Left-corner parsing

Laura Kassner

laura.kassner@gmx.de

Computational Linguistics II: Parsing

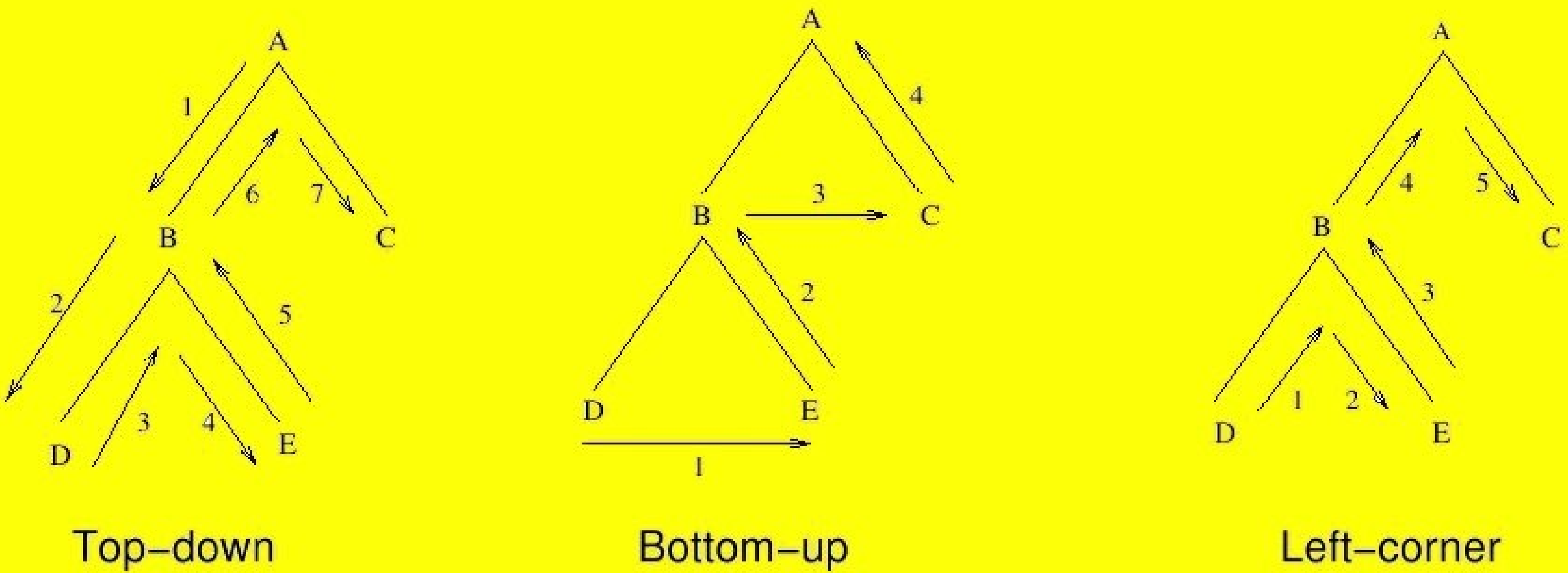
January 10th, 2007

Left-corner parsing

- Basics
- Building a left-corner recognizer…
- … and transforming it into a parser
- Comparison to top-down and bottom-up approaches

Left-corner parsing: Basics

What is left-corner parsing?



picture taken from Shravan Vasishth's HSP seminar slides

Left-corner parsing: Basics

- bottom-up and top-down aspects

Left-corner parsing: Basics

- bottom-up and top-down aspects
- bottom-up: rule k_0 -> k_1... k_n can only be applied if for every k_i (1 <= i <= n), a complete partial structure has been recognized

Left-corner parsing: Basics

- bottom-up and top-down aspects
- bottom-up: rule k_0 -> k_1... k_n can only be applied if for every k_i (1 <= i <= n), a complete partial structure has been recognized
- left-corner: a structure dominated by k_1 must have been recognized for a rule to be applied

Left-corner parsing: Basics

- bottom-up and top-down aspects
- bottom-up: rule k_0 -> k_1… k_n can only be applied if for every k_i $(1 <= i <= n)$, a complete partial structure has been recognized
- left-corner: a structure dominated by k_1 must have been recognized for a rule to be applied

=> k_1 is “left corner“ of the rule – first symbol on the right hand side

Left-corner parsing: Basics

- bottom-up and top-down aspects
- bottom-up: rule k_0 -> k_1... k_n can only be applied if for every k_i ($1 <= i <= n$), a complete partial structure has been recognized
- left-corner: a structure dominated by k_1 must have been recognized for a rule to be applied

=> k_1 is "left corner" of the rule – first symbol on the right hand side

=> rule used to make assumptions about the category dominating k_1 and about following constituents

Left-corner parsing: Basics

What is a left-corner parse?

Left-corner parsing: Basics

What is a left-corner parse?

- context-free grammar G = <N, T, S, R>

Left-corner parsing: Basics

What is a left-corner parse?

- context-free grammar G = <N, T, S, R>
- string w

Left-corner parsing: Basics

What is a left-corner parse?

- context-free grammar G = <N, T, S, R>
- string w

=> series of rule indices $\gamma = i_1 \dots i_n$ which corresponds to a derivation of string w in G

Left-corner parsing: Basics

Ordering rules:

Left-corner parsing: Basics

Ordering rules:

1 – β ist the tree structure implied by γ

Left-corner parsing: Basics

Ordering rules:

1 – β ist the tree structure implied by γ

2 – nodes in β are ordered the following way:

Left-corner parsing: Basics

Ordering rules:

1 – β ist the tree structure implied by γ

2 – nodes in β are ordered the following way:

- a) if n DD n_1 … n_m, all nodes of the subtree with root n_1 are in front of n;

Left-corner parsing: Basics

Ordering rules:

1 – β ist the tree structure implied by γ

2 – nodes in β are ordered the following way:

a) if n DD n_1 … n_m, all nodes of the subtree with root n_1 are in front of n;

b) n is in front of all other nodes it dominates

Left-corner parsing: Basics

Ordering rules:

1 – β ist the tree structure implied by γ

2 – nodes in β are ordered the following way:

a) if n DD n_1 … n_m, all nodes of the subtree with root n_1 are in front of n;

b) n is in front of all other nodes it dominates

c) all nodes dominated by n_i are in front of the nodes dominated by n_{i+1}

Left-corner parsing: Basics

Ordering rules:

1 – β ist the tree structure implied by γ

2 – nodes in β are ordered the following way:

a) if n DD n_1 … n_m, all nodes of the subtree with root n_1 are in front of n;

b) n is in front of all other nodes it dominates

c) all nodes dominated by n_i are in front of the nodes dominated by n_{i+1}

3 – the order of rule applications described by γ does not violate these rules

Left-corner parsing: Basics

=> inorder tree traversal!!!

