Bottom-up parsing algorithms

Cocke–Younger–Kasami algorithm

and

Chomsky Normal Form

Last time

Showed how to use JavaCUP for getting ASTs

But we never saw HOW the parser works

This time

Dip our toe into parsing

- Approaches to Parsing
- CFG transformations
  - Useless non-terminals
  - Chomsky Normal Form: A form of grammar that’s easier to deal with
- CYK: powerful, heavyweight approach to parsing

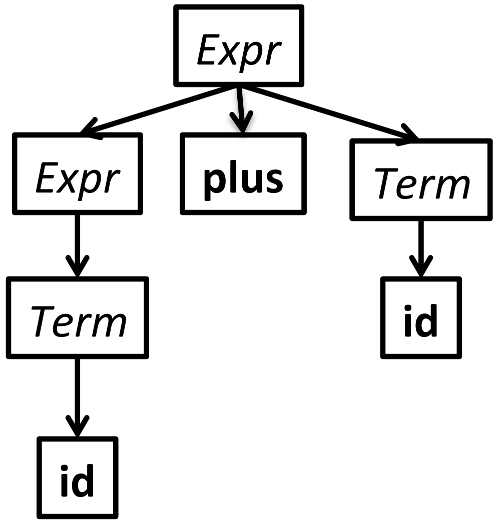
Approaches to parsing

Top Down / “Goal driven”

- Start at root of parse tree, grow downward to match the string

Bottom Up / “Data Driven”

- Start at terminal, generate subtrees until you get to the start



CYK: A general approach to Parsing (*Cocke–Younger–Kasami algorithm*)

Operates in O(n^3)

Works Bottom-Up

Only takes a grammar in Chomsky Normal Form

- This will not turn out to be a limitation

Chomsky Normal Form

All rules must be one of two forms:

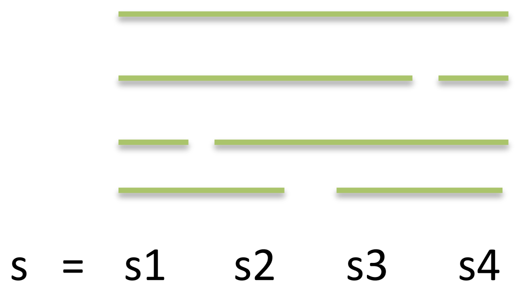
$X \longrightarrow \mathbf{t}$ (terminal)

$X \longrightarrow A\ B$

The only rule allowed to derive epsilon is the start *S*

What CNF buys CYK

- The fact that non-terminals come in pairs allows you to think of a subtree as a subspan of the input
- The fact that non-terminals are not nullable (except for start) means that each subspan has at least one character



