CS143: Parsing VI

David L. Dill

Stanford University

Parsing

- Connections with Formal Language Theory
- Semantic Actions
- Error recovery

Connections With Formal Language Theory

Properties of Grammars vs Properties of Languages

"G is LL(1)" – no conflicts in LL(1) parse table.

"$\mathcal{L}$ is LL(1)" – there exists an LL(1) grammar for $\mathcal{L}$.

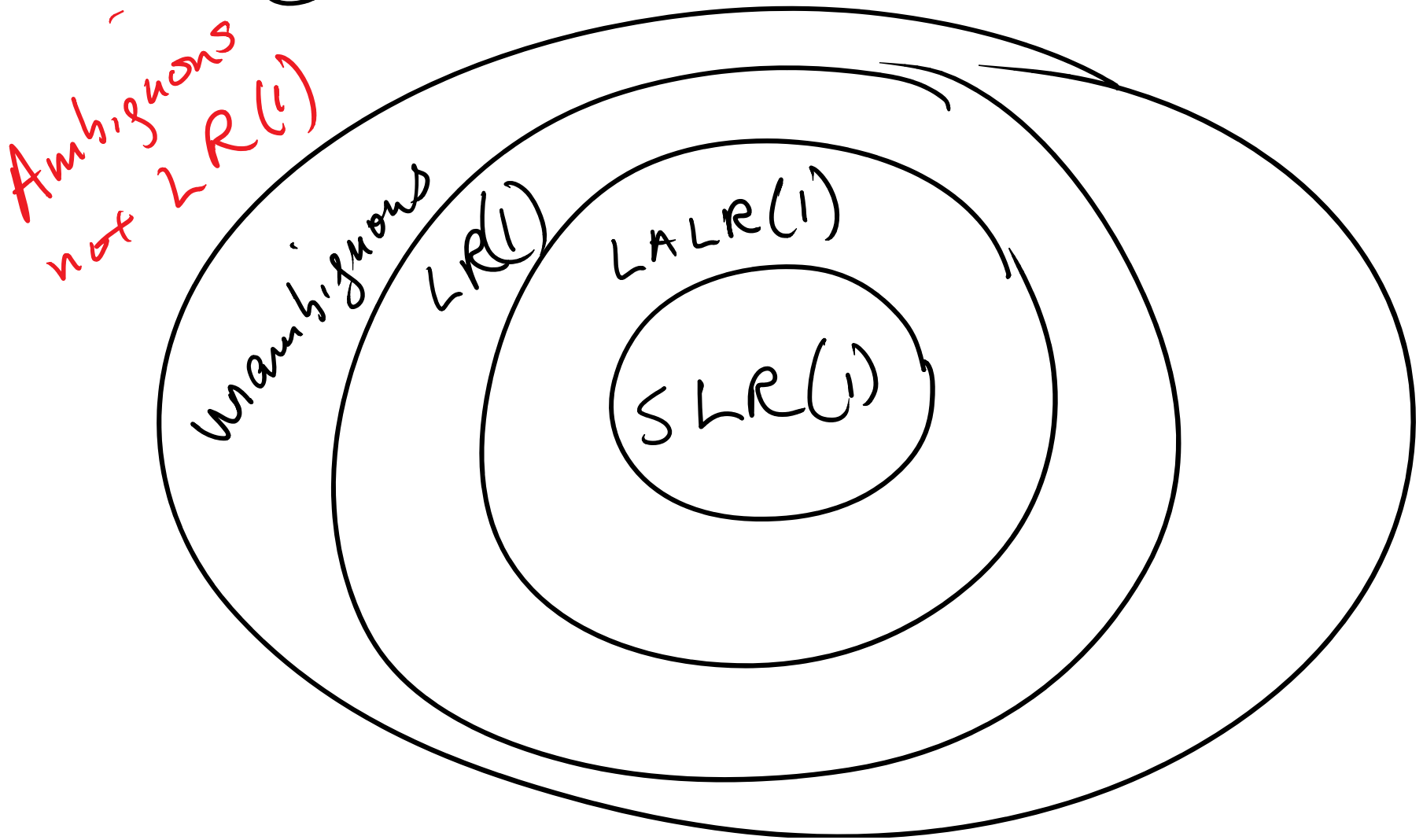
$S \to S$

$S \to a$

Is grammar LL(1)?

Is language LL(1)?

Classes of Context-Free Grammars



$S' \to S$
$S \to Aa$
$S \to Bb$
$S \to ac$
$A \to a$
$B \to a$

SLR(1)

