Compiler Jeopardy

Your Host:

Alex Trebek...

A Few Simple Rules

1. Three Rounds: Jeopardy!, Double Jeopardy, Final Jeopardy
2. I choose the question – category and point value.
3. Your team has 1 minute to write an answer.
4. Give answer to your judge, who decides and records score.
5. Highest score wins.
6. Wager any number of points in Final Jeopardy round.

Got Team?
Got Paper?
Here we go…

Click to Begin

Click to Double Jeopardy

General Compilers++	Scanning	Grammars	Parsing	Symbol Tables
100 Point	100 Point	100 Point	100 Point	100 Point
200 Points	200 Points	200 Points	200 Points	200 Points
300 Points	300 Points	300 Points	300 Points	300 Points

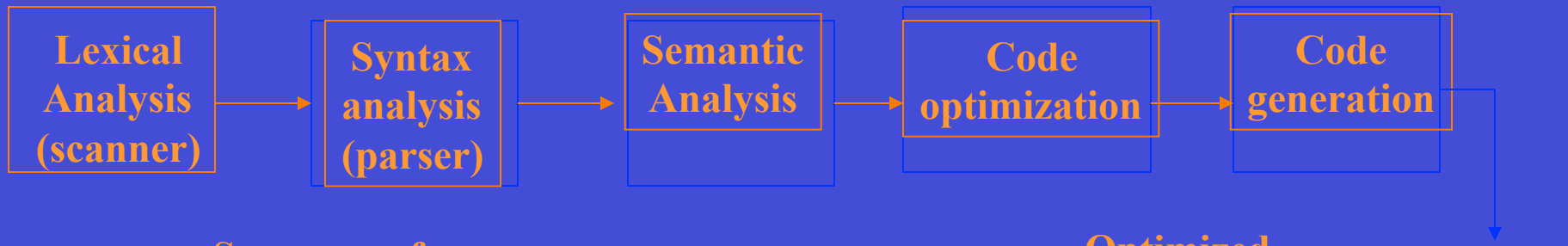
C100: List 3 advantages of a compiler over an interpreter

C100 Answer:

Execution of code faster
Exploit archiecture
Compilation only once – execute compiled code
Ability to optimize the code

C200: Draw the major compiler phases and their inputs & outputs

C200 Answer:



Front End
machine independent
language dependent

Middle

Back End
machine dependent
language independent

C300: Give 1 example of an error typically caught by the scanner, 1 caught during parser, and 1 error caught during semantic analysis.

C300 Answer:

Misformed token – ab#q
Statement misformed – x :== y
Variable not declared

S100: Show an example that clearly demonstrates the difference between lexeme and token.

S100 Answer:

Token – ID
Lexeme - max

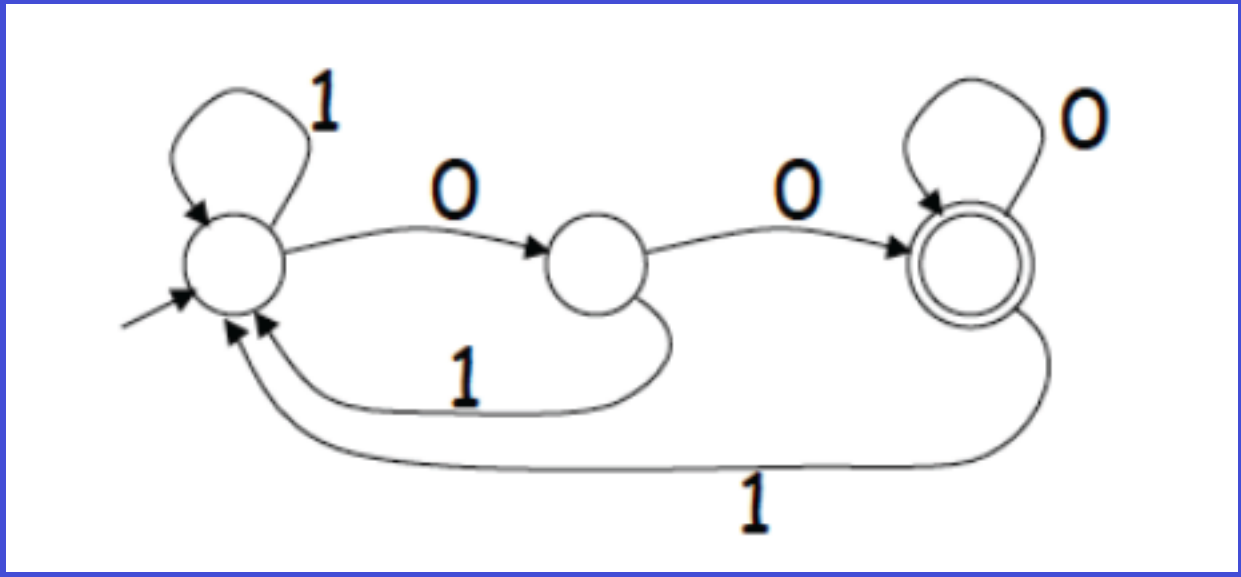
S200: Which of the following strings is accepted by the regular expression bba*b*(ab*a*b)*

