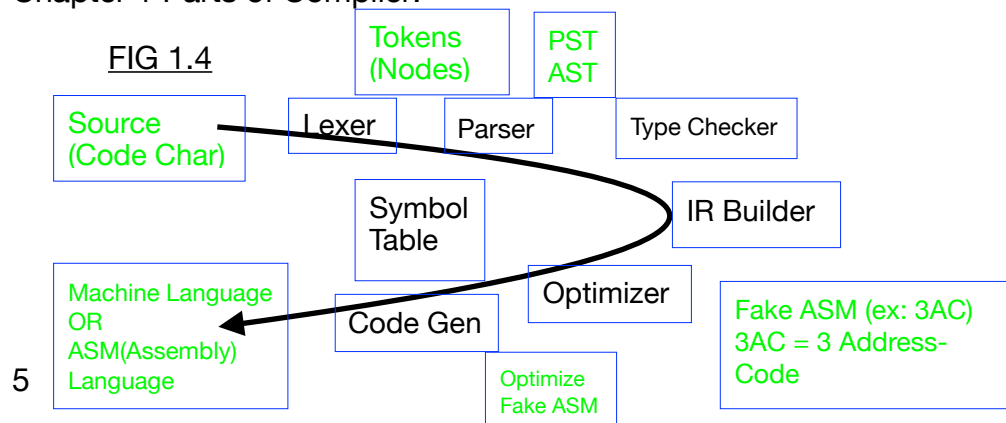
    Px = 1;
    # GLUE
    Return Lx + Rx + Px;
}

Void sumValTree(NODE RP)
{
    ....
    # Deal with LollyPOP
    Px = RP.VAL;
    ...
}

Void sumValForKind(NODE RP, INT RK)
{
    ....
    # Left Recur
    .... RK
    # Right Recur
    ..... RK
    # Deal with LollyPOP
    Px = (RK == RP.kind
        ? RP.VAL
        : 0);
}

```

Chapter 1 Parts of Compiler:



Lexer = Lexical Analysis

- Lang. REGEXES

Parser = Syntactic Analysis

- CFG (Context Free Grammar) Rules

Type Checker & IR Builder = Semantic Analysis (Good meaning)

- IR Builder (Intermediate Representation Builder)

- In each stages, since they are not source or final, they are IR

- AST + Decoration

Optimizer

Code Generation = Final representation (Emitter Phase)

- “Emits” Machine/ASM/Byte Code

- Bytecode usually mean for JAVA since it is old

- For interpreter/VM Architecture

- Machine Architecture Description

Symbol Table:

- Contains all user-define names (names = symbols)
- Are builded into debugger

Front End:

- Between beginning to Syntactics Analysis

Back End:

- After Syntactics Analysis to end

PST (Parse Tree): Convert to AST (through Parser)

AST (Abstract Syntax Tree): In one simple operation from PST —> AST

September 4, 2018

Homework: Read Fisher

Chapter 3 Scanner (Lexer)

Chapter 3.2 REGEX (Regular Expression)

- Regular Lang

1. LITERALS: "3", "Hi"

2. Wildcard Character "operator".

- Uses Period

IE: c.t

All matches of period wildcard \rightarrow {Cat, Cbt, C7t, C\$t, c t, c.t,...}

3. Escape (De-Opify)

- Uses Backslash

IE: c\.t \rightarrow {c.t}

IE: c\\t \rightarrow {c\t}

4. Optional

- Uses Question Mark

IE: Ca?t \rightarrow {ct, cat}

5. Grouping

- Uses Parenthesis

IE: C(a)t \rightarrow {Cat}

IE: (Ca)?t \rightarrow {t, Cat}

6. Zero or More (AKA: Kleene Star)

- Uses Astris

IE: (Ca)*t \rightarrow {t, Cat, CaCat,...}

IE: 123*4 \rightarrow {124, 1234, 12334, ...}

7. 1 or more (AKA: Kleene Plus, Positive Closure)

- Uses Plus

IE: $123+4 \rightarrow \{1234, 12334, \dots\}$

IE: $12.+4 \rightarrow \{12a4, 12b4, \dots, 12ab@724\}$

- Give me one or more “wildcard char op”

8. Any 1 Char: From the set

- Uses Brackets (Anything inside the bracket is auto escape)

IE: $[BFC]at \rightarrow \{Bat, Fat, Cat\}$

IE: $[BFC]?at \rightarrow \{at, Bat, Fat, Cat\}$

*9. Choose Sequence of (AKA “OR”)

- Uses Vertical Stroke

IE: $C(a|o|u)LL \rightarrow \{CaLL, CoLL, CuLL, CooLL, CoooLL\}$

- $a | o+ | u = a \text{ or } o+ \text{ or } u$