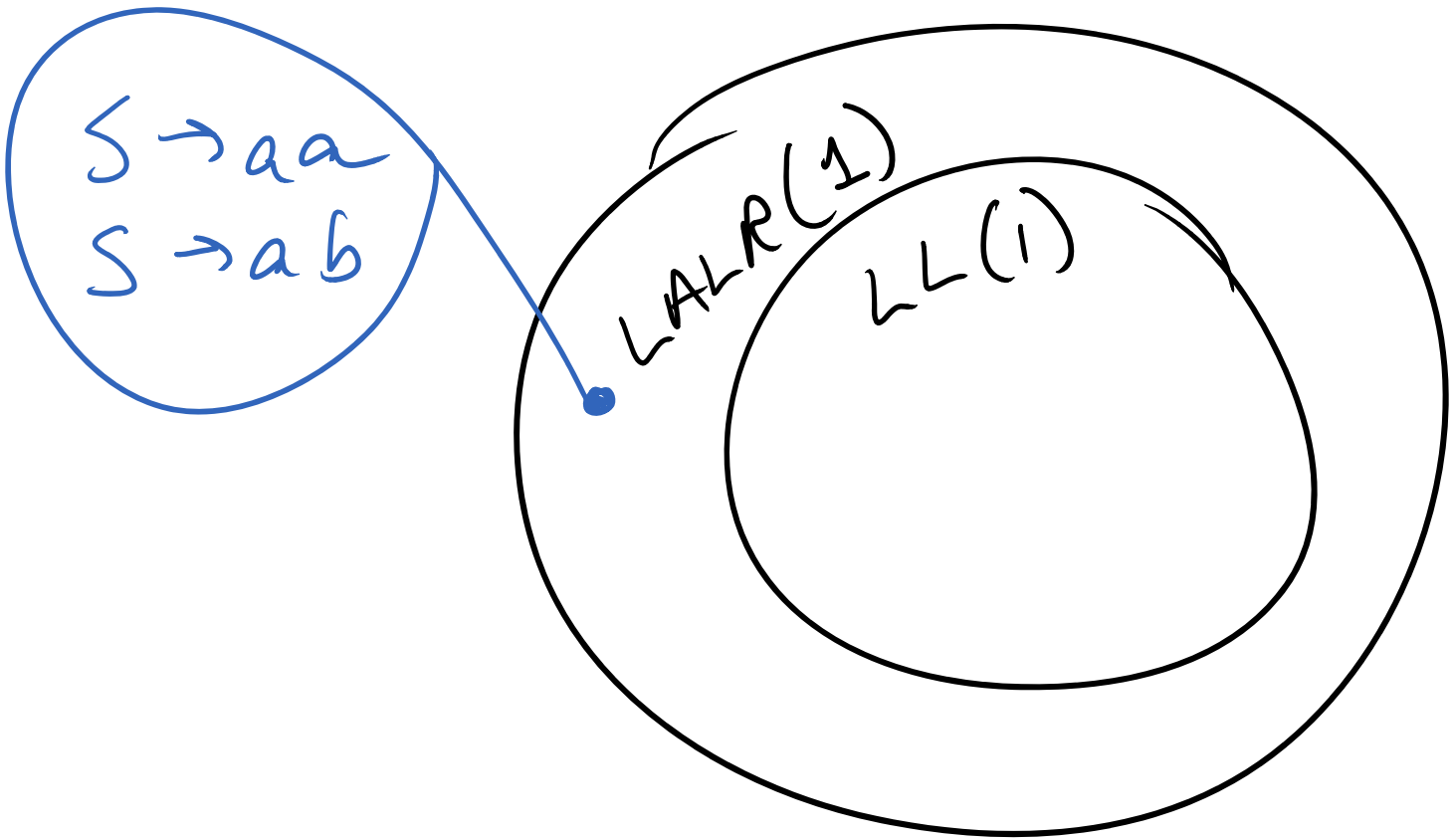
$S' \to S$
$S \to Aa$
$S \to Bb$
$S \to bAb$
$A \to a$
$B \to a$

LALR(1)
(not SLR(1))

$S' \to S$
$S \to Aa$
$S \to Bb$
$S \to bAb$
$S \to bBa$
$A \to a$
$B \to a$

LR(1)
(not LALR(1))

Classes of Grammars



Intuition for why LR parsing is more powerful than LL parsing.

$A \to \alpha$
$A \to \beta$

LL parsing looks at first symbol of RHS of production to decide.

$A \to \alpha \cdot a$
$A \to \beta \cdot b$

LR parsing uses <u>all</u> of RHS <u>plus</u> following symbol(s) to decide.