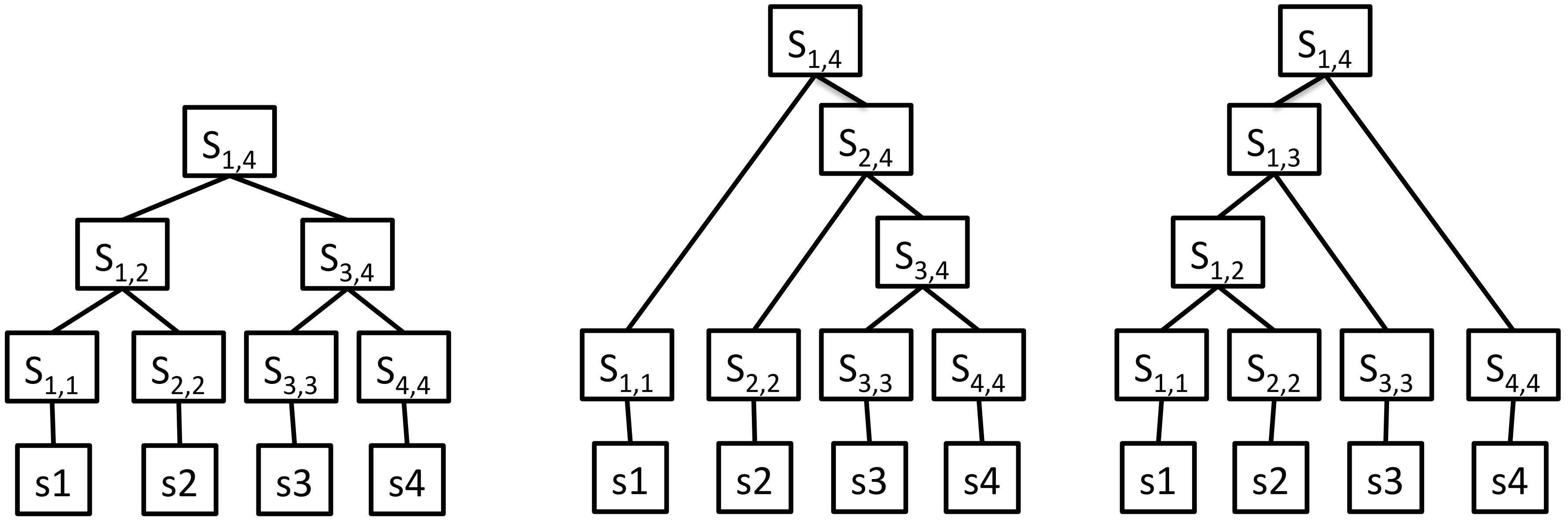
CYK: Dynamic Programming

$X \longrightarrow \mathbf{t}$

Prods. form the leaves of the parse tree

$X \longrightarrow A\ B$

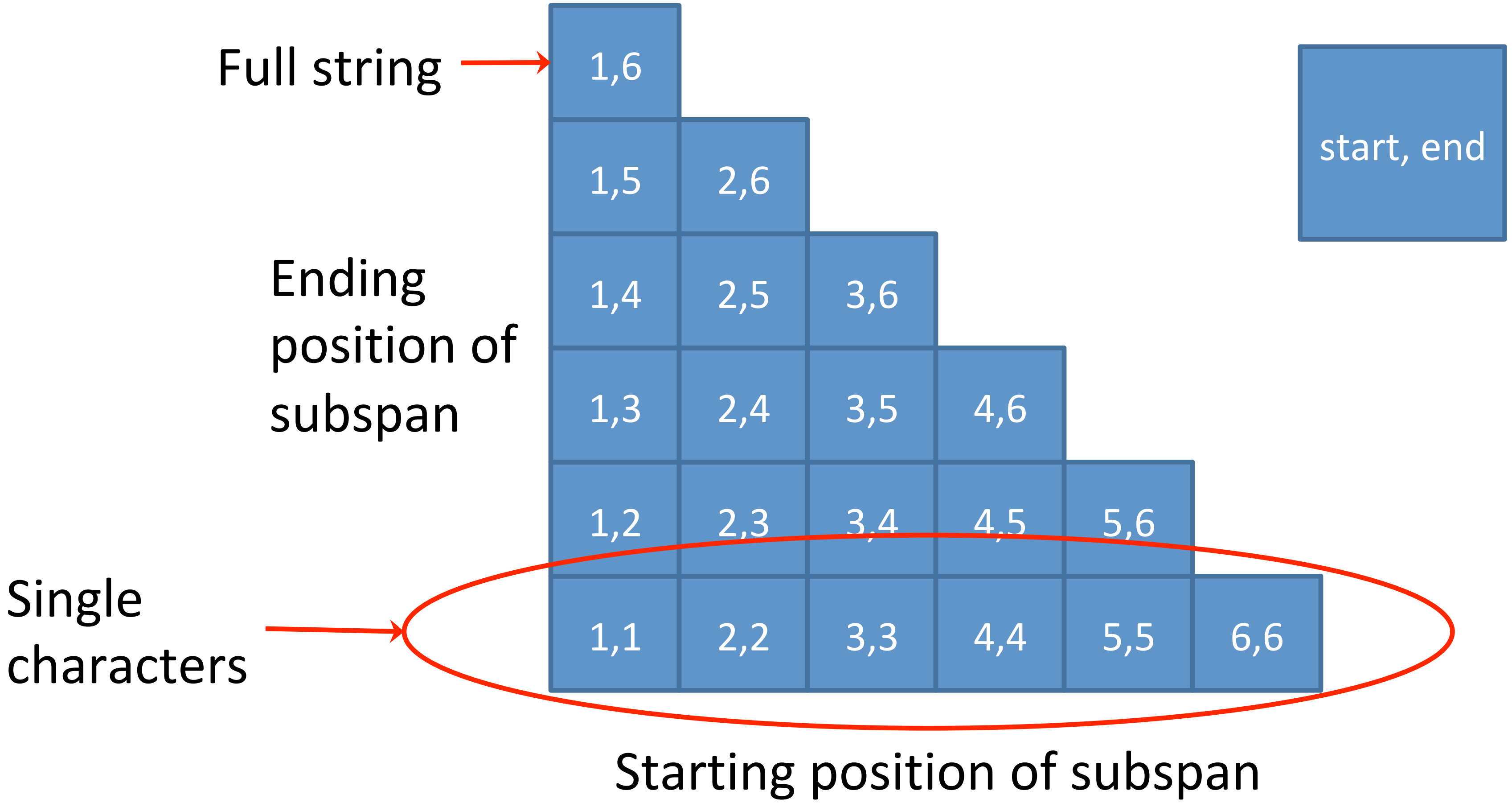
Form binary nodes



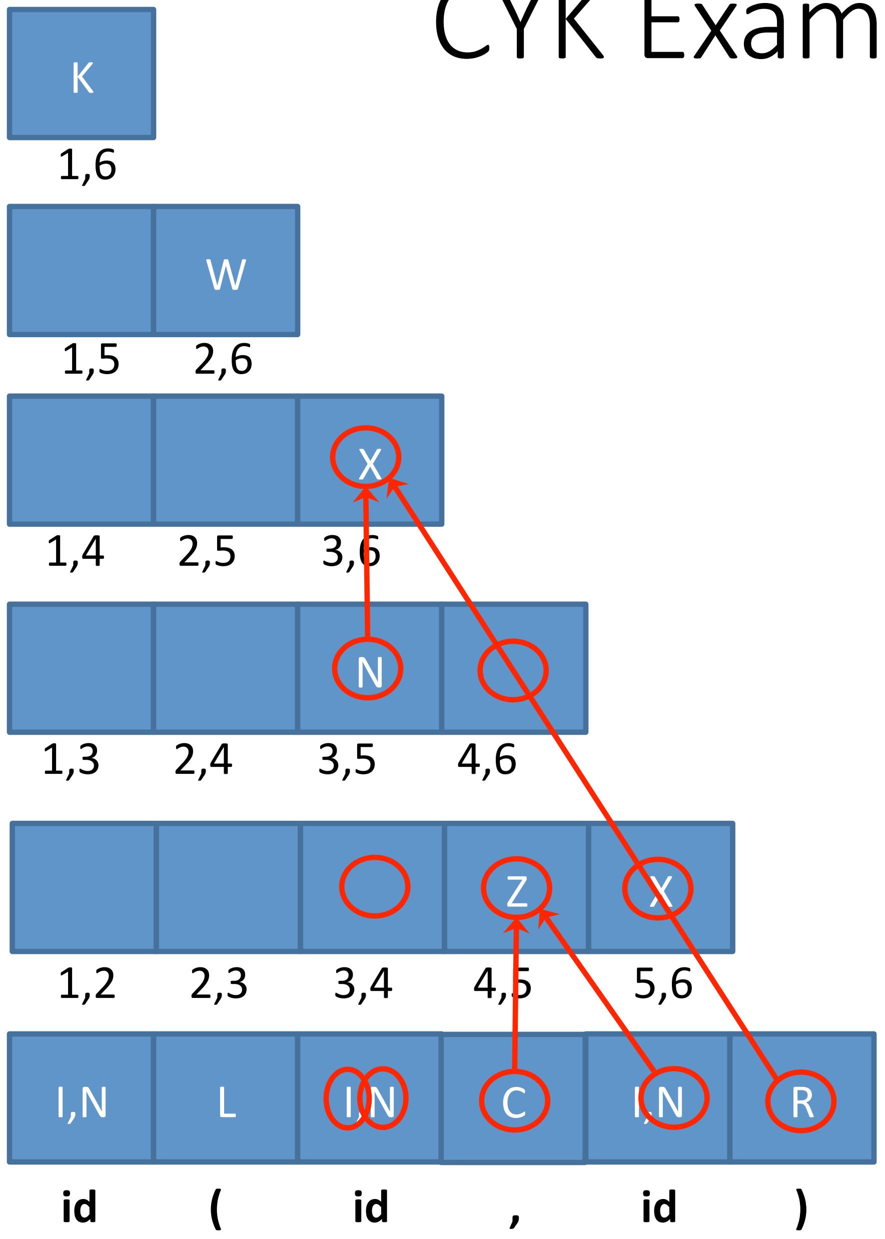
$S_{1,4}$
$S_{1,2}$ $S_{3,4}$
$S_{1,1}$ $S_{2,2}$ $S_{3,3}$ $S_{4,4}$
s1 s2 s3 s4
$S_{1,4}$
$S_{2,4}$
$S_{3,4}$
$S_{1,1}$ $S_{2,2}$ $S_{3,3}$ $S_{4,4}$
s1 s2 s3 s4
$S_{1,4}$
$S_{1,3}$
$S_{1,2}$
$S_{1,1}$ $S_{2,2}$ $S_{3,3}$ $S_{4,4}$
s1 s2 s3 s4

Running CYK …

Track every viable subtree from leaf to root. Here are all the subspans for a string of 6 terminals:

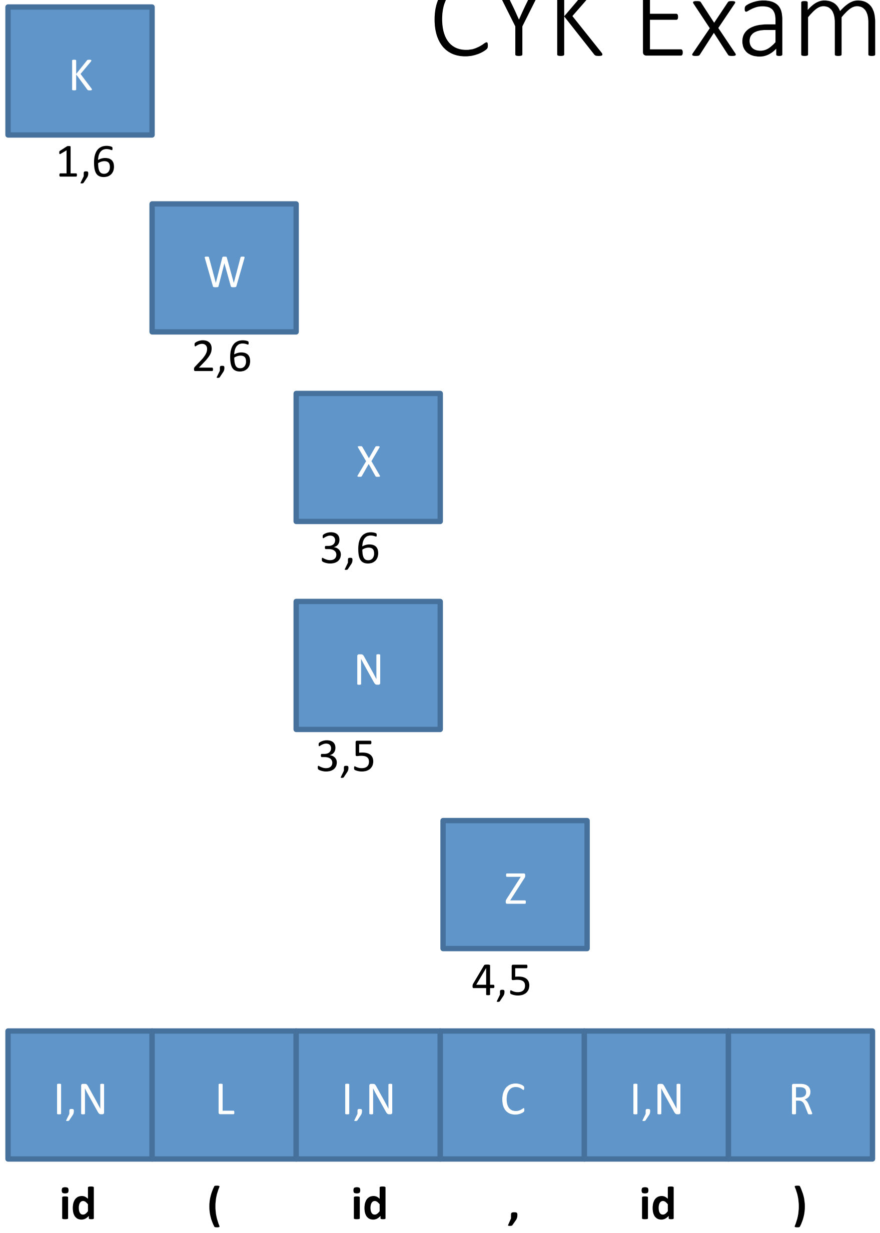


CYK Example



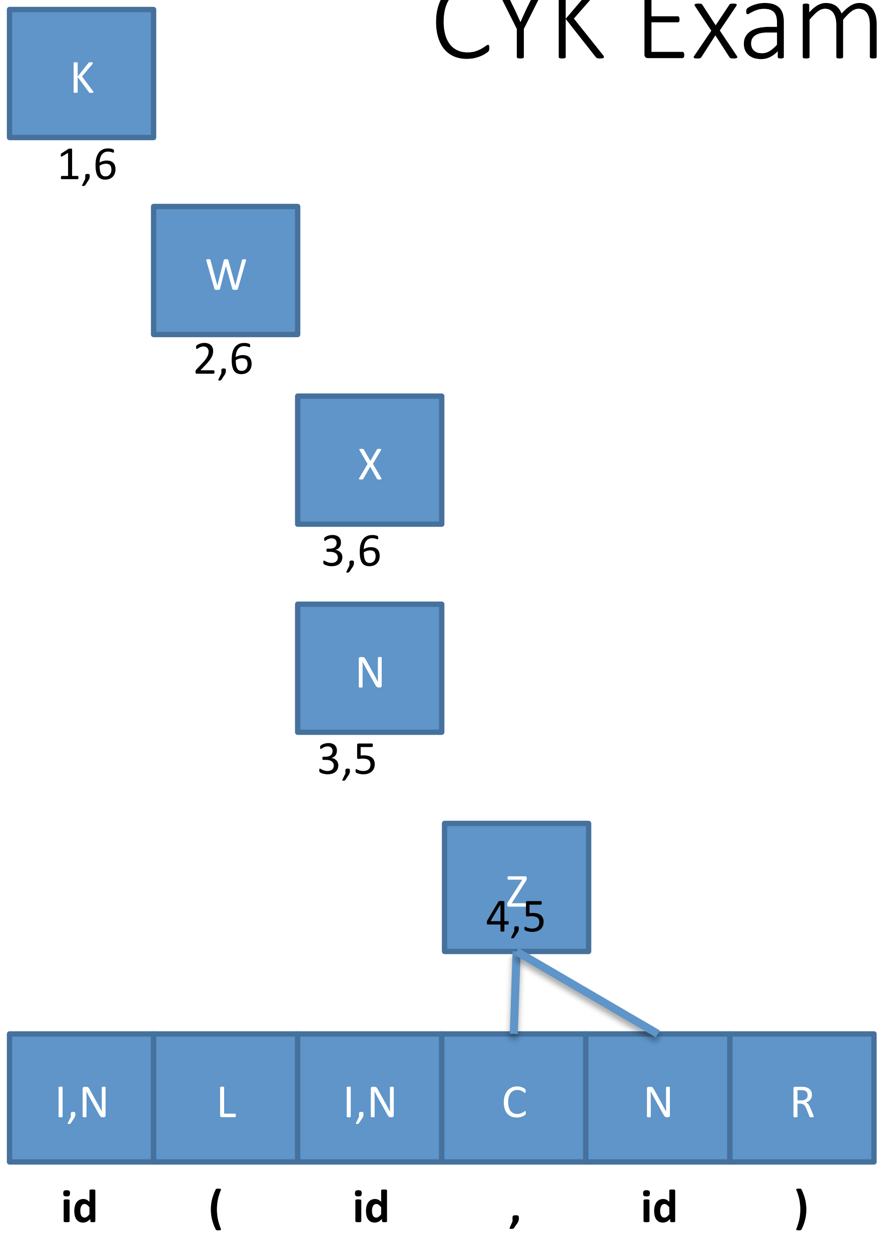
K	$\longrightarrow$	I W
K	$\longrightarrow$	I Y
W	$\longrightarrow$	L X
X	$\longrightarrow$	N R
Y	$\longrightarrow$	L R
N	$\longrightarrow$	**id**
N	$\longrightarrow$	I Z
Z	$\longrightarrow$	C N
I	$\longrightarrow$	**id**
L	$\longrightarrow$	**(**
R	$\longrightarrow$	**)**
C	$\longrightarrow$	**,**