10. In a Char Subset: A Range of..

- Uses Hyphen

IE: $a[A-D]z \rightarrow \{aAz, aBz, aCz, aDz\}$

IE: $[_A-Za-z0-9]$

$y := x * 2 + 3$

Project 1 Lexer

Digits = $[0.9]^+$

Leading Underscore = $'_ ' | [A-Za-z]$

FSM = Finite State Machine (AKA: DFA)

DFA = Deterministic(no choice) Finite Automaton

• States

2 Types:

SS = Start State

AS = Accept State(s) AKA Recognized

- Found a Match (Doesn't mean stop)



- Events (Words Event, Letters Event)
- Links/Moves (Labeled with Events)
 - moving from one state to another based on events

Input Event Sequence leading from SS to some AS

- A word/sentence in the “Language” of the Regex

IE: Regex, $R = C(a \mid o \mid u)t$

Lang, $L(R) = \{Cat, Cot, Cut\}$

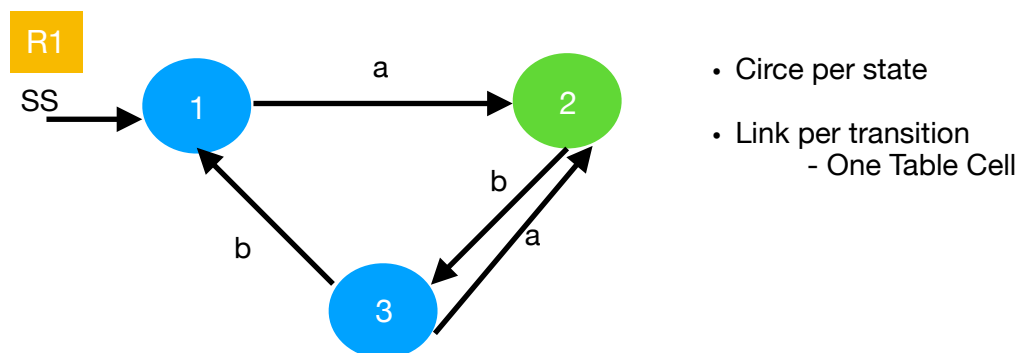
IE: $R' = \text{See the (cat|dog|bear)\?}$

$L(R') = \{\text{“See the cat?”}, \text{“See the dog?”}, \text{“See the bear?”}\}$