Classes of Grammars

$S \to aa$

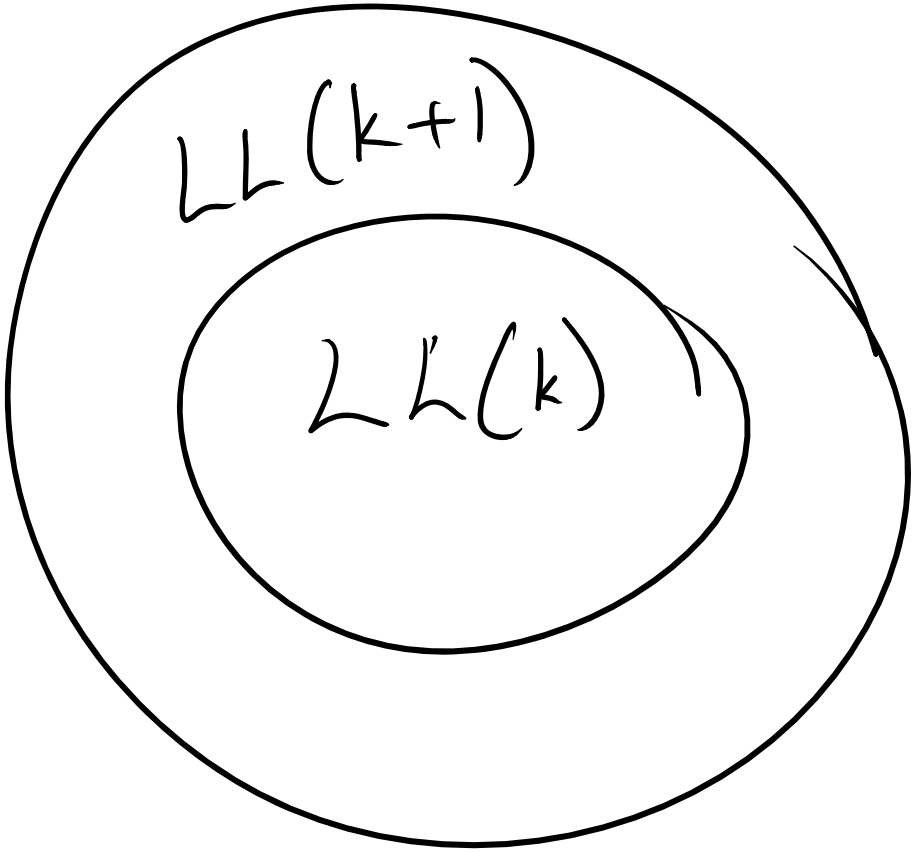
$S \to ab$

↑

LL(2)

but not

LL(1)



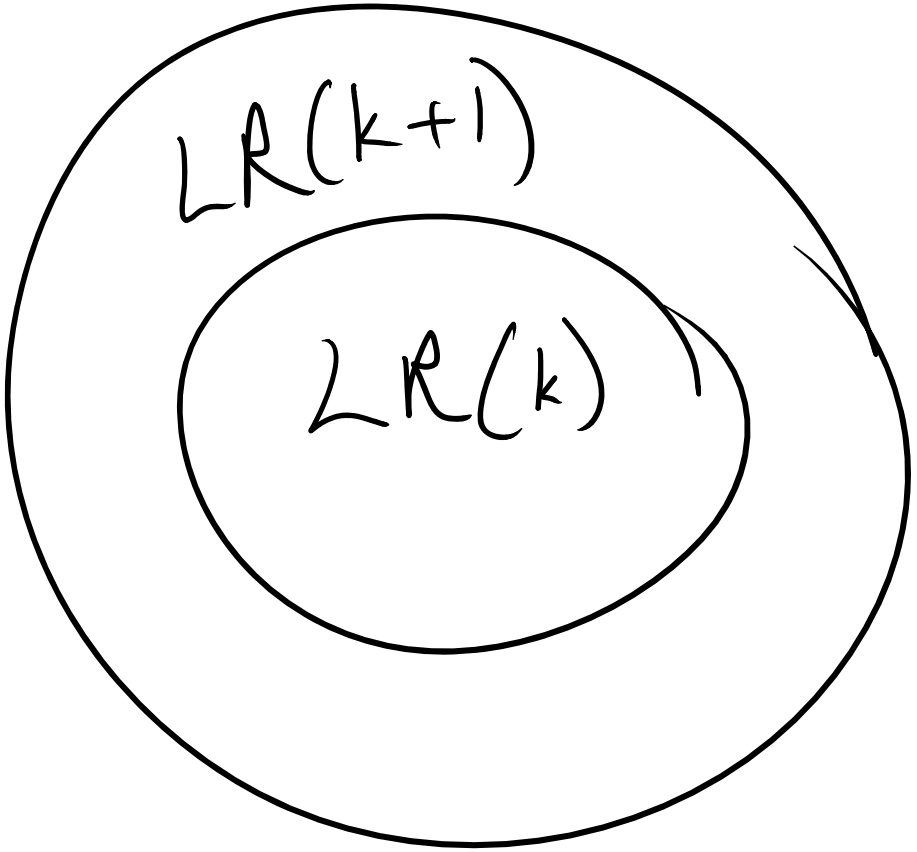
Classes of Grammars

$S \to Aaa$

$S \to Bab$

$A \to a$

$B \to a$



(LR(2) but not LR(1))