Left-corner parsing: Basics

An example:

Left-corner parsing: Basics

An example:

- grammar rules: **1: S -> AS** **2: S-> BB** **3: A -> bAA**

  4: A -> a **5: B -> b** **6: B -> e**

Left-corner parsing: Basics

An example:

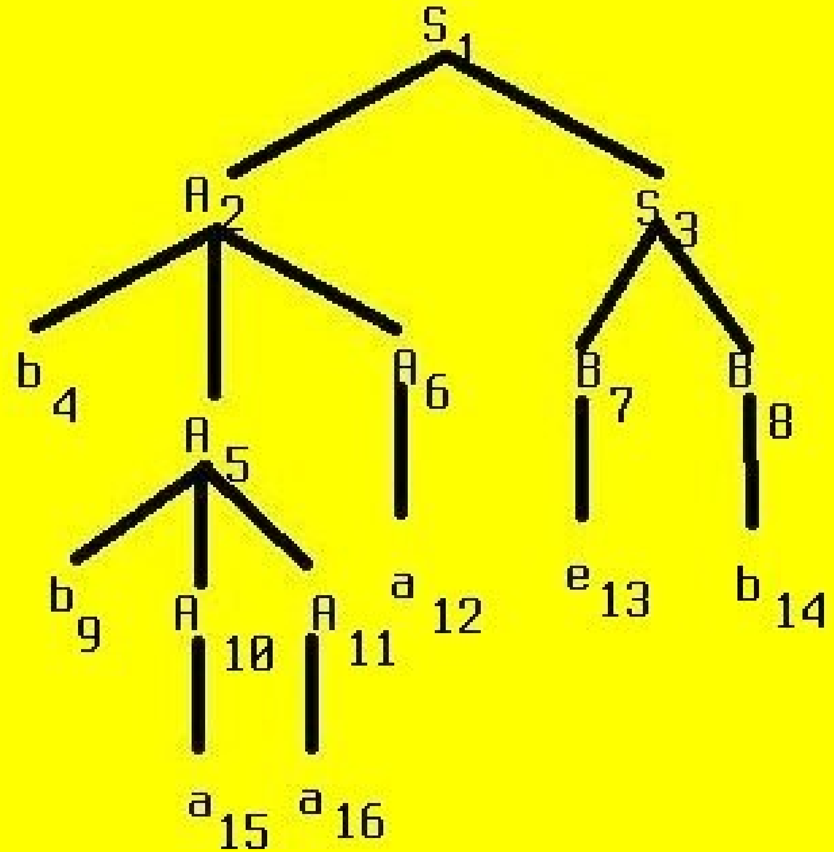
- grammar rules: **1: S -> AS** **2: S-> BB** **3: A -> bAA**

  4: A -> a **5: B -> b** **6: B -> e**
- sentence: bbaaab

Left-corner parsing: Basics

An example:

- grammar rules: **1: S -> AS** **2: S-> BB** **3: A -> bAA** **4: A -> a** **5: B -> b** **6: B -> e**
- sentence: bbaaab



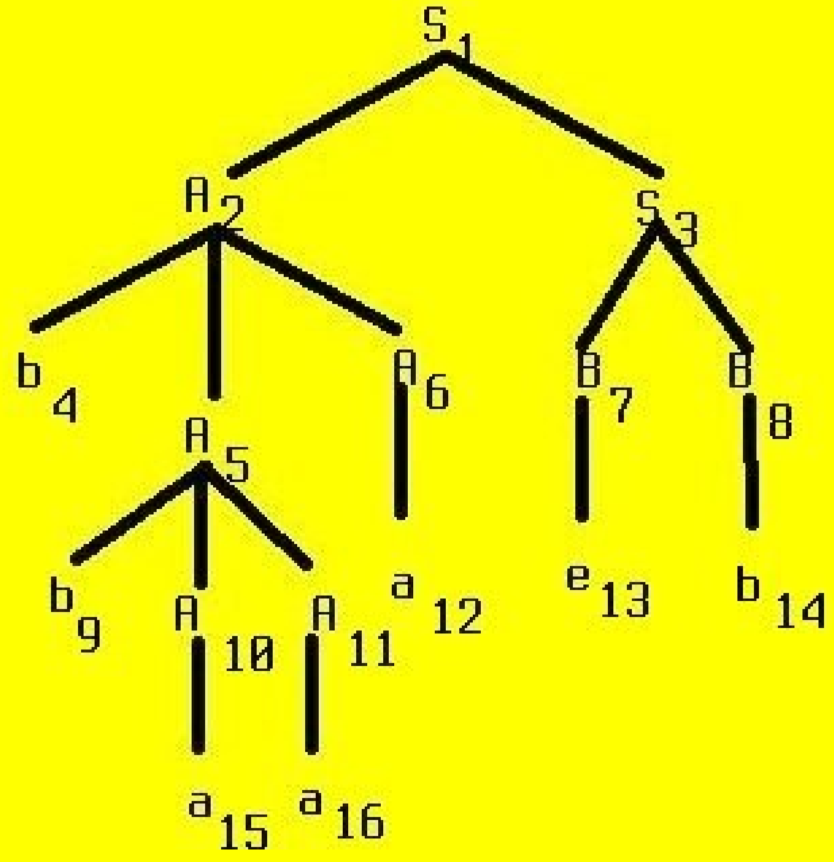
Left-corner parsing: Basics

An example:

- grammar rules: **1: S -> AS** **2: S-> BB** **3: A -> bAA** **4: A -> a** **5: B -> b** **6: B -> e**
- sentence: bbaaab