DFA Format/“Coding”

State transition

- Table/(Matrix)
- Diagram/(Graph)

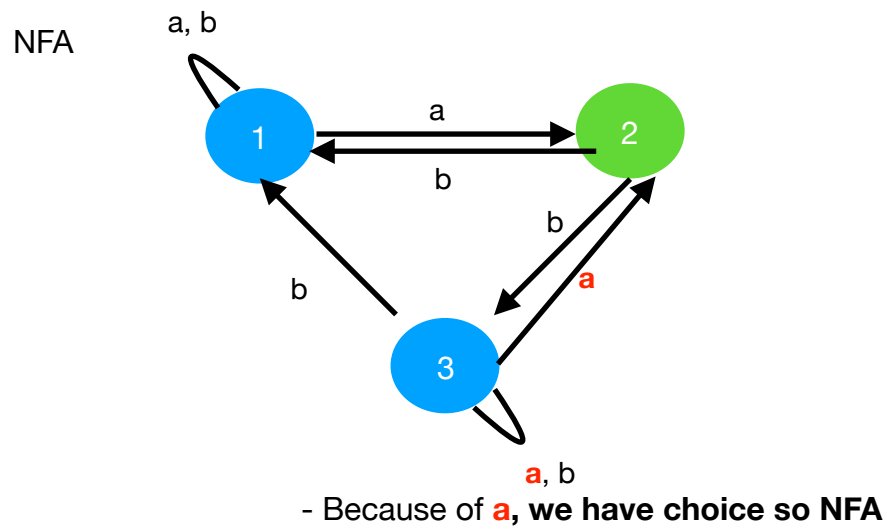


R1	a	b
1	2	-
(2)	-	3
3	2	1

= AS

- Row per state
- Column per event

- Empty Cell = no possible match



R2	a	b
1	1,2	1
(2)	-	1,3
3	2,3	1,3

- NFA Choice, 2 Ways:

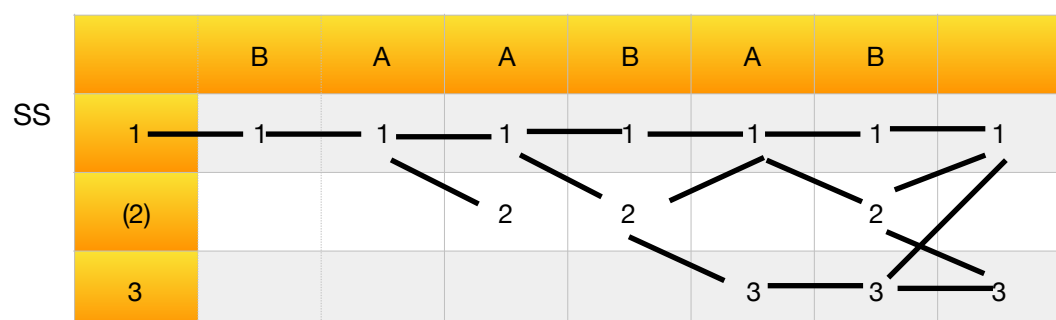
1. Choice of Moves (From State, on Event)

2. “Epsilon Move” - Greek E (ϵ) for empty

*(FISCHER uses Lambda, λ)

★ Convert NFA to DFA:

“Path Graph”: BAABAB\$ (\$ = end of input)

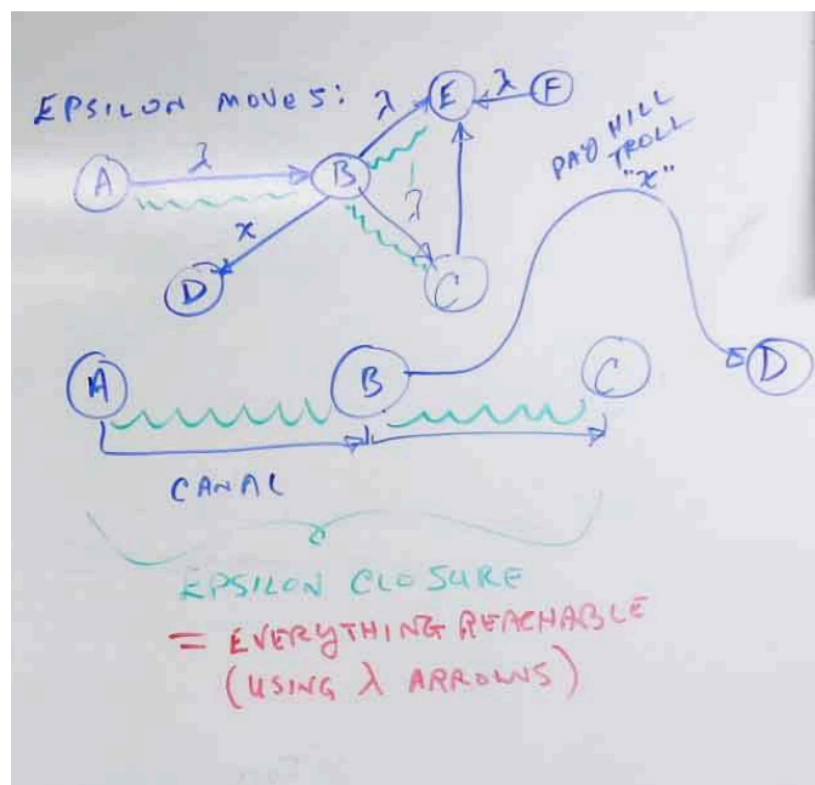


To DFA	a	b
{1}	{1,2}	{1}
{1,2}	{1,2}	{1,3}
{1,3}	{1,2,3}	{1,3}
{1,2,3}	{1,2,3}	{1,3}
{2,3}	{2,3}	

Q: How many DFA States from "N" NFA States max?

Ans: $2^n - 1$

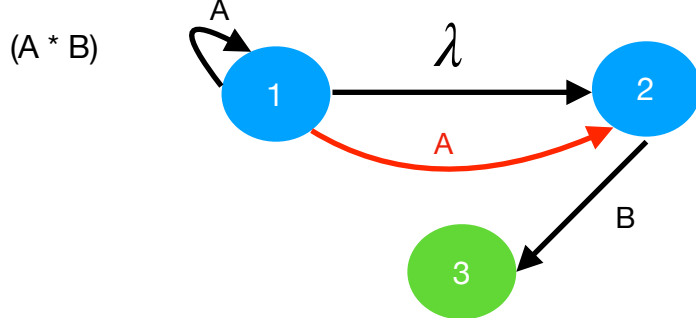
Epsilon Moves:



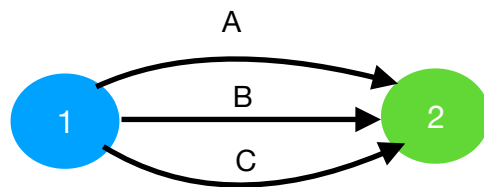
September 6, 2018

Why NFA Bother?