Undecidable Problems

Equivalence of CFGs

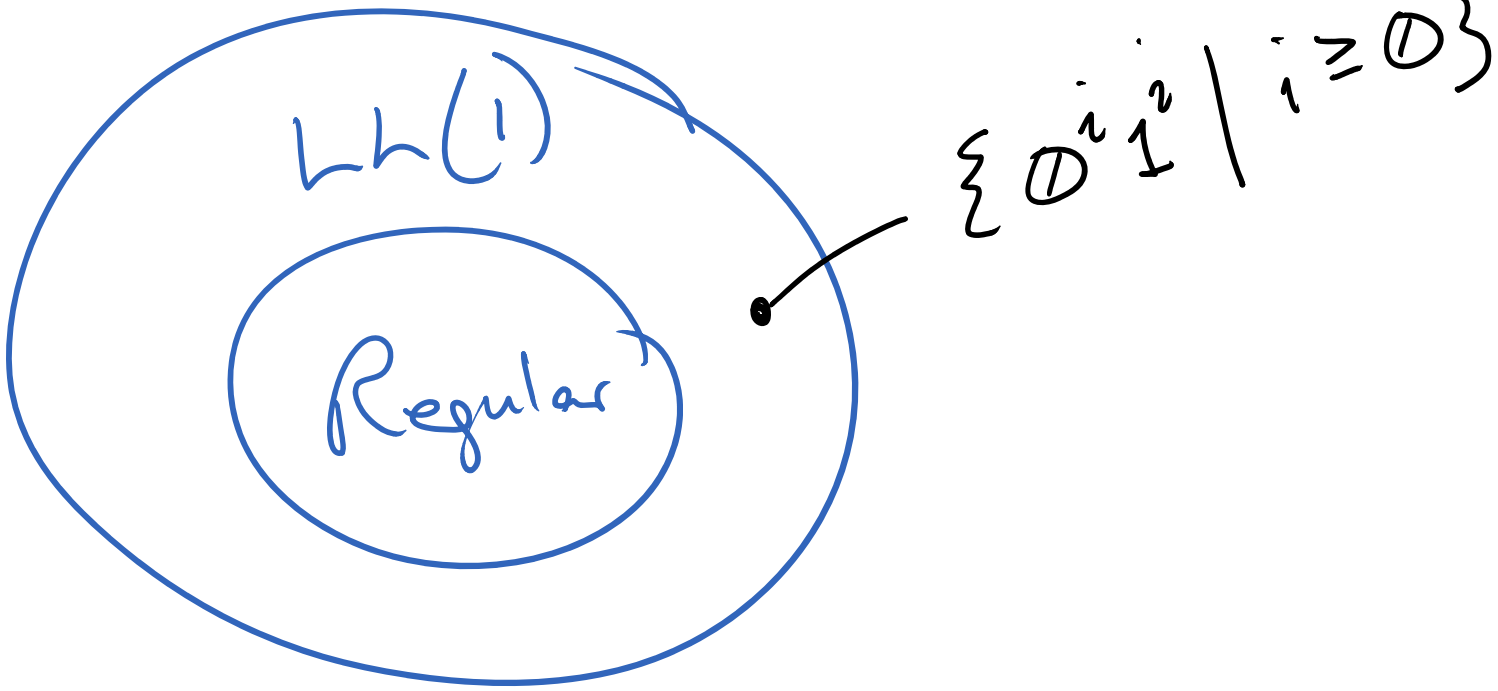
$$\mathcal{L}(G_1) = \mathcal{L}(G_2)\ ?$$

Ambiguity of a CFG. (!)

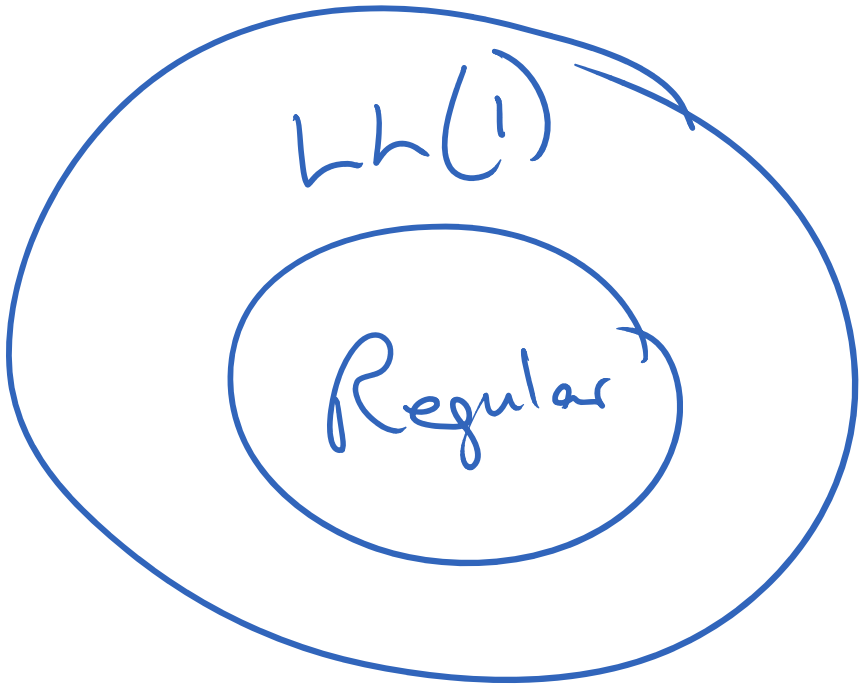
But it is decidable whether G is LL(1) LR(1) etc

If G is LL(1) or LR(1) it is not ambiguous.