Order of nodes:

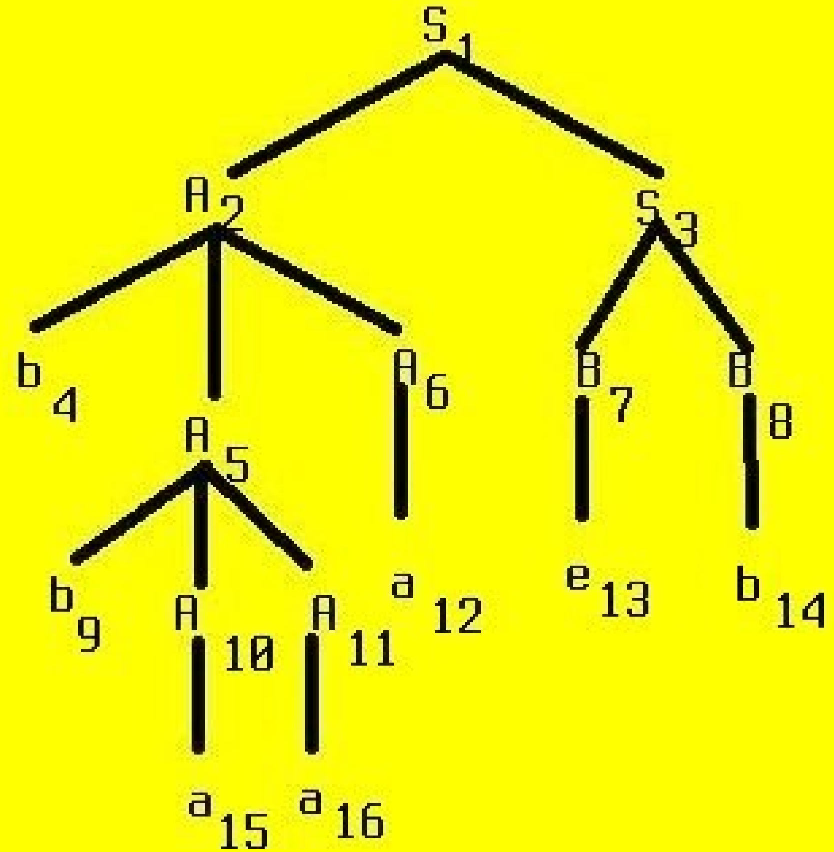
4 2 9 5 15 10 16 11 12 6 1 13 7 3 14 8



Left-corner parsing: Basics

An example:

- grammar rules: **1: S -> AS** **2: S-> BB** **3: A -> bAA** **4: A -> a** **5: B -> b** **6: B -> e**
- sentence: bbaaab



Order of nodes:

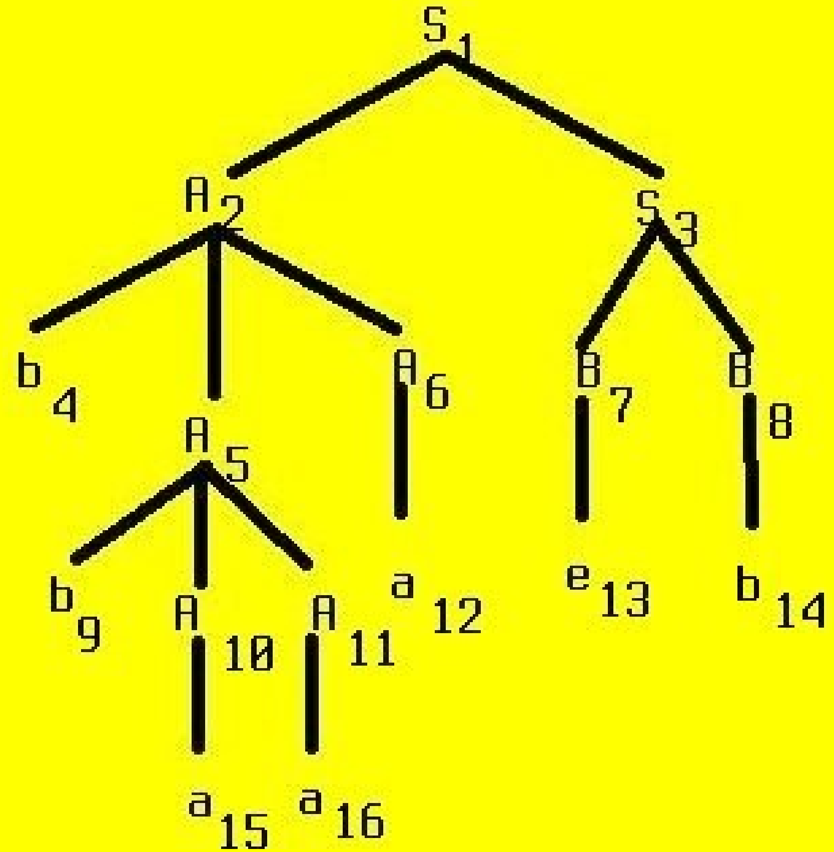
4 2 9 5 15 10 16 11 12 6 1 13 7 3 14 8

TD parse: 1 3 3 4 4 4 2 6 5

Left-corner parsing: Basics

An example:

- grammar rules: **1: S -> AS** **2: S-> BB** **3: A -> bAA** **4: A -> a** **5: B -> b** **6: B -> e**
- sentence: bbaaab



Order of nodes:

4 2 9 5 15 10 16 11 12 6 1 13 7

3 14 8

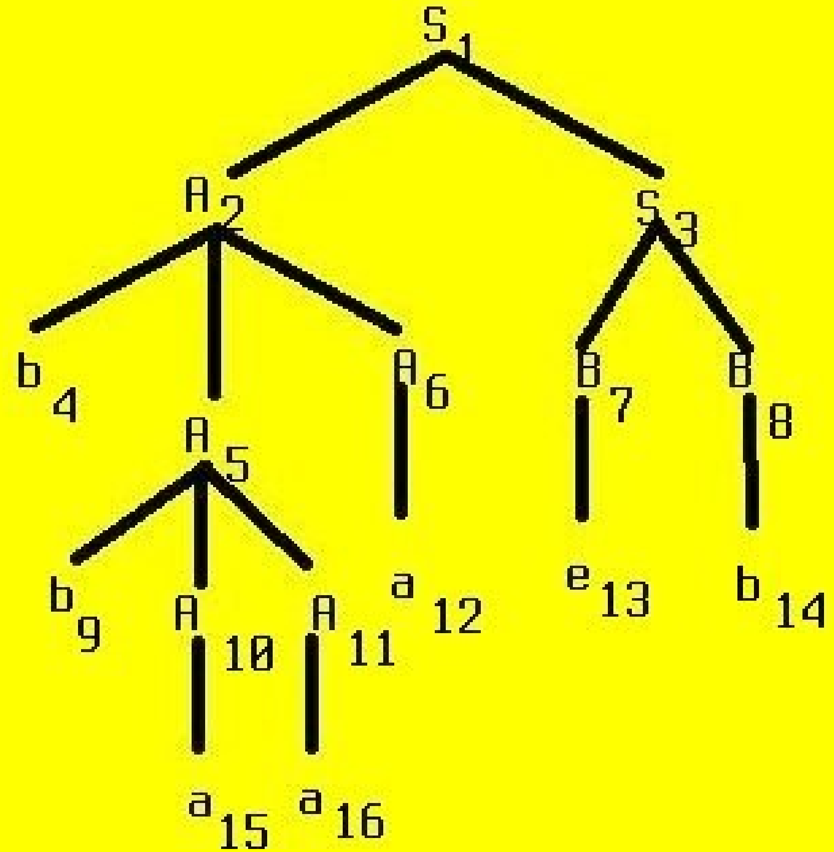
TD parse: 1 3 3 4 4 4 2 6 5

BU parse: 4 4 3 4 3 6 5 2 1

Left-corner parsing: Basics

An example:

- grammar rules: **1: S -> AS** **2: S-> BB** **3: A -> bAA** **4: A -> a** **5: B -> b** **6: B -> e**
- sentence: bbaaab



Order of nodes:

4 2 9 5 15 10 16 11 12 6 1 13 7

3 14 8

TD parse: 1 3 3 4 4 4 2 6 5

BU parse: 4 4 3 4 3 6 5 2 1

LC parse: 3 3 4 4 4 1 6 2 5

Questions?

Building a left-corner recognizer

Building a left-corner recognizer

Data: CFG <N, T, S, R>   Lexicon L

Building a left-corner recognizer

Data: CFG <N, T, S, R>   Lexicon L

Data structures: 3 stacks

Building a left-corner recognizer

Data: CFG <N, T, S, R>   Lexicon L

Data structures: 3 stacks

1) SENTENCE to be processed

Building a left-corner recognizer

Data: CFG <N, T, S, R>   Lexicon L

Data structures: 3 stacks

1) SENTENCE to be processed

2) CATEGORIES to be recognized

Building a left-corner recognizer

Data: CFG <N, T, S, R> Lexicon L

Data structures: 3 stacks

1) SENTENCE to be processed

2) CATEGORIES to be recognized

3) CONSTITUENTS we are looking for

Building a left-corner recognizer

Data: CFG <N, T, S, R>   Lexicon L

Data structures: 3 stacks

1) SENTENCE to be processed

2) CATEGORIES to be recognized

3) CONSTITUENTS we are looking for

Stack operations:

Building a left-corner recognizer

Data: CFG <N, T, S, R>   Lexicon L

Data structures: 3 stacks

1) SENTENCE to be processed

2) CATEGORIES to be recognized

3) CONSTITUENTS we are looking for

Stack operations:

pop(STACK) push(element, STACK) first(STACK)

Building a left-corner recognizer

Procedures

Building a left-corner recognizer
Procedures

REDUCE

Building a left-corner recognizer

Procedures

REDUCE

Preconditions:

Building a left-corner recognizer
Procedures

REDUCE

Preconditions:

1) There is a rule k_0 -> k_1 ... k_n in R or k_1 is part of k_0 for an arbitrary lexical category k_0

Building a left-corner recognizer
Procedures

REDUCE

Preconditions:

1) There is a rule k_0 -> k_1 … k_n in R or k_1 is part of k_0 for an arbitrary lexical category k_0

2) first(CATEGORIES) ∈ (N U T)

Building a left-corner recognizer
Procedures

REDUCE

Preconditions:

1) There is a rule k_0 -> k_1 ... k_n in R or k_1 is part of k_0 for an arbitrary lexical category k_0

2) first(CATEGORIES) ∈ (N U T)

Input:

SENTENCE with first = k_1; CATEGORIES; CONSTITUENTS

Building a left-corner recognizer
Procedures

REDUCE

Preconditions:

1) There is a rule k_0 -> k_1 ... k_n in R or k_1 is part of k_0 for an arbitrary lexical category k_0

2) first(CATEGORIES) ∈ (N U T)

Input:

SENTENCE with first = k_1; CATEGORIES; CONSTITUENTS

Output:

pop(SENTENCE); push(k_2 ... k_n t, CATEGORIES); push(k_0, CONSTITUENTS)

Building a left-corner recognizer
Procedures

REDUCE

=> delete first symbol from sentence (= left corner of rule)

=> rest of right hand side of rule is pushed onto CATEGORIES together with signal symbol for end of rule 't'

=> CONSTITUENTS keeps in mind we are looking for k_0

Building a left-corner recognizer
Procedures

MOVE

Building a left-corner recognizer
Procedures

MOVE

Preconditions:

Building a left-corner recognizer

Procedures

MOVE

Preconditions:

1) first(CATEGORIES) = t

Building a left-corner recognizer
Procedures

MOVE

Preconditions:

1) first(CATEGORIES) = t

2) first(CONSTITUENTS) = A $\in$ (N U T)

Building a left-corner recognizer
Procedures

MOVE

Preconditions:

1) first(CATEGORIES) = t

2) first(CONSTITUENTS) = A ∈ (N U T)

Input:

SENTENCE; CATEGORIES; CONSTITUENTS

Building a left-corner recognizer
Procedures

MOVE

Preconditions:

1) first(CATEGORIES) = t

2) first(CONSTITUENTS) = A Є (N U T)

Input:

SENTENCE; CATEGORIES; CONSTITUENTS

Output:

push(first(CONSTITUENTS), SENTENCE);
pop(CATEGORIES); pop(CONSTITUENTS)

Building a left-corner recognizer
Procedures

MOVE

=> right-hand-side of rule whose left-hand-side is A has been completely processed, A was recognized

=> push A onto SENTENCE

=> remove the 't' from CATEGORIES

=> remove A from CONSTITUENTS

Building a left-corner recognizer

Procedures

REMOVE

Building a left-corner recognizer

Procedures

REMOVE

Precondition:

first(SENTENCE) = first(CATEGORIES)

Building a left-corner recognizer
Procedures

REMOVE

Precondition:

first(SENTENCE) = first(CATEGORIES)

Input:

SENTENCE; CATEGORIES; CONSTITUENTS

Building a left-corner recognizer
Procedures

REMOVE

Precondition:

first(SENTENCE) = first(CATEGORIES)

Input:

SENTENCE; CATEGORIES; CONSTITUENTS

Output:

pop(SENTENCE); pop(CATEGORIES); CONSTITUENTS

Building a left-corner recognizer
Procedures

REMOVE

=> is applied iff first(SENTENCE) is a category k_i, a left corner, and this category has been recognized

Building a left-corner recognizer

The Algorithm

Building a left-corner recognizer

The Algorithm

RECOGNIZE$_{LC}$

Building a left-corner recognizer
The Algorithm

RECOGNIZE$_{LC}$

Data: CFG G = <N, T, S, R>   Lexicon L

sentence $w = w_1 \ldots w_n$, $n >= 1$

Building a left-corner recognizer
The Algorithm

RECOGNIZE$_{LC}$

Data: CFG G = <N, T, S, R>   Lexicon L
sentence w = w_1 … w_n, n >= 1

Input:

SENTENCE = [w_1 … w_n]; CATEGORIES = [S];
CONSTITUENTS = [ ]

Building a left-corner recognizer
The Algorithm

RECOGNIZE$_{LC}$

Data: CFG G = <N, T, S, R>   Lexicon L
sentence w = w_1 … w_n, n >= 1

Input:

SENTENCE = [w_1 … w_n]; CATEGORIES = [S]; CONSTITUENTS = [ ]

Output:

true / false

Building a left-corner recognizer
The Algorithm

RECOGNIZE$_{LC}$

Method:

```
if (SENTENCE == CATEGORIES ==
  CONSTITUENTS == [ ])   return true;
else
    if (there is a procedure P ∈ {REDUCE, MOVE,
      REMOVE} whose preconditions are met)
        RECOGNIZELC(P(SENTENCE, CATEGORIES,
          CONSTITUENTS));
    else return false;
```