1. bbab
2. bbaabab
3. bbaabbabba

S200

1. bbab
2. bbaabab

S300: Write a regular expression representing the strings accepted by this DFA.



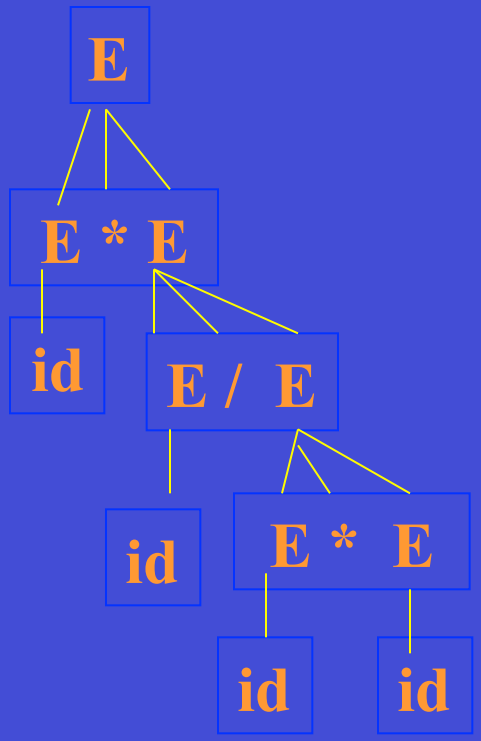
S300 Answer:

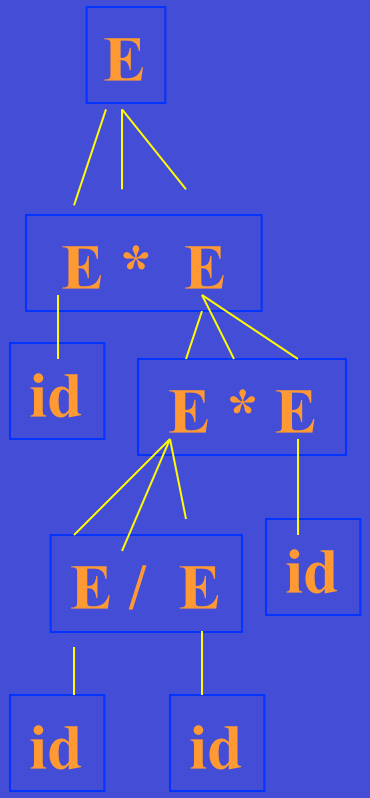
(0|1)* 00

G100: Show that this grammar is ambiguous.

E -> E * E | E / E | id

G100 Answer:





G200: Eliminate left recursion from this grammar using the general rule:

D -> D , id | id

G200 Answer:

D -> id A’
A’ -> , id A’ | ε

G300: Given the following grammar, what are the precedence and associativity rules for the operators?

E -> E & T | T % E | T

T -> T # F | T @ F | F

F -> g

G300 Answer:

Precedence:
#/ @
& %

Associativity

& is left
% is right
 # is left
@ is left

P100: What additional information is needed to determine if this state is adequate in an SLR(1) grammar?

E -> T.