Languages



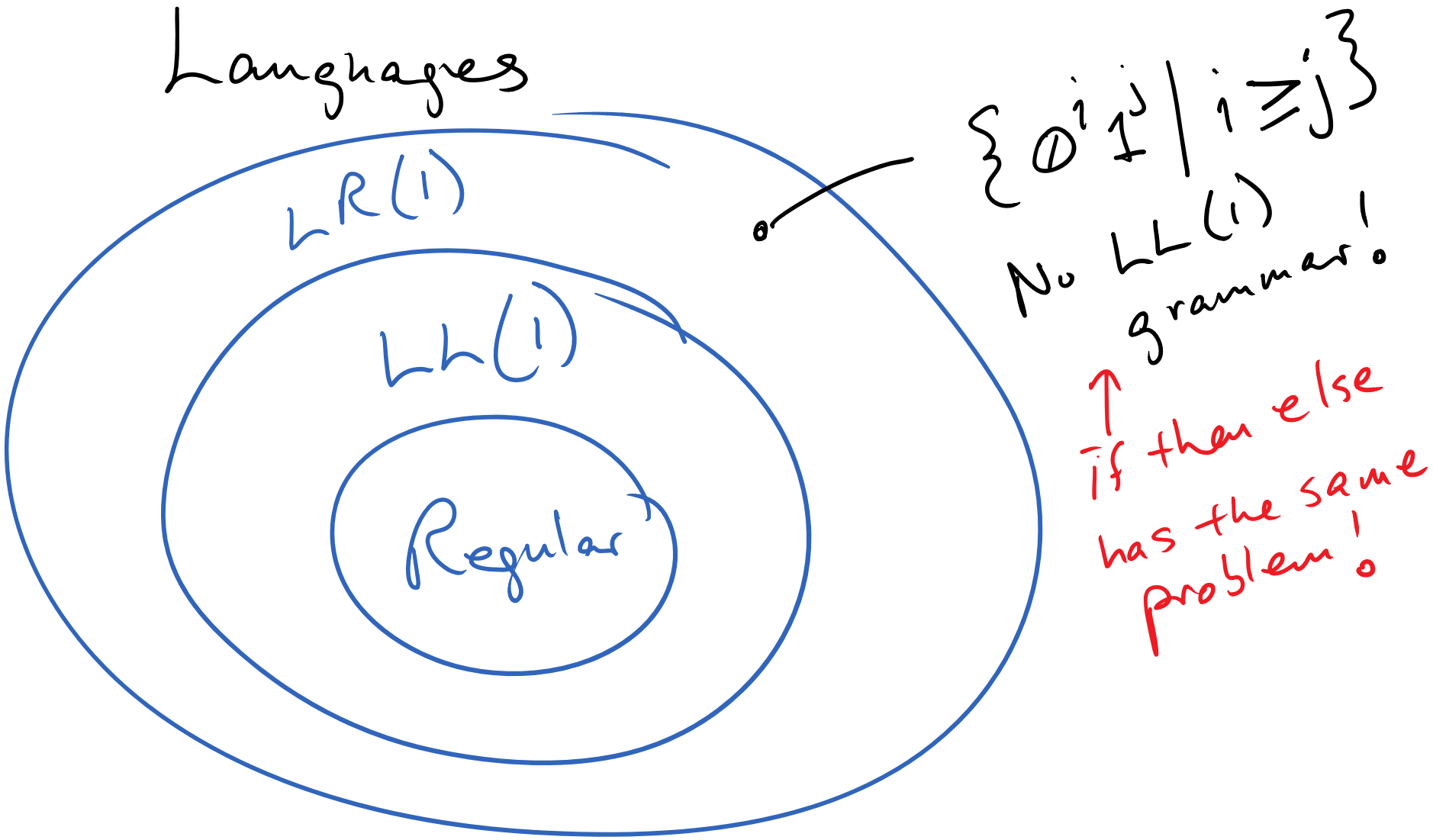
Languages

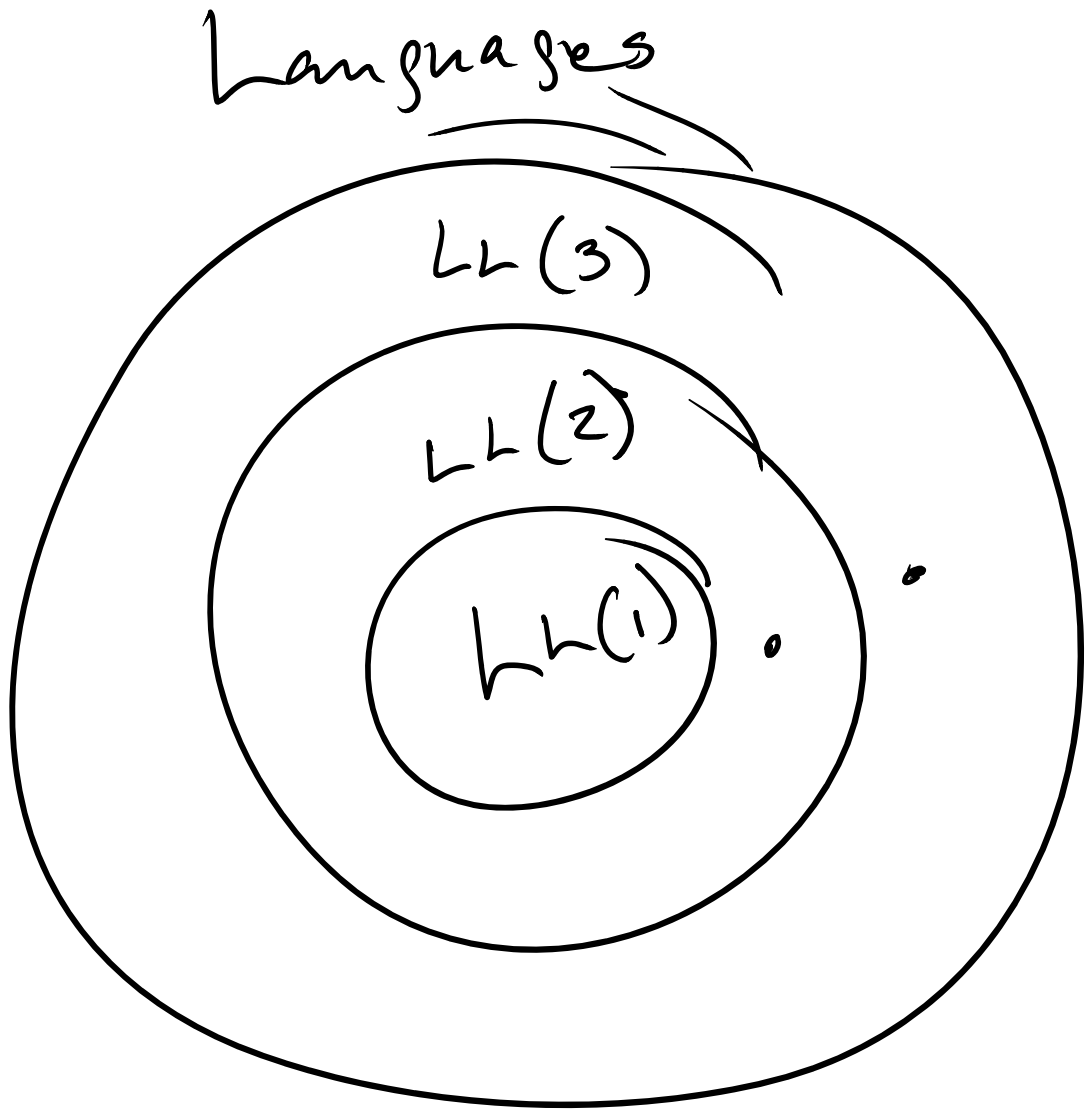


Every regular language can be written as a "right linear CFG"

$A \to w\,B$

at most one nonterminal, always at right end.


Languages
LR(1)
LL(1)
Regular
$\{0^i 1^j \mid i \geq j\}$
No LL(1) grammar.
if then else has the same problem!



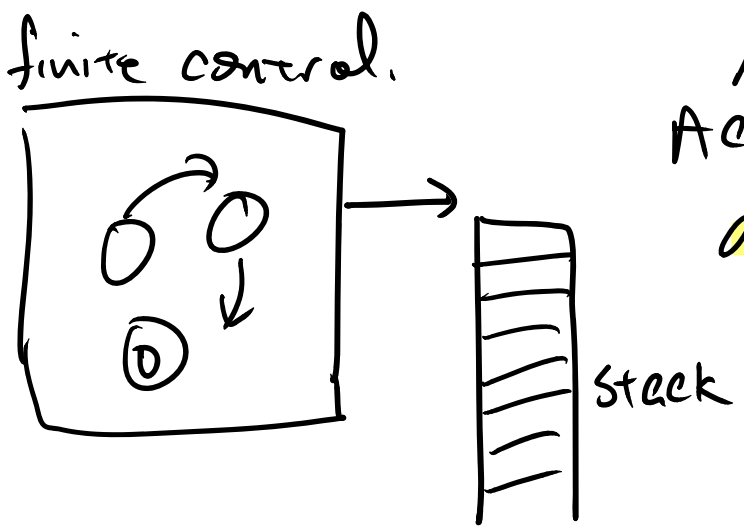
← proper subsets.

There is a language that is LL(2) but not LL(1).

Languages

$LR(k) = LR(1)$ for all $k \geq 1$.

$LR(1) =$ Deterministic Context-Free Languages.



Accepted by a deterministic push-down automaton

reverse

$$\{ w w^R \mid w \in \{a,b\}^* \} - \text{not } LR(1)$$

$\{ w w^R \mid w \in \{a,b\}^* \}$ — not LR(1)

(reverse)

Context-free, but not deterministic

There is no LR(k) grammar.

($\{ w c w \mid w \in \{a,b\}^* \}$ — is LR(1))

Equivalence of deterministic CFLs is decidable

Equivalence of two LR(1) CFGs is decidable

But probably not practical.

Syntax-Directed Translation

Build Abstract Syntax Trees

$E \rightarrow E + E \quad \{ \$\$ = plus(\$1, \$3); \}$

$E \rightarrow E * E \quad \{ \$\$ = mul(\$1, \$3); \}$

$E \rightarrow num \quad \{ \$\$ = \$1; \}$

Compute Directly

$E \rightarrow E + E \quad \{ \$\$ = \$1 + \$3; \}$

$E \rightarrow E * E \quad \{ \$\$ = \$1 * \$3; \}$

$E \rightarrow num \quad \{ \$\$ = \$1; \}$

How does this work?

How Semantic Actions Work

New "value stack" - parallels parse stack.

Shift - value of token is pushed

Reduce - semantic action is executed.

$i is index TOS - (|RHS| - 1)