CYK Example



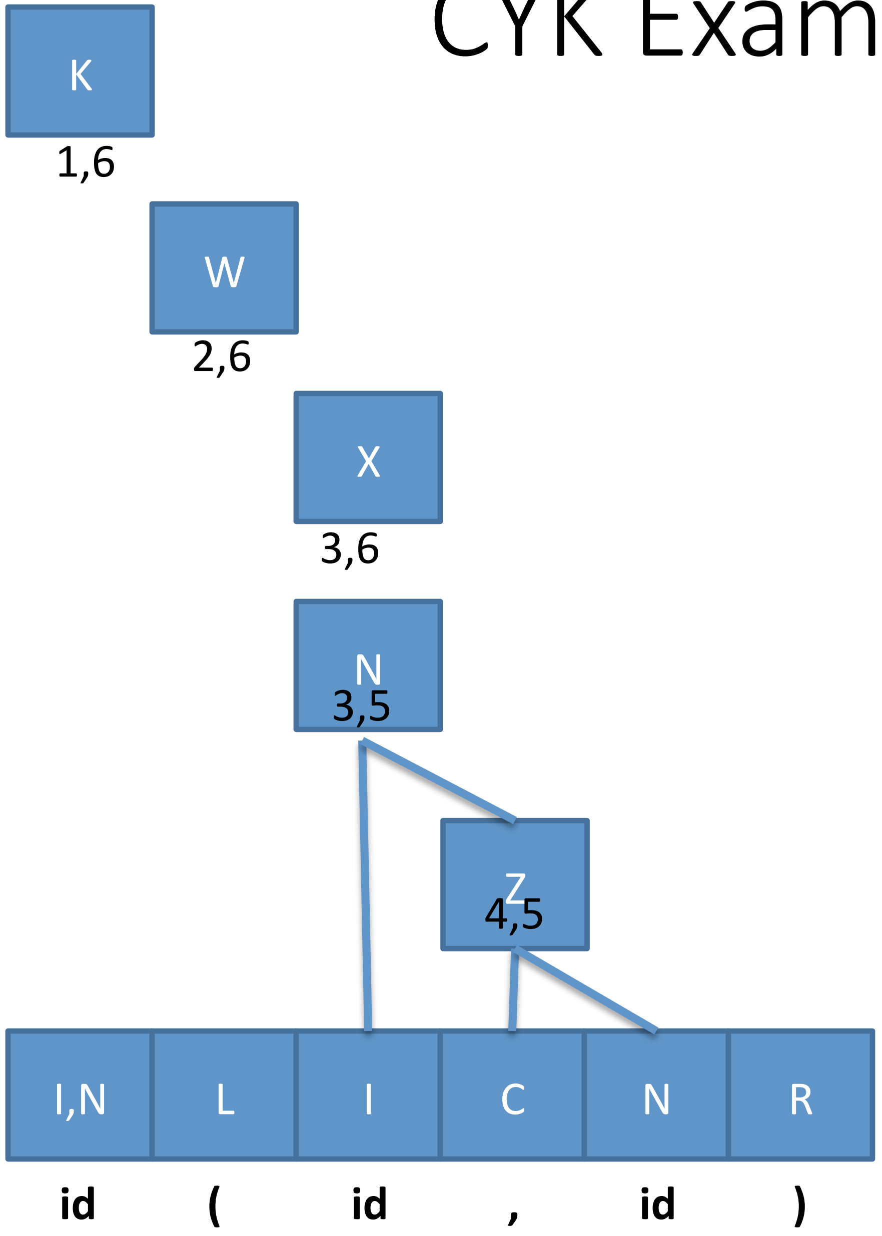
K	→	I W
K	→	I Y
W	→	L X
X	→	N R
Y	→	L R
N	→	**id**
N	→	I Z
Z	→	C N
I	→	**id**
L	→	**(**
R	→	**)**
C	→	**,**