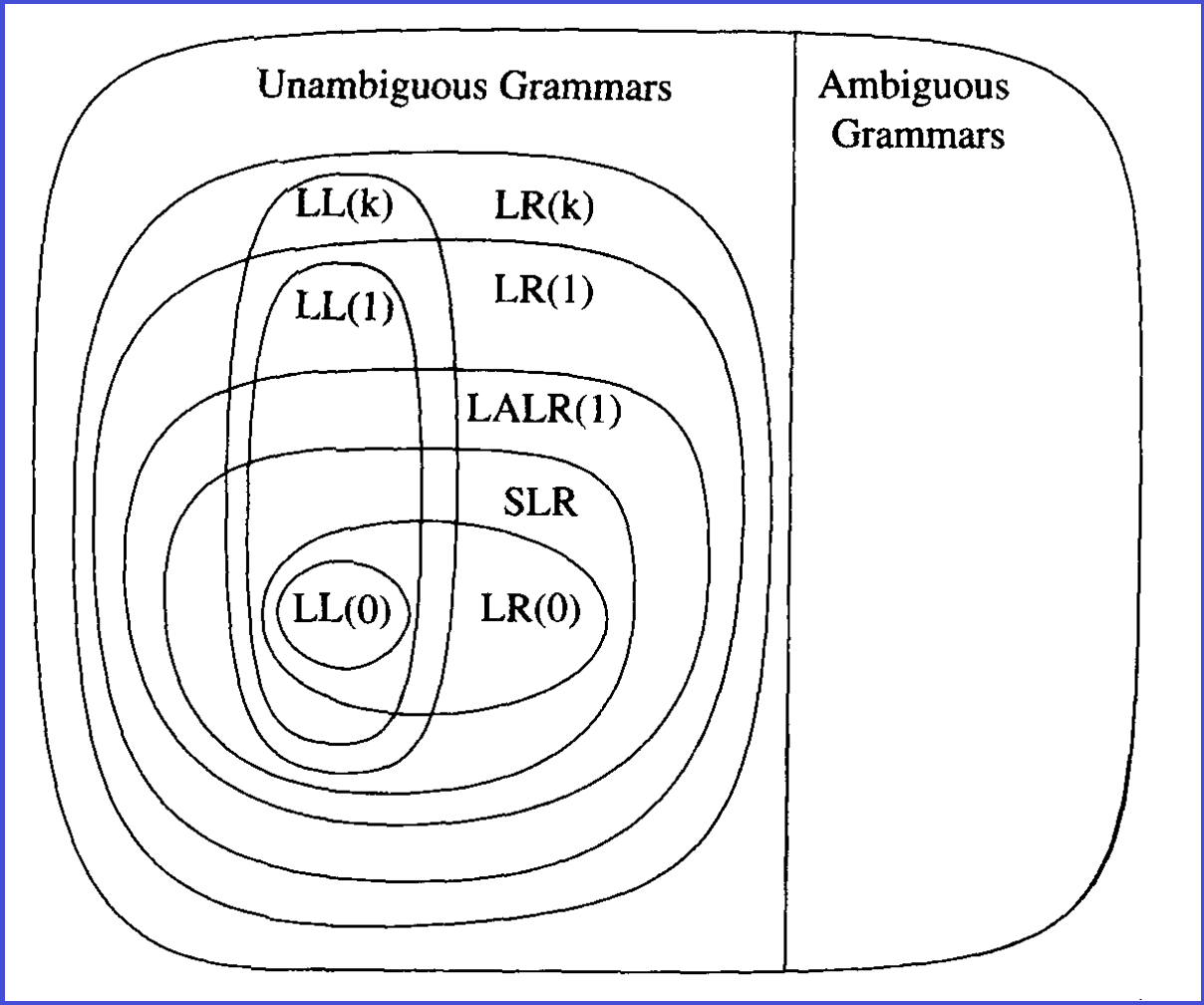
T -> T.+f

P100 Answer:

Follow set of E

P200: Show the grammar hierarchy including LR family, LL(1), ambiguous and context-free grammars

P200


Unambiguous Grammars
Ambiguous Grammars
LL(k)
LR(k)
LL(1)
LR(1)
LALR(1)
SLR
LL(0)
LR(0)

P300: Given the LR(1) state

S8: [E ->a., {;,+}]

[E-> E.*T, {;}]

Give the table entries for state 8 in the LR(1) parse table.

P300 Answer:

+	;	*
R1 {E -> A}	R {E -> A}	S {E -> E *T}

ST100: List the major operations performed on a symbol table.

ST100 Answer:

enter_scope()	start a new nested scope
lookup(x)	finds current x (or null) via scoping rules
insert_symbol(x)	add a symbol x to the table
local-lookup(x)	determines if x in local scope
exit_scope()	exit current scope

ST200: List 2 semantic checks that can be done at compile time and 2 that need to wait until runtime.

ST200 Answer:

Compile time:
Variable declared before use
Number of parameters matches number of arguments

Run time:
Subscripts in range
Size of array must be positive

ST300: Describe how each ST operation uses the active ST stack

ST300 Answer:

Enter scope: Produces new symbol table for declarations
Loopup (x) Searches stack from top for first use of variable
Insert_symbol – inserts variable in current symbol table – top of stack
Local-lookup – determines if x is in current block at top of stack
Exit_scope – removes current symbol table from stack

Click to
Final Jeopardy

Topdown	General Parsing	Bottomup	Surprise
200 Point	200 Point	200 Point	200 Point
400 Points	400 Points	400 Points	400 Points
600 Points	600 Points	600 Points	600 Points

GP200: Name at least 2 differences between top down and bottom up parsing.