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[der Meister su...]	[S]	[ ]	REDUCE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[der Meister su…]	[S]	[ ]	REDUCE
[Meister sucht…]	[t S]	[det]	MOVE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[der Meister su…]	[S]	[ ]	REDUCE
[Meister sucht…]	[t S]	[det]	MOVE
[det Meister su…]	[S]	[ ]	REDUCE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[der Meister su…]	[S]	[ ]	REDUCE
[Meister sucht…]	[t S]	[det]	MOVE
[det Meister su…]	[S]	[ ]	REDUCE
[Meister sucht…]	[n t S]	[NP]	REDUCE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[der Meister su…]	[S]	[ ]	REDUCE
[Meister sucht…]	[t S]	[det]	MOVE
[det Meister su…]	[S]	[ ]	REDUCE
[Meister sucht…]	[n t S]	[NP]	REDUCE
[sucht einen F…]	[t n t S]	[n NP]	MOVE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[der Meister su…]	[S]	[ ]	REDUCE
[Meister sucht…]	[t S]	[det]	MOVE
[det Meister su…]	[S]	[ ]	REDUCE
[Meister sucht…]	[n t S]	[NP]	REDUCE
[sucht einen F…]	[t n t S]	[n NP]	MOVE
[n sucht einen F…]	[n t S]	[NP]	REMOVE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[der Meister su...]	[S]	[ ]	REDUCE
[Meister sucht...]	[t S]	[det]	MOVE
[det Meister su...]	[S]	[ ]	REDUCE
[Meister sucht...]	[n t S]	[NP]	REDUCE
[sucht einen F...]	[t n t S]	[n NP]	MOVE
[n sucht einen F...]	[n t S]	[NP]	REMOVE
[sucht einen F...]	[t S]	[NP]	MOVE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[der Meister su...]	[S]	[ ]	REDUCE
[Meister sucht...]	[t S]	[det]	MOVE
[det Meister su...]	[S]	[ ]	REDUCE
[Meister sucht...]	[n t S]	[NP]	REDUCE
[sucht einen F...]	[t n t S]	[n NP]	MOVE
[n sucht einen F...]	[n t S]	[NP]	REMOVE
[sucht einen F...]	[t S]	[NP]	MOVE
[NP sucht einen...]	[S]	[ ]	REDUCE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[der Meister su…]	[S]	[ ]	REDUCE
[Meister sucht…]	[t S]	[det]	MOVE
[det Meister su…]	[S]	[ ]	REDUCE
[Meister sucht…]	[n t S]	[NP]	REDUCE
[sucht einen F…]	[t n t S]	[n NP]	MOVE
[n sucht einen F…]	[n t S]	[NP]	REMOVE
[sucht einen F…]	[t S]	[NP]	MOVE
[NP sucht einen…]	[S]	[ ]	REDUCE
[sucht einen F…]	[VP t S]	[S]	REDUCE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[der Meister su...]	[S]	[ ]	REDUCE
[Meister sucht...]	[t S]	[det]	MOVE
[det Meister su...]	[S]	[ ]	REDUCE
[Meister sucht...]	[n t S]	[NP]	REDUCE
[sucht einen F...]	[t n t S]	[n NP]	MOVE
[n sucht einen F...]	[n t S]	[NP]	REMOVE
[sucht einen F...]	[t S]	[NP]	MOVE
[NP sucht einen...]	[S]	[ ]	REDUCE
[sucht einen F...]	[VP t S]	[S]	REDUCE
[einen Fehler]	[t VP t S]	[v S]	MOVE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[der Meister su...]	[S]	[ ]	REDUCE
[Meister sucht...]	[t S]	[det]	MOVE
[det Meister su...]	[S]	[ ]	REDUCE
[Meister sucht...]	[n t S]	[NP]	REDUCE
[sucht einen F...]	[t n t S]	[n NP]	MOVE
[n sucht einen F...]	[n t S]	[NP]	REMOVE
[sucht einen F...]	[t S]	[NP]	MOVE
[NP sucht einen...]	[S]	[ ]	REDUCE
[sucht einen F...]	[VP t S]	[S]	REDUCE
[einen Fehler]	[t VP t S]	[v S]	MOVE
[v einen Fehler]	[VP t S]	[S]	REDUCE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[der Meister su...]	[S]	[ ]	REDUCE
[Meister sucht...]	[t S]	[det]	MOVE
[det Meister su...]	[S]	[ ]	REDUCE
[Meister sucht...]	[n t S]	[NP]	REDUCE
[sucht einen F...]	[t n t S]	[n NP]	MOVE
[n sucht einen F...]	[n t S]	[NP]	REMOVE
[sucht einen F...]	[t S]	[NP]	MOVE
[NP sucht einen...]	[S]	[ ]	REDUCE
[sucht einen F...]	[VP t S]	[S]	REDUCE
[einen Fehler]	[t VP t S]	[v S]	MOVE
[v einen Fehler]	[VP t S]	[S]	REDUCE
[einen Fehler]	[NP t VP t S]	[VP S]	REDUCE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[der Meister su...]	[S]	[ ]	REDUCE
[Meister sucht...]	[t S]	[det]	MOVE
[det Meister su...]	[S]	[ ]	REDUCE
[Meister sucht...]	[n t S]	[NP]	REDUCE
[sucht einen F...]	[t n t S]	[n NP]	MOVE
[n sucht einen F...]	[n t S]	[NP]	REMOVE
[sucht einen F...]	[t S]	[NP]	MOVE
[NP sucht einen...]	[S]	[ ]	REDUCE
[sucht einen F...]	[VP t S]	[S]	REDUCE
[einen Fehler]	[t VP t S]	[v S]	MOVE
[v einen Fehler]	[VP t S]	[S]	REDUCE
[einen Fehler]	[NP t VP t S]	[VP S]	REDUCE
[Fehler]	[t NP t VP t S]	[det VP S]	MOVE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[der Meister su...]	[S]	[ ]	REDUCE
[Meister sucht...]	[t S]	[det]	MOVE
[det Meister su...]	[S]	[ ]	REDUCE
[Meister sucht...]	[n t S]	[NP]	REDUCE
[sucht einen F...]	[t n t S]	[n NP]	MOVE
[n sucht einen F...]	[n t S]	[NP]	REMOVE
[sucht einen F...]	[t S]	[NP]	MOVE
[NP sucht einen...]	[S]	[ ]	REDUCE
[sucht einen F...]	[VP t S]	[S]	REDUCE
[einen Fehler]	[t VP t S]	[v S]	MOVE
[v einen Fehler]	[VP t S]	[S]	REDUCE
[einen Fehler]	[NP t VP t S]	[VP S]	REDUCE
[Fehler]	[t NP t VP t S]	[det VP S]	MOVE
[det Fehler]	[NP t VP t S]	[VP S]	REDUCE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[der Meister su...]	[S]	[ ]	REDUCE
[Meister sucht...]	[t S]	[det]	MOVE
[det Meister su...]	[S]	[ ]	REDUCE
[Meister sucht...]	[n t S]	[NP]	REDUCE
[sucht einen F...]	[t n t S]	[n NP]	MOVE
[n sucht einen F...]	[n t S]	[NP]	REMOVE
[sucht einen F...]	[t S]	[NP]	MOVE
[NP sucht einen...]	[S]	[ ]	REDUCE
[sucht einen F...]	[VP t S]	[S]	REDUCE
[einen Fehler]	[t VP t S]	[v S]	MOVE
[v einen Fehler]	[VP t S]	[S]	REDUCE
[einen Fehler]	[NP t VP t S]	[VP S]	REDUCE
[Fehler]	[t NP t VP t S]	[det VP S]	MOVE
[det Fehler]	[NP t VP t S]	[VP S]	REDUCE
[Fehler]	[n t NP t VP t S]	[NP VP S]	REDUCE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[der Meister su...]	[S]	[ ]	REDUCE
[Meister sucht...]	[t S]	[det]	MOVE
[det Meister su...]	[S]	[ ]	REDUCE
[Meister sucht...]	[n t S]	[NP]	REDUCE
[sucht einen F...]	[t n t S]	[n NP]	MOVE
[n sucht einen F...]	[n t S]	[NP]	REMOVE
[sucht einen F...]	[t S]	[NP]	MOVE
[NP sucht einen...]	[S]	[ ]	REDUCE
[sucht einen F...]	[VP t S]	[S]	REDUCE
[einen Fehler]	[t VP t S]	[v S]	MOVE
[v einen Fehler]	[VP t S]	[S]	REDUCE
[einen Fehler]	[NP t VP t S]	[VP S]	REDUCE
[Fehler]	[t NP t VP t S]	[det VP S]	MOVE
[det Fehler]	[NP t VP t S]	[VP S]	REDUCE
[Fehler]	[n t NP t VP t S]	[NP VP S]	REDUCE
[ ]	[t n t NP t VP t S]	[n NP VP S]	MOVE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[ ]	[t n t NP t VP t S]	[n NP VP S]	MOVE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[ ]	[t n t NP t VP t S]	[n NP VP S]	MOVE
[n]	[n t NP t VP t S]	[NP VP S]	REMOVE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[ ]	[t n t NP t VP t S]	[n NP VP S]	MOVE
[n]	[n t NP t VP t S]	[NP VP S]	REMOVE
[ ]	[t NP t VP t S]	[NP VP S]	MOVE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[ ]	[t n t NP t VP t S]	[n NP VP S]	MOVE
[n]	[n t NP t VP t S]	[NP VP S]	REMOVE
[ ]	[t NP t VP t S]	[NP VP S]	MOVE
[NP]	[NP t VP t S]	[VP S]	REMOVE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[ ]	[t n t NP t VP t S]	[n NP VP S]	MOVE
[n]	[n t NP t VP t S]	[NP VP S]	REMOVE
[ ]	[t NP t VP t S]	[NP VP S]	MOVE
[NP]	[NP t VP t S]	[VP S]	REMOVE
[ ]	[t VP t S]	[VP S]	MOVE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[ ]	[t n t NP t VP t S]	[n NP VP S]	MOVE
[n]	[n t NP t VP t S]	[NP VP S]	REMOVE
[ ]	[t NP t VP t S]	[NP VP S]	MOVE
[NP]	[NP t VP t S]	[VP S]	REMOVE
[ ]	[t VP t S]	[VP S]	MOVE
[VP]	[VP t S]	[S]	REMOVE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[ ]	[t n t NP t VP t S]	[n NP VP S]	MOVE
[n]	[n t NP t VP t S]	[NP VP S]	REMOVE
[ ]	[t NP t VP t S]	[NP VP S]	MOVE
[NP]	[NP t VP t S]	[VP S]	REMOVE
[ ]	[t VP t S]	[VP S]	MOVE
[VP]	[VP t S]	[S]	REMOVE
[ ]	[t S]	[S]	MOVE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[ ]	[t n t NP t VP t S]	[n NP VP S]	MOVE
[n]	[n t NP t VP t S]	[NP VP S]	REMOVE
[ ]	[t NP t VP t S]	[NP VP S]	MOVE
[NP]	[NP t VP t S]	[VP S]	REMOVE
[ ]	[t VP t S]	[VP S]	MOVE
[VP]	[VP t S]	[S]	REMOVE
[ ]	[t S]	[S]	MOVE
[S]	[S]	[ ]	REMOVE