CYK Example



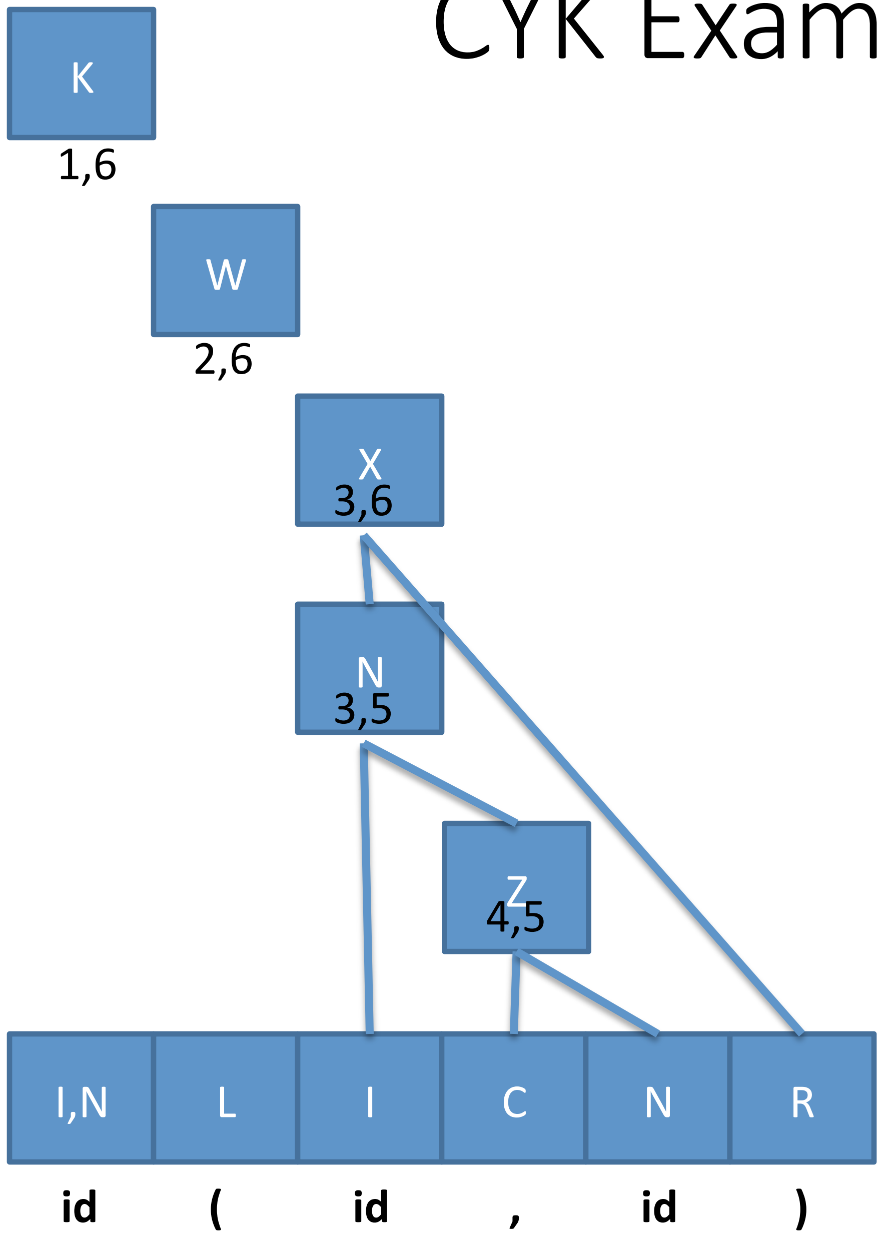
K	$\longrightarrow$	I W
K	$\longrightarrow$	I Y
W	$\longrightarrow$	L X
X	$\longrightarrow$	N R
Y	$\longrightarrow$	L R
N	$\longrightarrow$	**id**
N	$\longrightarrow$	I Z
Z	$\longrightarrow$	C N
I	$\longrightarrow$	**id**
L	$\longrightarrow$	**(**
R	$\longrightarrow$	**)**
C	$\longrightarrow$	**,**

CYK Example



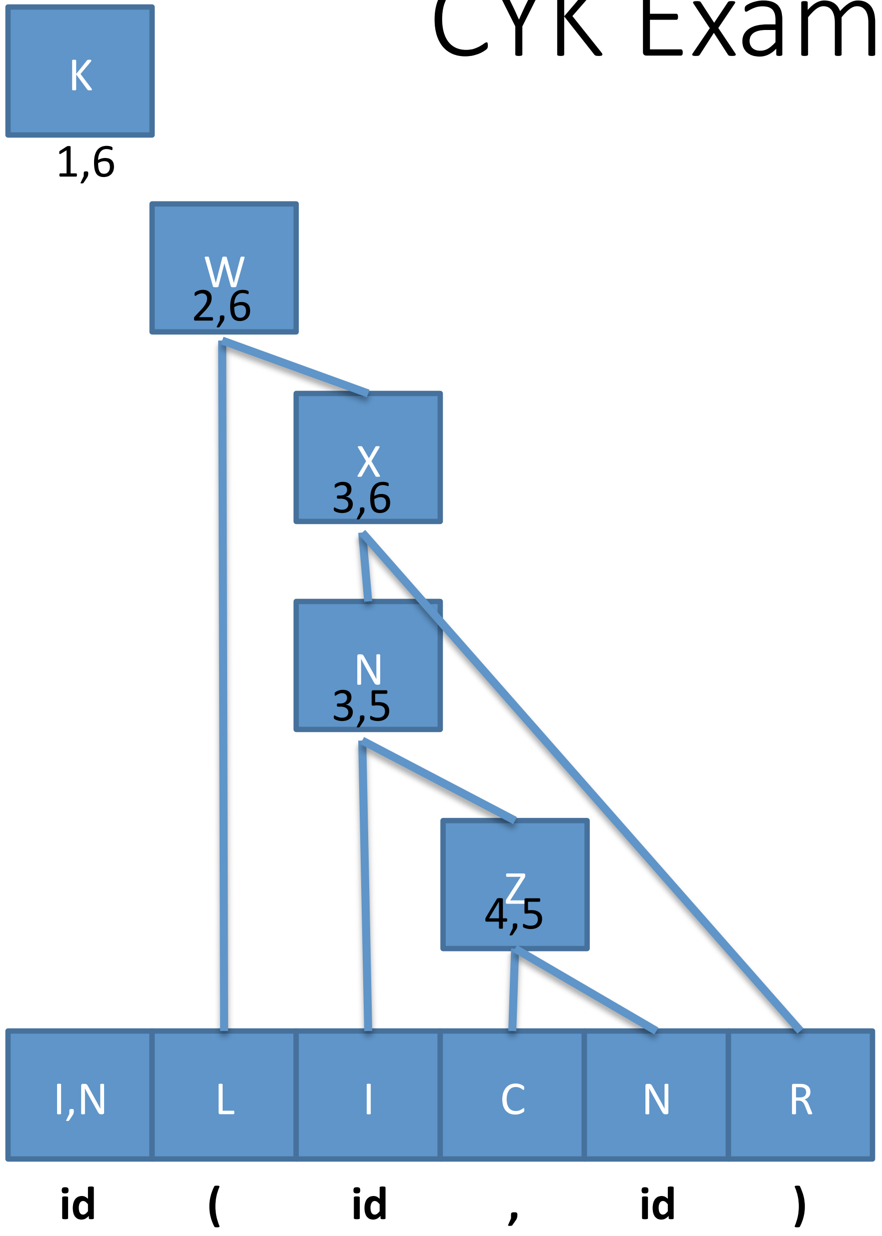
K	$\longrightarrow$	I W
K	$\longrightarrow$	I Y
W	$\longrightarrow$	L X
X	$\longrightarrow$	N R
Y	$\longrightarrow$	L R
N	$\longrightarrow$	**id**
N	$\longrightarrow$	I Z
Z	$\longrightarrow$	C N
I	$\longrightarrow$	**id**
L	$\longrightarrow$	**(**
R	$\longrightarrow$	**)**
C	$\longrightarrow$	**,**