GP200 Answer:

Top down goal directed – from top matches string
Bottom up – reduces string

Top down cannot handle left recrusive grammars
Bottom up can handle left recursive grammars

GP400: Left factoring:

Which kind of parsing is it done for?

What problem does it solve?

Show an example.

GP400 Answer:

Predictive top down parsing
It avoids the parser having to backtrack after wrong decisions
Due to common prefixes.

Example: common prefix is ‘a’

X -> a B | a C
C -> B d e | d
B -> x

GP600: Conflicts in Bison:

What happens if you don't resolve them?

How can you resolve them?

GP600 Answer:

Get shift reduce errors or reduce reduce errors
Can resolve them using special symbols to indicate Associativity and Precedence of operators

Or can change grammar

T200: Compute the FIRST for each right hand side of the grammar:

E -> E + T | T

T -> T & S | S | a

S -> S / F | (F) | b

F -> c

T200 Answer:

First E = {a, b, (, c}
etc

T400: Compute the Follow for each nonterminal in:

E -> TB

B -> + TB | ε

T -> FC

C -> *FC | ε

F -> (E) | a

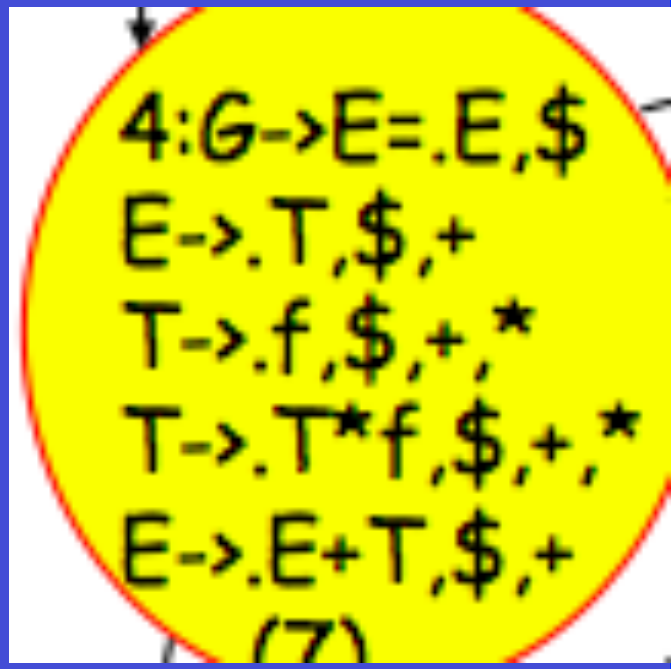
T400 Answer:

T600: Write the conditions for a grammar G to be LL(1)

T600 Answer:

A grammar G is LL(1) iff
whenever there exists A -> α | β in G,
all of the following conditions hold true:
* FIRST(α) ∩FIRST(β) = 0
* At most 1 of α and β derive the empty string
* If β derives the empty string, then FIRST(α) ∩ FOLLOW

B200: Show the LR(1) table entries for the state:



B200 Answer:

B400: For the grammar below, show the LR(1) and the LR(0) closures for the initial state S'-> .G

S' -> G
G -> E = E | f
E -> T | E + T
T -> f | T * f

B400 Answer:

B600: What are the two ways to create an LALR(1) parser without using a parser generator?

B600 Answer:

Construct LR(1) sets of items
And merge core states

Or
Construct LR(0) sets of items and lookahead
Sets as you create the LR(0) items

SU200: What is the full name of the chair of our CS department?

SU200 Answer:

SU400: When there is no local declaration of a name, where do we go next to look for it in STATIC scoping? DYNAMIC scoping?

SU400 Answer:

Static scoping:
Search Symbol tables for static scoping based on scoping from text of program – most closely nested rule ; Next outer block from current block (not always global)

Dynamic Scoping:
Search symbol tables based on run time calls; the caller's Stack frame

SU600: List 5 entities that are named in the Decaf language

SU600 Answer:

Program
Variable
Formals
Interface
Class

Final Jeopardy

Make your wager

Final Jeopardy

Here we go…

Click here for Final Jeopardy

What has been the value of compilers in computing technology?

Ability to write code in high level languages