Building a left-corner recognizer
Example

Der Meister sucht einen Fehler

SENTENCE	CATEGORIES	CONSTITUENTS	procedure
[ ]	[t n t NP t VP t S]	[n NP VP S]	MOVE
[n]	[n t NP t VP t S]	[NP VP S]	REMOVE
[ ]	[t NP t VP t S]	[NP VP S]	MOVE
[NP]	[NP t VP t S]	[VP S]	REMOVE
[ ]	[t VP t S]	[VP S]	MOVE
[VP]	[VP t S]	[S]	REMOVE
[ ]	[t S]	[S]	MOVE
[S]	[S]	[ ]	REMOVE
[ ]	[ ]	[ ]	true

Building a left-corner recognizer

Why is $\text{RECOGNIZE}_{\text{LC}}$ non-deterministic?

Building a left-corner recognizer

Why is RECOGNIZE_{LC} non-deterministic?

- there may be several rules whose left corner is equal to first(SENTENCE)

Building a left-corner recognizer

Why is RECOGNIZE$_{LC}$ non-deterministic?

- there may be several rules whose left corner is equal to first(SENTENCE)
- there may be configurations where you could either REDUCE or REMOVE:

Building a left-corner recognizer

Why is RECOGNIZE$_{LC}$ non-deterministic?

- there may be several rules whose left corner is equal to first(SENTENCE)
- there may be configurations where you could either REDUCE or REMOVE:
    - a newly created structure can be used to complete the structure we are working at => REMOVE

Building a left-corner recognizer

Why is RECOGNIZE$_{LC}$ non-deterministic?

- there may be several rules whose left corner is equal to first(SENTENCE)
- there may be configurations where you could either REDUCE or REMOVE:
    - a newly created structure can be used to complete the structure we are working at => REMOVE
    - or it could constitute a new structure of its own => REDUCE

Building a left-corner recognizer

=> use breadth-first or depth-first search
to check all possible configurations

Building a left-corner recognizer
breadth-first search

RECOGNIZE$_{LC/BF}$

Data: CFG G = <N, T, S, R>   Lexicon L
sentence w = w_1 … w_n, n >= 1

Input:

SENTENCE = [w_1 … w_n]; CATEGORIES = [S]; CONSTITUENTS = [ ]

Output: true / false

Structures: CONFIGS – set of configurations, null at the beginning

Building a left-corner recognizer
breadth-first search

RECOGNIZE$_{LC/BF}$

Method:

if (SENTENCE == CATEGORIES == CONSTITUENTS == [ ])   return true;

else CONFIGS = set of all configurations derivable from the actual configuration using REMOVE, REDUCE or MOVE

if (CONFIGS == null) return false;

else for every configuration C ∈ CONFIGS:

RECOGNIZE$_{LC/BF}$(SENTENCE$_C$, CATEGORIES$_C$, CONSTITUENTS$_C$);

Questions?

A left-corner parsing algorithm

A left-corner parsing algorithm

- introduce another stack: STRUCTURE

A left-corner parsing algorithm

- introduce another stack: STRUCTURE
- empty at the beginning; filled along the way

A left-corner parsing algorithm

- introduce another stack: STRUCTURE
- empty at the beginning; filled along the way
- return value: the structure stored in stack STRUCTURE

A left-corner parsing algorithm
Modifying the procedures

$MOVE_{LC/BF}$

Preconditions:

1) first(CATEGORIES) = t

2) first(CONSTITUENTS) = A ∈ (N U T)

Input:

SENTENCE; CATEGORIES; CONSTITUENTS; STRUCTURE

Output:

push(first(CONSTITUENTS), SENTENCE); pop(CATEGORIES); pop(CONSTITUENTS); STRUCTURE

A left-corner parsing algorithm
Modifying the procedures

$\text{MOVE}_{\text{LC/BF}}$

=> just insert another parameter

for the structure stack

A left-corner parsing algorithm
Modifying the procedures

$\text{REDUCE}_{LC/BF}$

Preconditions:

1) There is a rule k_0 -> k_1 … k_n in R or k_1 is part of k_0 for an arbitrary lexical category k_0

2) first(CATEGORIES) ∈ (N U T)

Input:

SENTENCE with first = k_1; CATEGORIES; CONSTITUENTS; STRUCTURE

Output: pop(SENTENCE); push(k_2 … k_n t, CATEGORIES); push(k_0, CONSTITUENTS); structure1(k_0, k_1, STRUCTURE)