$$ = $1 = new TOS after reduction

$E \rightarrow E + E \quad \{ \$\$ = \$1 + \$3; \}$

$E \rightarrow E * E \quad \{ \$\$ = \$1 * \$3; \}$

$E \rightarrow num \quad \{ \$\$ = \$1; \}$

Stack	Input
$	1+2*3$
$1 1	+2*3$
$E 1	+2*3$
$E + 1	2*3$
$E+2 1 2	*3$

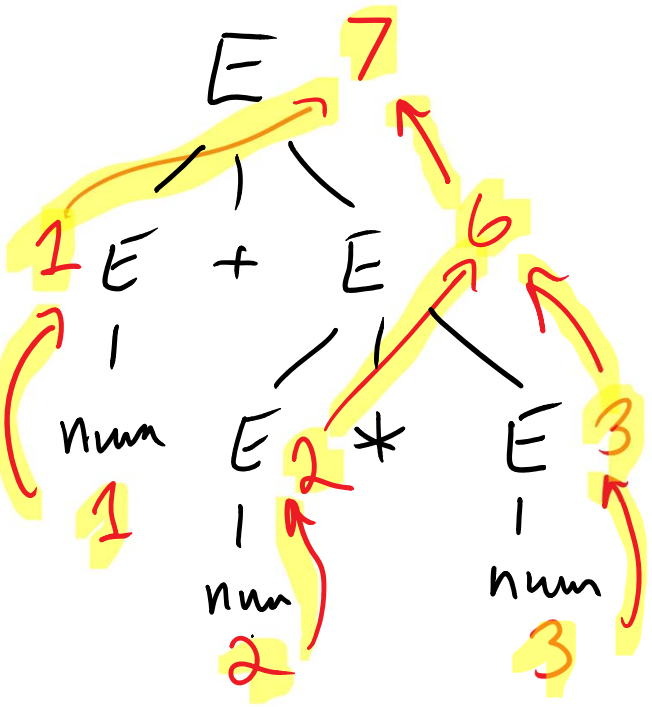
$E \rightarrow E + E \quad \{ \$\$ = \$1 + \$3; \}$

$E \rightarrow E * E \quad \{ \$\$ = \$1 * \$3; \}$

$E \rightarrow num \quad \{ \$\$ = \$1; \}$

Stack	Input
$ E + E	* 3 $
1 2	
$ E + E *	3 $
1 2	
$ E + E * 3	$
1 2 3	
$ E + E * E	$
1 2 3	
$ E + E	$
1 6	
$ E	$
7	

Bottom-up Computation



Announcements

- PA2 due today
- Midterm here (NVIDIA auditorium) Thursday – lecture time
- High probability midterm questions
  - Regular expressions, DFAs for something.
  - Lookahead, start conditions.
  - Which symbols/productions are useless (understand the mathematical definition!)
  - Why is CFG ambiguous?
  - What are FNE, FOLLOW, FIRST of a CFG?
  - Given LL(1) parse table and input, what is sequence of stacks/inputs?
  - Generates some LR(0), LR(1), LALR(1) states.
  - Find SLR(1), LALR(1), LR(1) conflicts
  - Given LR(1) parse table and input, what is sequence of stacks/inputs?

Inherited Values

Values move down the tree.