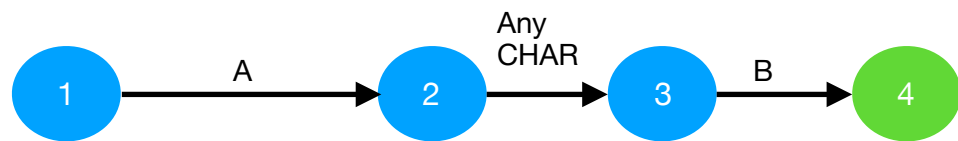
Regex to FSM (Finite State Machine)



(A | B | C)

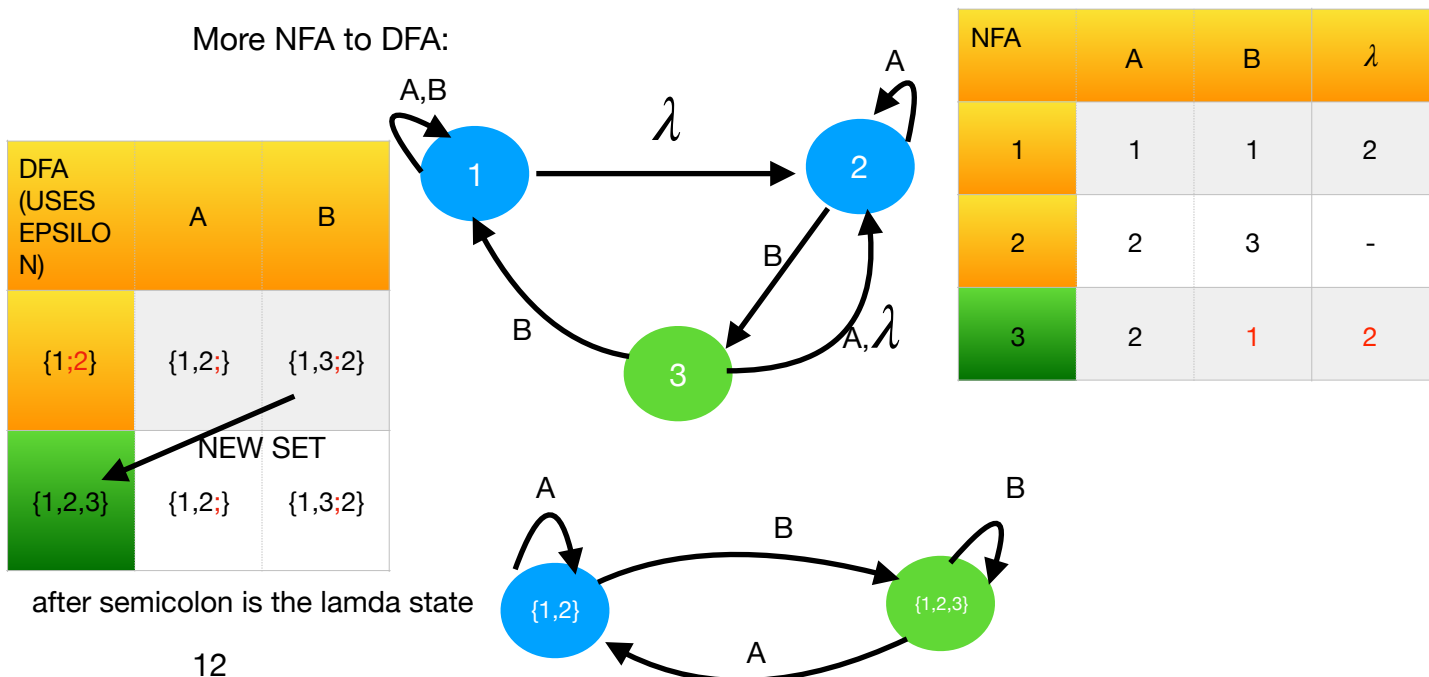


(A . B)



- These Scenario can be combined: (A . B)(A | B | C)

More NFA to DFA:



- The red semi colon is there even though on cell {1;2}A the lamda doesn't go anywhere beside 1 and 2
- On cell {1;2}B , we got a new set, {1,3,2} so we start a new row for {1,2,3}.
 - After that row, no more new set, so we stop
- Since 3 is the accept state, anything with a 3 will be green, aka accepted

READ FISHER

Chapter 4

Chapter 5.1 - 5.4

September 11, 2018

Read Chapter 3:

Skip 3.5 Lex

Read Chapter 4:

then 4.4

Previous: Regular Langs

- REGEX

CFG: Context-Free Grammar

CFG Rules

- LHS = RHS (left hand side = right hand side)

1 Symbol = Sequence 0 or more Symbols

::=

<— = LHS “expands to” RHS in A “Derivation”

GMR “G”:

IE: $S = X \mid Y$ is an example of Combo - Rule

- Rule 1: $S = X$ is Simple - Rule
- OR
- Rule 2: $S = Y$

History: High-Level Langs

1957: FORTRAN (FTN)

1958: LISP (A.I)

1960: Cobol

1960: Algol 60

- Algol 68 (Euro ver.)

Popular Langs: (TIOBE website)

Market Shares:

Java - 16%

C - 14%

C++ - 8%

Python - 5%

C# - 4%

VB - 4%

PHP

Javascript

SQL

RUBY

IE: $X = a \mid yxy$

"a" = lower case "yxy" = symbol sequence

- RHS = Terminal Symbol

1. Can't expand

2. Not on LHS

- LHS = Non-Terminal Symbol

Lang "G": All "sentence" described by GMR "G"

Q: is "bab" in $L(G)$?

