http://lara.epfl.ch

Compiler Construction 2011

CYK Algorithm for Parsing General Context-Free Grammars

Why Parse General Grammars

- Can be difficult or impossible to make grammar unambiguous
  - thus LL(k) and LR(k) methods cannot work, for such ambiguous grammars
- Some inputs are more complex than simple programming languages
  - mathematical formulas:
    x = y /\ z   ?   (x=y) /\ z   x = (y /\ z)
  - natural language:

    I saw the man with the telescope.
  - future programming languages

Ambiguity

1)

2)

I saw the man with the telescope.

CYK Parsing Algorithm

C:

John **Cocke** and Jacob T. Schwartz (1970). Programming languages and their compilers: Preliminary notes. Technical report, Courant Institute of Mathematical Sciences, New York University.

Y:

Daniel H. **Younger** (1967). Recognition and parsing of context-free languages in time n^3. *Information and Control* 10(2): 189–208.

K:

T. **Kasami** (1965). An efficient recognition and syntax-analysis algorithm for context-free languages. Scientific report AFCRL-65-758, Air Force Cambridge Research Lab, Bedford, MA.

Two Steps in the Algorithm

1) Transform grammar to normal form

   called Chomsky Normal Form

   (Noam Chomsky, mathematical linguist)

2) Parse input using transformed grammar

   dynamic programming algorithm

"a method for solving complex problems by breaking them down into simpler steps.
It is applicable to problems exhibiting the properties of overlapping subproblems" (>WP)

Balanced Parentheses Grammar

Exercise:
- copy normal form grammar
- for each rule of type (1) in normal form indicate rules in original grammar

Original grammar G

$S \rightarrow$ "" | (S) | S S

(1) (2) (3)

Modified grammar in Chomsky Normal Form:

$S \rightarrow$ "" | S' ← if "" $\in L(G)$ (0)

$S' \rightarrow N_{(}\, N_{S)} \mid N_{(}\, N_{)} \mid S'\, S'$ (2, 2, 3)

$N_{S)} \rightarrow S'_{2} N_{)}$

Rules (1): $N \rightarrow N_1 N_2$ (nonterminals)

$N_{(} \rightarrow$ (

$N_{)} \rightarrow$)

Rules (2): $N \rightarrow t$ (N: nonterminal, t: terminal)

- Terminals: ( ) Nonterminals: S S' $N_{S)}$ $N_{)}$ $N_{(}$

($N_{S)}$: nonterminal with funny name)

Idea How We Obtained the Grammar

$S \rightarrow (\; S \;)$

because S can be empty
but S' cannot

$S' \rightarrow N_(\; N_{S)} \quad | \; N_(\; N_)$

$N_(\rightarrow ($

$N_{S)} \rightarrow S' \; N_)$

$N_) \rightarrow)$

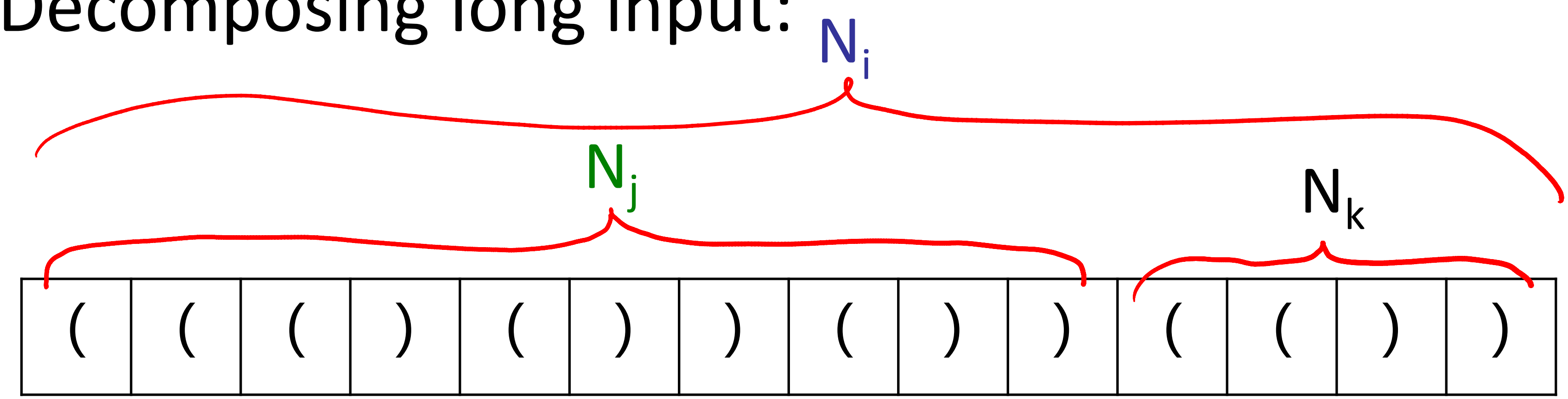
Chomsky Normal Form transformation
can be done fully mechanically