$E \rightarrow let\ x = E\ in\ E$

$E \rightarrow E + E$

$E \rightarrow E * E$

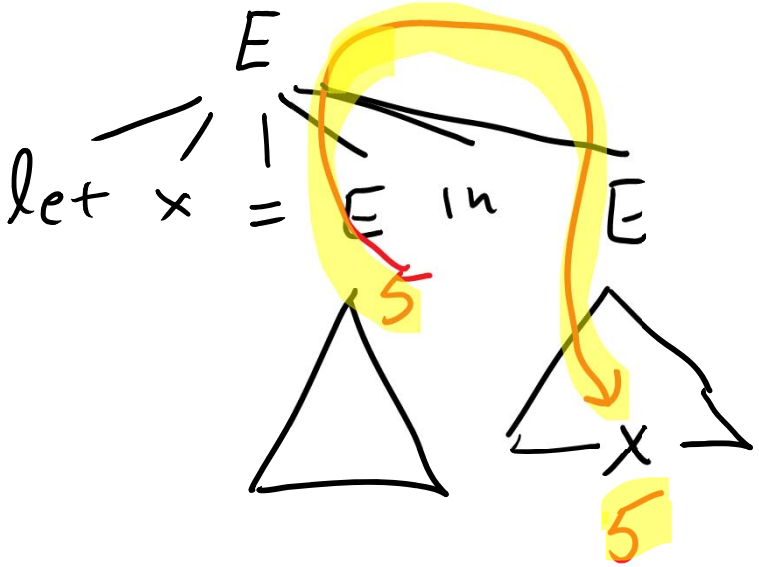
$E \rightarrow num$

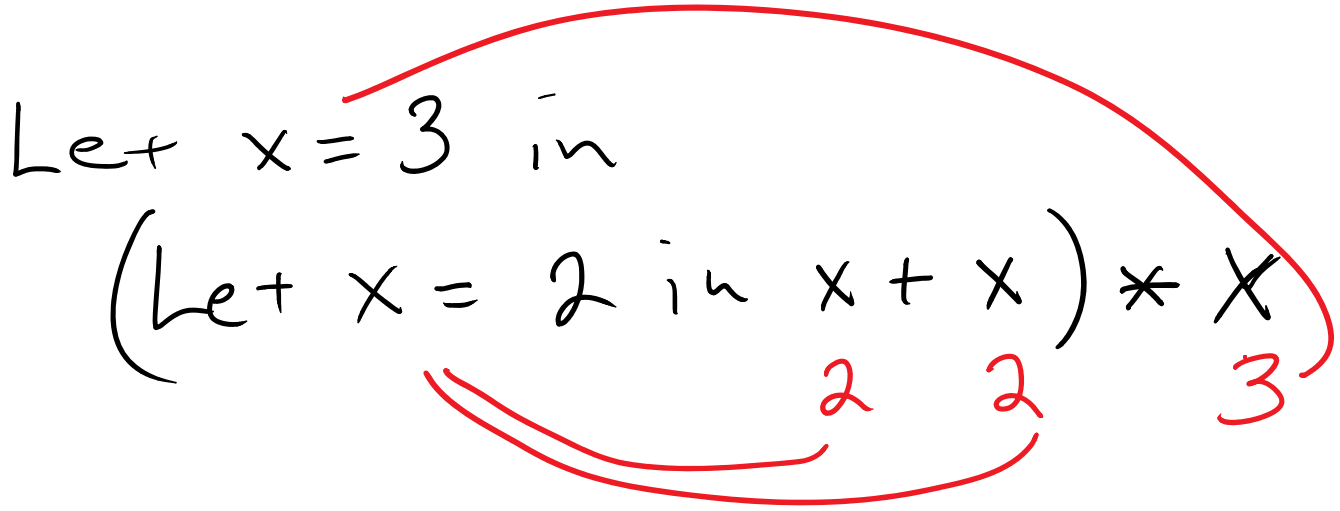
$E \rightarrow x$

$E \rightarrow (E)$

one variable, x, which can be bound to the value of an expression.

← and used


E
let x = E in E
5
x
5



$= 12$

Action "in the middle" of a production

$A \to B \{\$\$ = \$1\}\ C \{\$\$ = \$2 + \$3\}$

↑ this refers to value of M

Bison makes a new nonterminal

$A \to B\ M\ C \{\$\$ = \$2 + \$3\}$

$M \to \varepsilon \{\$\$ = \$1;\}$

Action "in the middle" of a production

$A \rightarrow B \{\$\$ = \$1\} C \{\$\$ = \$2 + \$3\}$

Bison makes a new nonterminal

$A \rightarrow B\ M\ C \quad \{\$\$ = \$2 + \$3\}$

$M \rightarrow \varepsilon \quad \{\$\$ = \$1;\}$

↑ But this refers to "B" in the $A \rightarrow B\ M\ C$ production

Inherited Values

global variable

$E \rightarrow let\ x = E\ \{ x = \$4; \}\ in\ E\ \{\$\$ = \$7\}$

$E \rightarrow E + E\ \{\$\$ = \$1 + \$3;\}$

$E \rightarrow E * E\ \{\$\$ = \$1 + \$3;\}$

$E \rightarrow num\ \{\$\$ = \$1;\}$

$E \rightarrow x\ \{\$\$ = x;\}$

$E \rightarrow (E)\ \{\$\$ = \$2;\}$

Inherited Values

$5

$E \rightarrow$ let $x =$ E { $x = \$4;$ } in E { $\$\$ = \$7$ }

$E \rightarrow E + E$ { $\$\$ = \$1 + \$3;$ }

$E \rightarrow E * E$ { $\$\$ = \$1 + \$3;$ }

$E \rightarrow$ num { $\$\$ = \$1;$ }

$E \rightarrow x$ { $\$\$ = x;$ }

$E \rightarrow (E)$ { $\$\$ = \$2;$ }

Problem

```
Let x = 3 in
    (Let x = 2 in x + x) * x
```

2 2

So parser uses 2, not 3, for this.

When this is parsed, it sets x = 2

Solutions

1. Store x values in a global stack

$E \rightarrow$ Let x = $E\{$xstack.push($4);$\}$ in E

↑ new value of x goes on top of stack.

Solutions

1. Store x values in a global stack

```
E → Let x = E { xstack.push($4); } in E
      { xstack.pop(); $$ = $7; }
```

↑ restore outer x value.

Solutions

1. Store x values in a global stack

```
E → Let x = E { xstack.push($4); } in E
      { xstack.pop(); $$ = $7; }

E → x { $$ = xstack.top(); }
```

get current x value

Solutions

2. Use parser value stack

$E \rightarrow$ Let x = E in { $$ = $4; } E { $$ = $7; }

$E \rightarrow$ x { $$ = $0 }

↑ save x value here

Solutions

2. Use parser value stack

$E \rightarrow$ Let x = E in { $$ = $4; } E { $$ = $7; }

$E \rightarrow x$ { $$ = $0 }

↑ save x value here

x value gets popped off stack when this is reduced.

Solutions

2. Use parser value stack

$E \rightarrow \text{Let } x = E \text{ in } \{\$\$ = \$4;\}\ E\ \{\$\$ = \$7;\}$

$E \rightarrow x\ \{\$\$ = \$0\}$

↑ current x value about to reduce to E

Stack

$ ... Let x = E in M x

↑ $0 ↑ $1

Error Recovery

Challenge

Report errors accurately

11. "Read user's mind"

Especially hard after the first error

Challenge

Report errors accurately

"Read user's mind"

Especially hard after the first error

Most frequent reason for hand written parsers

Panic Mode

Delete tokens until "synchronizing token" is found (e.g. ";")

Resume parsing at that point

error token

YACC and descendents

On parse error

1. Pop stack until top state has an "error" transition
2. Take the transition
3. Delete symbols until it is possible to follow 3 tokens successfully.
4. Resume parsing

Add common errors to lexer / parser

Spelling errors for keywords

Error productions

$E \rightarrow E\ E$ for omitted *

Problems:

Complexity, conflicts

More Elaborate

Find "closest" syntactically correct program.

"Closest" = minimum edit distance

= minimum number of delete/insert/change moves to get to correct program.

1. Computationally costly.
2. Doesn't read user's mind