Try to derive from Starting Symbol

A: * We can only " \rightarrow " into something that is rule in GMR "G"

S // $S \rightarrow X$

X // $X \rightarrow yxy$

yxy // **second y** $\rightarrow b$

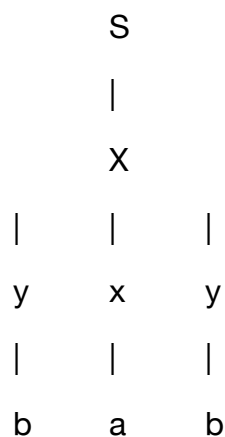
- Right-Most Derivation

yxb // $X \rightarrow a$

yab // $Y \rightarrow b$

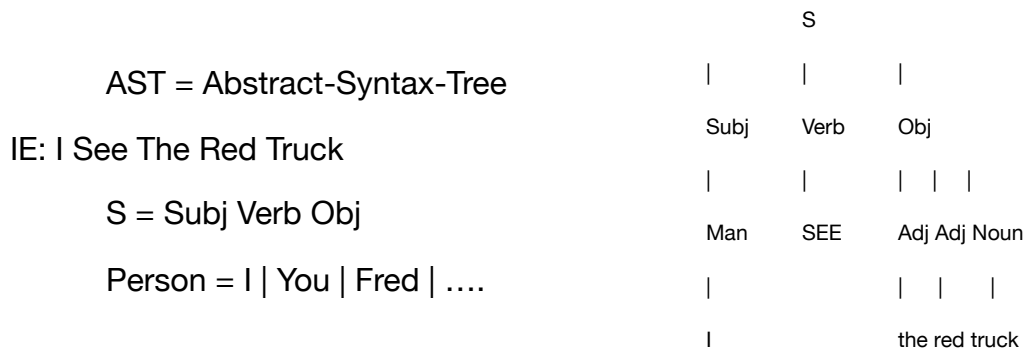
bab // WIN

Parse Tree (PST):



LR = Left-To-Right Scan & Right-Most Derivation

LL = Left-To-Right Scan & Left-Most Derivation



“GE”: Arith Expr GMR

$E = E '+' T \mid E '-' T \mid T$

$T = T '*' F \mid T '\div' F \mid F$

$F = \text{id} \mid k \mid '(' E ')'$

IE: $B * B - 4 * A * C$

PIC GOES HERE

“Recursive Descent” Parser

- Uses Depth First Search (DFS)

REG:

- Each Non-T gets a Function
- Each Rule gets a “Trial” in that function
 - to match next input Sequence

BOOL E () // Match the first E = ... rule

{

Input_Pos = Current;

if (match (E) && match('+') && match('T'))

```

{
    return True;
}
Else ( E() && match('-') && T())
{
    return True;
}
Else ( T() ) return True;
return False;

```

Cons:

- Most Tries Fail
 - Very Slow
- If Error, tries everything first

LL Parse Machine

Machine + Table + Stack

4 Steps Predict the rules Partial Derivation

September 13, 2018

LL Parser:

Machine, Table, Stack

Table: (LHS, TERM) — LHS = top of stack TERM = Front of input

	Terminal 1	Terminal 2	...
LHS 1	###	-	
LHS 2	-	-	R#2
LHS 3	-	R#16	
...			

R#16 = Simple Rule (Not Combo)

$N = N A B C \rightarrow$ Left Recursion

$X = A B X C \rightarrow$ Also recursion

Things we need to do:

1. Need simple rules
2. Get rid of left recursion

Combo \rightarrow Simple:

$$E = E + T \mid E - T$$

- Breaking it down : $E = E + T$

$$E = E - T$$

$$A = B [C-DF] G$$

- Breaking it down: $A = B C G$

$$A = B D G$$

$$A = B F G$$

- The F is there as extra step

$$A = B ? C$$

- Breaking it Down: $A = B C$

$$A = C$$

$$A = X * Y$$

- Breaking it down: $A = N Y$

$$N = X N \quad - \text{Right Recursion}$$

- As much X as I want with N since *

$$N = \lambda$$

$$A = B (X \mid Y) ? C$$

- Breaking it Down: $A = B X C$

$$A = B Y C$$

$$A = B C$$

$$A = B X + Y \text{ or } A = B X X^* Y \text{ (C.F Ex: p 138 \#3, 7)}$$

- $A = B X X^* Y$ is similar to $A = X^* Y$ so just pretend $B X$ is the front

LRE = Left Recursion Elimination

Direct LRE = $N = N$ - Directly calling itself, NOT foo calls bar and bar calls foo.

GE: * In GE, lowercase are TERM and uppercase are Non-TERM

$$E = E + T \mid E - T \mid T$$

$$T = T * F \mid T / F \mid F$$

$$F = i \mid K \mid (' E ')$$

- 9 Rules

3 Rules to do LRE for one NON-TERMINAL, N:

1. Add a new NON-TERMINAL (EX: "X")

- Which means X goes to nothing AKA $X = \lambda$