CYK Example



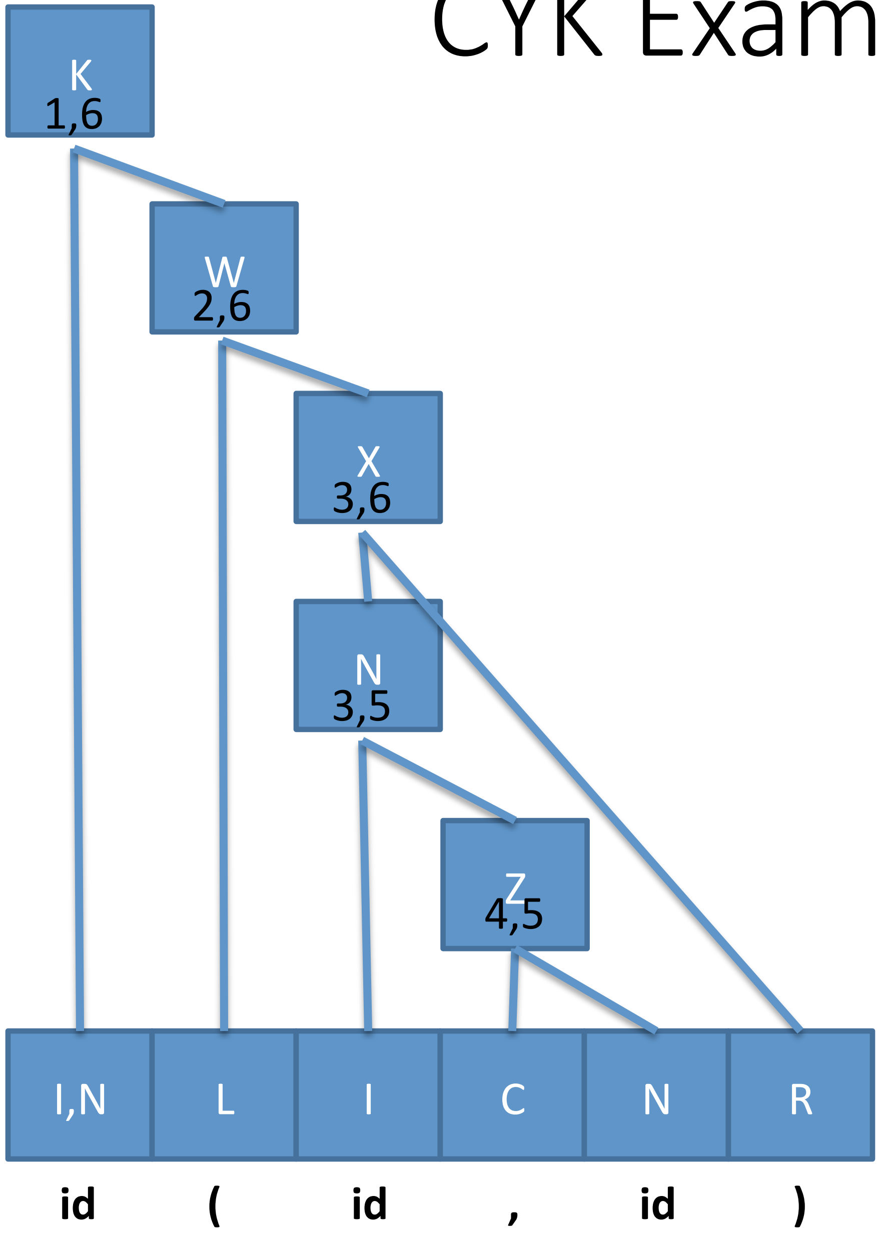
K	$\longrightarrow$	I W
K	$\longrightarrow$	I Y
W	$\longrightarrow$	L X
X	$\longrightarrow$	N R
Y	$\longrightarrow$	L R
N	$\longrightarrow$	**id**
N	$\longrightarrow$	I Z
Z	$\longrightarrow$	C N
I	$\longrightarrow$	**id**
L	$\longrightarrow$	**(**
R	$\longrightarrow$	**)**
C	$\longrightarrow$	**,**