Dynamic Programming to Parse Input

Assume Chomsky Normal Form, 3 types of rules:

$S \rightarrow$ “” | S'	(only for the start non-terminal)
$N_j \rightarrow t$	(names for terminals)
$N_i \rightarrow N_j\ N_k$	(just **2** non-terminals on RHS)

Decomposing long input:



find all ways to parse substrings of length 1,2,3,…

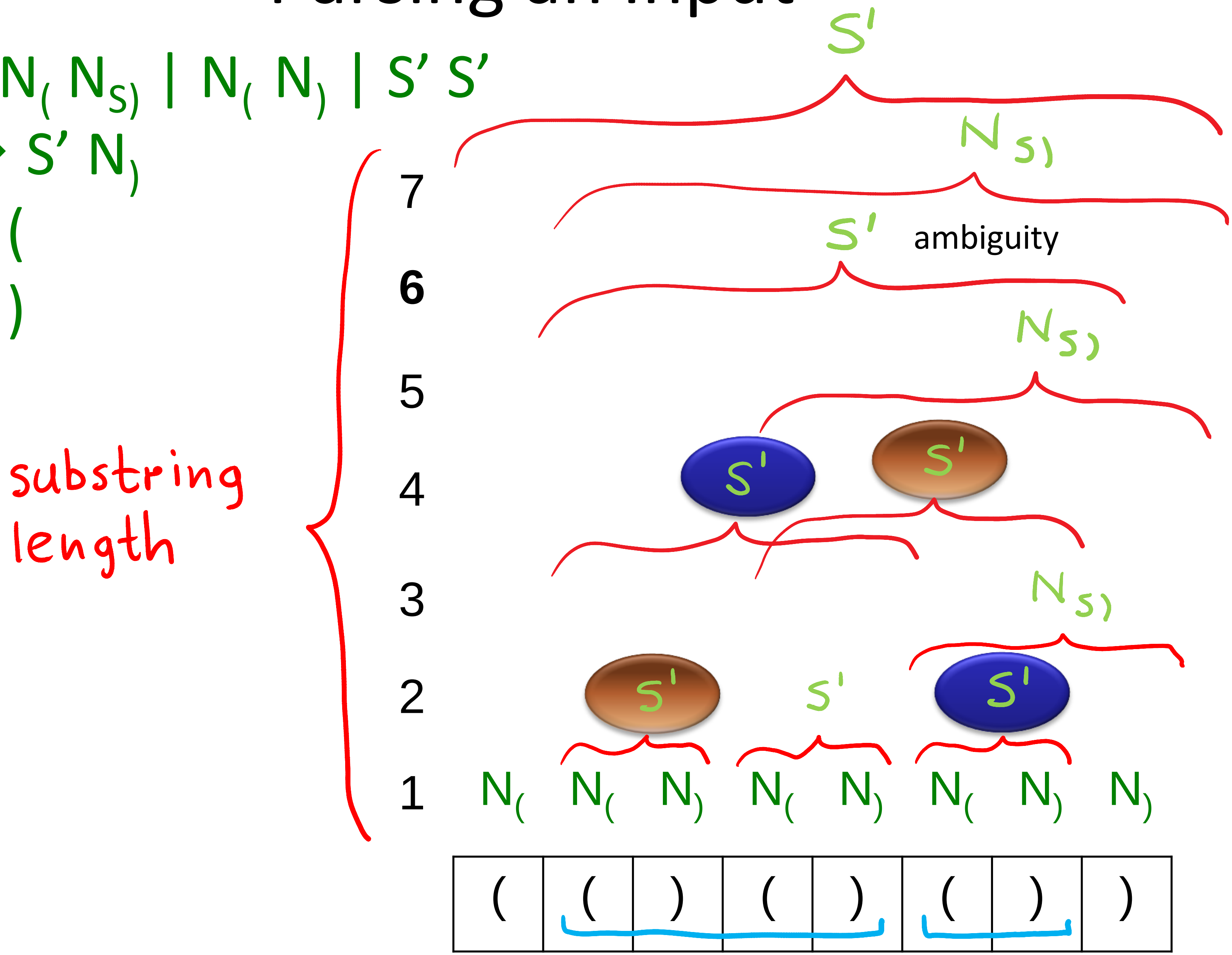
Parsing an Input

$S' \rightarrow N_(\; N_{S)} \mid N_(\; N_) \mid S' \; S'$

$N_{S)} \rightarrow S' \; N_)$

$N_(\rightarrow ($

$N_) \rightarrow)$



Algorithm Idea

$S' \rightarrow S' \; S'$

w_{pq} – substring from p to q

d_{pq} – all non-terminals that could expand to w_{pq}

Initially d_{pp} has $N_{w(p,p)}$

key step of the algorithm:

if $X \rightarrow Y\,Z$ is a rule,

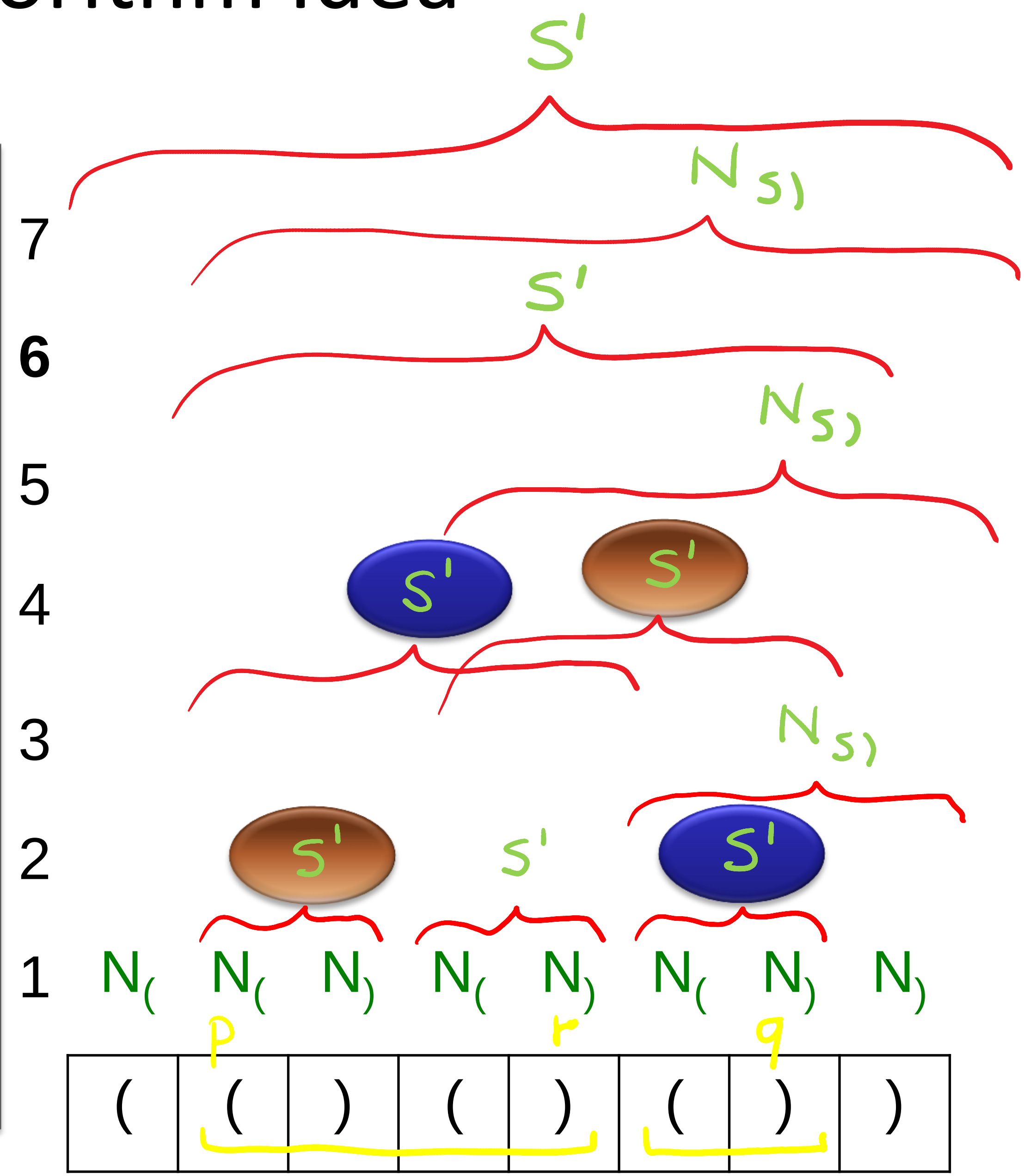
Y is in $d_{p\,r}$, and

Z is in $d_{(r+1)q}$

then put X into d_{pq}

$(p \leq r < q)$,

in increasing value of (q-p)



Algorithm

INPUT: grammar G in Chomsky normal form
word w to parse using G

OUTPUT: true iff (w in L(G))

```
N = |w|
var d : Array[N][N]
for p = 1 to N {
  d(p)(p) = {X | G contains X->w(p)}
  for q in {p + 1 .. N} d(p)(q) = {} }
for k = 2 to N // substring length
 for p = 0 to N-k // initial position
  for j = 1 to k-1 // length of first half
   val r = p+j-1; val q = p+k-1;
   for (X::=Y Z) in G
    if Y in d(p)(r) and Z in d(r+1)(q)
      d(p)(q) = d(p)(q) union {X}
return  S in d(0)(N-1)
```

What is the running time as a function of grammar size and the size of input?

$O(N^3 \cdot |G|)$