2. Append X to all N right hand sides.

3. Replace "Lefty Part" ($N = N$) with ($X =$)

- so $E = E \rightarrow X =$

GE:

$$E = E + T \mid E - T \mid T$$

$$T = T * F \mid T / F \mid F$$

$$F = i \mid K \mid (' E ')$$

$$Q = \lambda$$

$$E = E + T Q$$

$$E = E - T Q$$

$$E = T Q$$

$$Q = + T Q$$

$$Q = - T Q$$

$$R = \lambda$$

$$T = T^* F R$$

$$T = T / F R$$

$$T = F R$$

$$R =^* F R$$

$$R = / F R$$

GE2:

$$E = T Q$$

$$Q = + T Q \mid - T Q \mid \lambda$$

$$T = F R$$

$$R = * F R \mid / F R \mid \lambda$$

$$F = i \mid K \mid ' (E ')$$

$$L(\text{GE2}) == L(\text{GE})$$

Next Chapter: 4.5.2

September 18, 2018

Disappearing Non-T's

Direct Epsilon Rule: $N = \lambda$ \leftarrow 0th step

Indirect Epsilon Rule: $N = AB \mid fg$

$A = \lambda$ \leftarrow 0th step

$B = C \mid h$

$C = \lambda$ \leftarrow 0th step

Consult Further (CF) Fig 4.7, p 129

ALT: KEY: Bottom Up

ALGO - Direct Epsilon, First

1 Step Indirect, Next

2 Step ,

ETC

Disappearing Set of Non-Ts:

“Dirty Bit”

Simple Rules:

(N, -) \leftarrow go to DS

(N, AB)

(N, fg) \leftarrow RHS, only Terms

(A, -) \leftarrow go to DS

(B, C)

(B, h) \leftarrow Take out

(C, -) \leftarrow go to DS

Step:

0 : (N, A, C) \rightarrow Disappearing Set

- Look at right hand side for N, A, C and eliminate

- Red = take out based on NAC

- for (B, C), right hand side became empty so add B

1: (N, A, C, B) \rightarrow DS

Point:

Add direct Epsilon Rule

For each indirect

Disappearing Non-T:

Simplify Table Build

Setup:

DS \leftarrow Empty

SR \leftarrow All Simple Rules

Dirty = True;

while (Dirty)

{ // One Step

 Dirty = False

 For each Pair in SR

 RHS = Pair.RHS

 For each N, A, C in DS

 RHS -= N, A, C

 IF RHS is Empty

 {

 DS += Pair.LHS

 Dirty = True

 }

}

First Sets:

- Let RHS

- Look at L(RHS) RHS = “Reduced Grammar”

• Set of all Terms that start a sentence in L(RHS)

$F(\lambda) = \{ \}$

$F(h) = \{ h \}$

$F(fg) = \{ f \}$

$F(C) = \{ \}$

Union $F(C=RHS)$

$F(AB) = A = F(\lambda) = \{ \}$

September 20, 2018

No Class

September 25, 2018

Exam Review:

NOX (not in exam): MINI - SWE Rules, PST

In exam:

- Treewalking
 - not just binary tree
- Spiderweb Diagram

EX:

- What is it?
- What are the parts?
- What are the links?
- Leads to glossary terms

- Front-End
- Back-End

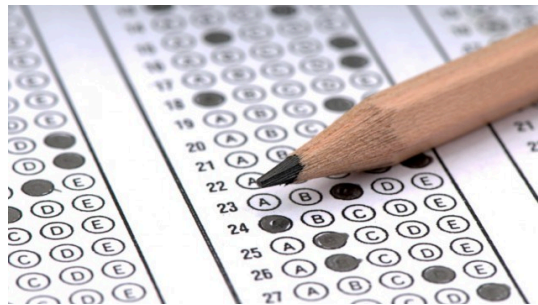
What Parts?

Glossary Items:

- Bytecode
- (PST) = parse tree
 - non standard, but mirrors AST
- AST = abstract syntax tree

AST vs PST

- PST = every node and its kids are



- Non terminal = shows on left and right side
- Terminal doesn't expand so it on only right
- AST = contains only Terminal symbol

Why AST?

- Twice as many nodes
- Messing with tree, mom and kids in PST will change
- LEXER
- Scanner
- Tokens
- ASM
 - Assembly language
- CFG
 - Context Free Grammar
- IR
 - Intermediate Representation
 - Source code is NOT IR
 - PST and AST are IR
 - Code coming out are NOT IR
- 3AC
 - 3 Address Code
- Semantic Analysis
 - Checking on meaning
 - Making well form sentences with reasonable meaning

EX: The Bread ate the cat : Bad

The Mouse ate the bread: good
- Syntax Analysis

- Check on Sentence Form by Grammar (part of speech)
- Glue words together to form proper sentence
- Well form coded

- Ex: a statements, assignment, etc

The Bread ate the cat : works here

- Since Semantic will check meaning

- **Lexical Analysis**

- Check if word exist in Dictionary

IE: Is word in “Lexicon”

- Glueing letters together to form word
- Deals with stuff from below the level of words

- EX: words are “+=“ which are tokens

- REGEX

- What is t?
- What are the Operations?
- What are their actions?
- What are legal tokens/words in L(R)
 - L(R) = Language of Regex

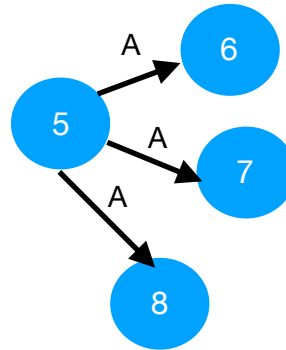
- DFA

- What is it?