CYK Example



K	⟶	I W
K	⟶	I Y
W	⟶	L X
X	⟶	N R
Y	⟶	L R
N	⟶	**id**
N	⟶	I Z
Z	⟶	C N
I	⟶	**id**
L	⟶	**(**
R	⟶	**)**
C	⟶	**,**

CYK Example



K	⟶	I W
K	⟶	I Y
W	⟶	L X
X	⟶	N R
Y	⟶	L R
N	⟶	**id**
N	⟶	I Z
Z	⟶	C N
I	⟶	**id**
L	⟶	**(**
R	⟶	**)**
C	⟶	**,**

Cleaning up our grammars

We want to avoid unnecessary work

- Remove *useless* rules

Eliminating Useless Nonterminals

1. If a nonterminal cannot derive a sequence of terminal symbols then it is useless
2. If a nonterminal cannot be derived from the start symbol, then it is useless

Eliminate Useless Nonterms

If a nonterminal cannot derive a sequence of terminal symbols, then it is useless

```
Mark all terminal symbols
Repeat
   If all symbols on the
   righthand side of a
   production are marked
      mark the lefthand side
Until no more non-terminals
can be marked
```

Example:

S	$\longrightarrow$	X | Y
X	$\longrightarrow$	( )
Y	$\longrightarrow$	(Y Y)

Eliminate Useless Nonterms

If a nonterminal cannot be derived from the start symbol, then it is useless

```
Mark the start symbol
Repeat
    If the lefthand side of a
    production is marked
        mark all righthand
        non-terminal
Until no more non-terminals
can be marked
```

Example:

S	⟶	A B
A	⟶	**+** | **-** | ε
B	⟶	**digit** | B **digit**
C	⟶	**.** B

Chomsky Normal Form

4 Steps

- Eliminate epsilon rules
- Eliminate unit rules
- Fix productions with terminals on RHS
- Fix productions with > 2 nonterminals on RHS

Eliminate (Most) Epsilon Productions

If a nonterminal *A* immediately derives epsilon

- Make copies of all rules with *A* on the RHS and delete all combinations of *A* in those copies

Example 1

F	→	**id (** A **)**
A	→	ε
A	→	N
N	→	**id**
N	→	**id ,** N

F	→	**id (** A **)**
F	→	**id ( )**
A	→	N
N	→	**id**
N	→	**id ,** N

Example 2

X	→	*A* **x** *A* **y** *A*
A	→	ε
A	→	**z**

X	→	*A* **x** *A* **y** *A*
	|	*A* **x** *A* **y**
	|	*A* **x** **y** *A*
	|	*A* **x** **y**
	|	**x** *A* **y** *A*
	|	**x** *A* **y**
	|	**x** **y** *A*
	|	**x** **y**
A	→	**z**

Eliminate Unit Productions

Productions of the form $A \longrightarrow B$ are called unit productions

Place B anywhere A could have appeared and remove the unit production

Example 1

F	→	**id (** A **)**
F	→	**id ( )**
A	→	N
N	→	**id**
N	→	**id ,** N

$$\Big\Downarrow$$

F	→	**id (** N **)**
F	→	**id ( )**
N	→	**id**
N	→	**id ,** N

Fix RHS Terminals

For productions with Terminals and something else on the RHS

- For each terminal t add the rule

  $X \longrightarrow \mathrm{t}$

  Where X is a new non-terminal

- Replace t with X in the in the original rules

Example

F	⟶	**id (N)**
F	⟶	**id ( )**
N	⟶	**id**
N	⟶	**id , N**

F	⟶	I L N R
F	⟶	I L R
N	⟶	**id**
N	⟶	I C N
I	⟶	**id**
L	⟶	**(**
R	⟶	**)**
C	⟶	**,**

Fix RHS Nonterminals

For productions with > 2 Nonterminals on the RHS

- Replace all but the *first* nonterminal with a new nonterminal
- Add a rule from the new nonterminal to the replaced nonterminal sequence
- Repeat

Example

$F \rightarrow I\ L\ N\ R$

$$\Downarrow$$

$F \rightarrow I\ W$

$W \rightarrow L\ N\ R$

$$\Downarrow$$

$F \rightarrow I\ W$

$W \rightarrow L\ X$

$X \rightarrow N\ R$

Parsing is Tough

CYK parses an arbitrary CFG, but

- $O(n^3)$
- Too slow!

For special class of grammars

- $O(n)$
- Includes LL(1) and LALR(1)

Classes of Grammars

LL(1)
- Scans input from Left-to-right (first L)
- Builds a Leftmost Derivation (second L)
- Can peek (1) token ahead of the token being parsed
- Top-down “predictive parsers”

LALR(1)
- Uses special lookahead procedure (LA)
- Scans input from Left-to-right (second L)
- Rightmost derivation (R)
- Can also peek (1) token ahead

LALR(1) strictly more powerful, much harder to understand

In summary

We talked about how to parse with CYK and Chomsky Normal Form grammars