- What Parts?
- SS,(AS) = Start state, Accepted State
- L(G) = language of grammar
- State transition
 - TABLE (Matrix)
 - Diagram (Graph)

- Convert from Diagram to Table?

- NFA

- What is it?
- How to tell if NFA vs DFA?
 - Choice move?



- Epsilon move λ
- Extra Column in table for λ

- Convert NFA to DFA

- Know how
 - Path graph + fill in
 - All subset of NFA states
 - Start with DFA SS. and use Reachability (row by row)
- Epsilon Closure
 - CANALS
 - WETNESS
 - HILL TROLLS

Glossary Items:

- | | |
|---------------|------------------------------------|
| • REGEX | • Recognized |
| • DFA | • Accepted |
| • FSM | - Recognized vs Accepted =
SAME |
| • NFA | |
| • EPS Closure | |

- SS

- Why bother with NFA?

- REGEX operations give NFA

EX: $A (AB \mid AC) D$

- History

- 1957 Fortran - Math Equation
- 1958 LISP - Lambda Calc
- 1960 ALGOL - Fortran-killer
- 1960 COBOL - Don't need Programmers

- Arithmetic Expression Grammar

- How to use it to build
 - A PST from an input Expression

Recalls:

PST mom + kids

GMR: LHS + RHS

Glossary:

- LHS • RHS

- L_1 = Left-To-Right Scan of Input Tokens

- L_2 = Left-Most Derivation

- R_2 = Right-Most Derivation

EX:

a b X c F a Y Z p Q a - Cap = Non Terminal

- LL expands the Capitalized Letter from left to right
- RR expands the Capitalized Letter from right to left

- Recursive Descent Parser

- What is it?
- How to build (given GMR)
 - It is LL (Left Recursion)
- Mutual Function Call Recursion
 - A calls B and B calls A (Has to be a loop)
- PROS
 - Rewind the input
 - Real simple to build the functions
- CONS
 - Checking all possible Derivations
 - Could take a long time because it tries and fail to check
 - Very SLOW

- LL Parse Machine

- What is it?
- What parts?
 - Machines
 - Tables
 - What is it?
 - Parts?
 - Row Headers
 - Has LHS Symbol
 - Column Headers
 - Has Event/Terminal Symbol
 - Cells
 - One Simple Rules in a Cell

- LHS is Row headers
- How does it index?
 - (LHS,Token) = (Row, Column)
- Runtime Stack
 - Any LL parser can't handle Left Recursion
- Left Recursion in a Simple Rule
 - What is it?
 - $A = A ..$
 - Convert Combo Rules to Simple Rules

Glossary:

- LRE (Left-Recursion Elimination)
 - Removes Left Recursion in Grammar
- Disappearing Non-Terminals
 - Direct (easy)
 - Indirect
 - How to find them all

Glossary:

- Terminal Symbol
- Non-Terminal Symbol

What is a Epsilon Rule?

- Rule where Terminal goes to nothing

September 27, 2018

MIDTERM

October 2, 2018

GE2 : (LRE'd)

$$E = \overset{1}{T}Q$$

$$Q = \overset{2}{+}TQ \mid \overset{3}{-}TQ \mid \overset{4}{\text{esp}}$$

$$T = \overset{5}{F}R \leftarrow \text{IF "F" has an esp, then there is a ghost rule for T goes to R}$$

$$R = \overset{6}{*}FR \mid \overset{7}{\div}FR \mid \overset{8}{\text{esp}}$$

$$F = \overset{9}{i} \mid \overset{10}{k} \mid \overset{11}{(' E ')}$$

* Build First Step, Each simple rule (RHS)

LL Table

only terminal →	i	k	+	-	*	÷	()	\$ (EOF)	T (F,i)
E										E
Q										.
T										.
R										.
F	R#9 or F=i									.

-----> F(R#9) = {i}

L(F=i & GE2) = {"i"}

F(f=i) = {i}

F = First Set

f=i = GE2 Non-T

F(#9) = {i}

F(#10) = {k}

F(#11) = {'('} ← every thing has to start with (for rule 11

F(#6) = {*}

$$F(\#7) = \{\div\}$$

$$F(\#2) = \{+\}$$

$$F(\#3) = \{-\}$$

$$F(\#4) = \{ \} \leftarrow \text{eps goes to nothing (eps} = \lambda)$$

$$F(\#8) = \{ \} \leftarrow \lambda$$

$$F(\#5) = \{F(f)\} = \{i, k, '('\}$$

$$- F(f) = F(\#9) + F(\#10) + F(\#11) = \{i, k, '('\}$$

$$F(\#1) = \{F(T)\} = \{i, k, '('\}$$

$$- F(T) = \{F(\#5)\} = \{i, k, '('\}$$

New Table: to get rule, look at equation and see what rule applies

- EX: $Q = \text{rule } 2 \mid 3 \mid 4 \rightarrow$ we put 2 and 3 under (Q, +) & (Q, -)

only terminal →	i	k	+	-	*	÷	()	\$ (EOF)
E	1	1					1		
Q			2	3					
T	5	5					5		
R					6	7			
F	9						11		

First Set Algorithm:

1. If RHS starts with sym X, Add $F(x)$ to $F(\text{RHS})$
2. If X has an esp rule $A = XYZ$, then Add $F(A=YZ) \rightarrow$ Ghost Rule
And keep going with Y , etc
3. If like $A = \dots Y$ & Y has eps rule
 \rightarrow Ignore it (Handle in Follow Set)

Follow Sets: $W(.)$

- . = Non Term

W-1. $W(S.S) += \$(\text{EOF})$

W-2. $W(N)$:

Find N in each RHS of Rule " $Z = \dots$ "

A. If $Z = \dots NXYZ \dots$

ADD $F(X)$ to $W(N)$

B. IF $X = \text{eps}$ is a Rule

Because "Ghost Rule"
with X gone.

ADD $F(Y)$ to $W(N)$, etc. $Z = \dots N [] YZ \dots$

C. IF Ghost Rule $Z = \dots N \leftarrow \text{RHS} - \text{Last Symbol}$

ADD $W(Z)$ to $W(N)$

Continuing from Set of Rules

$W(E) = \{\$, '\}'$

$W(Q) = \{W(E), W(Q), W(Q)\} \leftarrow \text{Recursion } W(Q)$

$W(E) = \text{Rule 1}$

$W(Q) = \{\$, '\}'$

$W(Q) = \text{Rule 2}$

$W(T) = \{F(Q), W(E), F(Q), W(Q), F(Q), W(Q)\}$

$F(Q) \ \& \ W(E) = R\#1$

$F(Q) \ \& \ W(Q) = R\#2$

$F(Q) \ \& \ W(Q) = R\#3$

$= \{+, -, \$, '\}'$

$W(R) = \{W(T), W(R), W(R)\}$

$W(T) = R\#5$

$W(R) = \{+, -, \$, '\}'$

$W(F) = \text{BUT WAIT!!!}$

Done for 2 reasons:

1. No $F = \lambda$ Rule

2. No Further worry about other λ Rules

LL TABLE:

only terminal →	i	k	+	-	*	÷	()	\$ (EOF)
E	1	1					1		
Q			2	3				4	4
T	5	5					5		
R			8	8	6	7		8	8
F	9	10					11		

- NO two rules in a single cell

• It means Grammar Bug (Or your bug)

October 4, 2018

LL Table

Cell Conflict:

$S = i \text{ eq } (\text{inp} \mid E)$ $S = {}^1 S1 \ \& \ S = {}^2 S2$

$S1 = {}^3 i \text{ eq } \text{inp}$ $F(S1) = \{ i \}$

$S2 = {}^4 i \text{ eq } E$ $F(S2) = \{ i \}$

$L(S1) = \{ "i" \}$

$L(S2) = \{ "i \text{ eq } \dots",$
 $"i \text{ eq } \dots", \dots \}$

LL Table	i	
...			
S		1,2	
S1		3	
S2		4	

Left Factoring: (Direct)

$S = {}^1 i \text{ eq } T$ $F(S) = \{ i \}$

$T = {}^3 \text{inp} \mid {}^4 E$ $F(\#4) = F(E) = \{ i, k, '(\}$

LL Table	i	..	iNP	K	('
...						
S		1				
T		4		3	4	4

$F(E) = \{i, k, '('\}$ from previous set E

- inp = INPUT

- E = expression

- eq = equals

- SO $S = i \text{ eq } (\text{inp} \mid E)$

- ³ = Input Statement

- ⁴ = Assignment Statement

LL MECH:

- Setup: Push \$, Symbols

- // Stack Element is a Symbol

While STACK

- Let TOP, FRONT.....

- M1: IF TOP == FRONT

- IF \$ == TOP, Success (We're done)

- ELSE, POP & Advance the STACK

- M2E: (Error)

- IF TOP is a TERM, ERROR (We're done)

- // TOP is a non-TERM

- Let Cell == LLT[TOP_{ID}, FRONT_{ID}] LLT = LL Table

- M3E: IF Cell is Empty, ERROR

- M4: POP //No Advance

- PUSH (Reversed(RHS))

- MP: Print TOP, FRONT, STACK

In Lab:

Stack:
: \$ E
: \$ Q T

$X + Y \$$

$i_x + i_y \$$

E = ¹TQ

Q = ²+TQ | ³-TQ | ⁴ λ

T = ⁵FR

R = ⁶*FR | ⁷÷FR | ⁸ λ

F = ⁹i | ¹⁰k | ¹¹(' E ')

I GAVE UP!!!

LL Table	i	k	+	-	×	÷	()	\$
E	1	1					1		
Q			2	3				4	4
T	5	5					5		
R			8	8	6	7		8	8
F	9	10					11		

October 9, 2018

October 11, 2018