A left-corner parsing algorithm
Modifying the procedures

REDUCE$_{LC/BF}$ – new subprocedure structure1

A left-corner parsing algorithm
Modifying the procedures

$REDUCE_{LC/BF}$ – new subprocedure structure1

Input: STRUCTURE, symbols k_0, k_1 ∈ (N U T)

A left-corner parsing algorithm

Modifying the procedures

$REDUCE_{LC/BF}$ – new subprocedure structure1

Input: STRUCTURE, symbols k_0, k_1 ∈ (N U T)

Output: modified STRUCTURE'

A left-corner parsing algorithm
Modifying the procedures

$REDUCE_{LC/BF}$ – new subprocedure structure1

Input: STRUCTURE, symbols k_0, k_1 ∈ (N U T)

Output: modified STRUCTURE'

Method:

if (STRUCTURE == [ ] U
first(STRUCTURE) == k'α with k' != k_1))

return push((k_0 k_1), STRUCTURE)

else return(push((k_0 first(STRUCTURE)),
pop(STRUCTURE)))

A left-corner parsing algorithm

Modifying the procedures

$REDUCE_{LC/BF}$

=> add structure1(k_0,k_1,STRUCTURE) to output

structure1:

=> if there is already a structure dominated by k_1, integrate the new symbols, else build up a new structure description

A left-corner parsing algorithm
Modifying the procedures

REMOVE$_{LC/BF}$

Precondition:

first(SENTENCE) = first(CATEGORIES)

Input:

SENTENCE; CATEGORIES; CONSTITUENTS; STRUCTURE

Output:

pop(SENTENCE); pop(CATEGORIES); CONSTITUENTS; structure2(CONSTITUENTS, STRUCTURE)

A left-corner parsing algorithm

Modifying the procedures

REMOVE$_{LC/BF}$ – subprocedure structure2

A left-corner parsing algorithm
Modifying the procedures

REMOVE$_{LC/BF}$ – subprocedure structure2

Input: CONSTITUENTS; STRUCTURE

A left-corner parsing algorithm
Modifying the procedures

$REMOVE_{LC/BF}$ – subprocedure structure2

Input: CONSTITUENTS; STRUCTURE

Output: modified STRUCTURE'

A left-corner parsing algorithm
Modifying the procedures

REMOVE$_{LC/BF}$ – subprocedure structure2

Input: CONSTITUENTS; STRUCTURE

Output: modified STRUCTURE'

Method:

```
if(CONSTITUENTS == [ ])
  return STRUCTURE

else
  return(push((second(STRUCTURE) +
  first(STRUCTURE)), pop(pop(STRUCTURE))))
```

A left-corner parsing algorithm
Modifying the procedures

REMOVE$_{LC/BF}$ with subprocedure structure2

=> if CONSTITUENTS is not empty, associate the last two partial structure descriptions on STRUCTURE

A left-corner parsing algorithm
Example

Eva sah Adam am Morgen

SENTENCE	CATEGORIES	CONSTITUENTS	STRUCTURE
[Eva sah Adam...]	[S]	[ ]	[ ]
[sah Adam...]	[t S]	[n]	[(n1)]
[n sah Adam...]	[S]	[ ]	[(n1)]
[sah Adam...]	[t S]	[NP]	[(NP(n1))]
[NP sah Adam...]	[S]	[ ]	[(NP(n1))]
[sah Adam...]	[VP t S]	[S]	[S(NP(n1))]
[Adam am Morgen]	[t VP t S]	[v S]	[(v2)(S(NP(n1)))]
[v Adam am Morgen]	[VP t S]	[S]	[(v2)(S(NP(n1)))]
[Adam am Morgen]	[NP PP t VP t S]	[VP S]	[(VP(v2))(S(NP...]
[am Morgen]	[t NP PP t VP t S]	[n VP S]	[(n3)(VP(v2))(S...]
...	...	...	...

(S (NP(n1)) (VP (v2) (NP(n3)) (PP (p4) (NP(n5)))))

Questions?

Left-corner parsing with look-ahead

Left-corner parsing with look-ahead

- become more efficient...

Left-corner parsing with look-ahead

- become more efficient…
- … by reducing number of rules that can be used to generate next derivation

Left-corner parsing with look-ahead

- become more efficient…
- … by reducing number of rules that can be used to generate next derivation
- for every nonterminal n, calculate the set of all symbols which are left corners of constituents reachable from n

Left-corner parsing with look-ahead

- become more efficient…
- … by reducing number of rules that can be used to generate next derivation
- for every nonterminal n, calculate the set of all symbols which are left corners of constituents reachable from n

=> relation “LINK“

Left-corner parsing with look-ahead

LINK(G)

Left-corner parsing with look-ahead

LINK(G)

set of all ordered pairs <X, Y> with $X \in N$ and $Y \in (N \text{ U } T)$ which fulfill either of these conditions:

Left-corner parsing with look-ahead

LINK(G)

set of all ordered pairs <X, Y> with X ∈ N and Y ∈ (N U T) which fulfill either of these conditions:

1) X = Y (reflexivity)

Left-corner parsing with look-ahead

LINK(G)

set of all ordered pairs <X, Y> with X ∈ N and Y ∈ (N U T) which fulfill either of these conditions:

1) X = Y (reflexivity)

2) there is a rule X -> Yα ∈ R

Left-corner parsing with look-ahead

LINK(G)

set of all ordered pairs <X, Y> with X Є N and Y Є (N U T) which fulfill either of these conditions:

1) X = Y (reflexivity)

2) there is a rule X -> Yα Є R

3) <X, X'> Є LINK(G) and <X',Y> Є LINK(G) for an arbitrary X' Є N (transitivity)

Left-corner parsing with look-ahead

LINK(G)

set of all ordered pairs <X, Y> with X Є N and Y Є (N U T) which fulfill either of these conditions:

1) X = Y (reflexivity)

2) there is a rule X -> Yα Є R

3) <X, X'> Є LINK(G) and <X',Y> Є LINK(G) for an arbitrary X' Є N (transitivity)

=> should be calculated before parsing

Left-corner parsing with look-ahead
Example

Grammar G with rules:

S -> X2 X3 X4 **X2 -> e f**

X3 -> X1 **X1 -> g**

X4 -> h

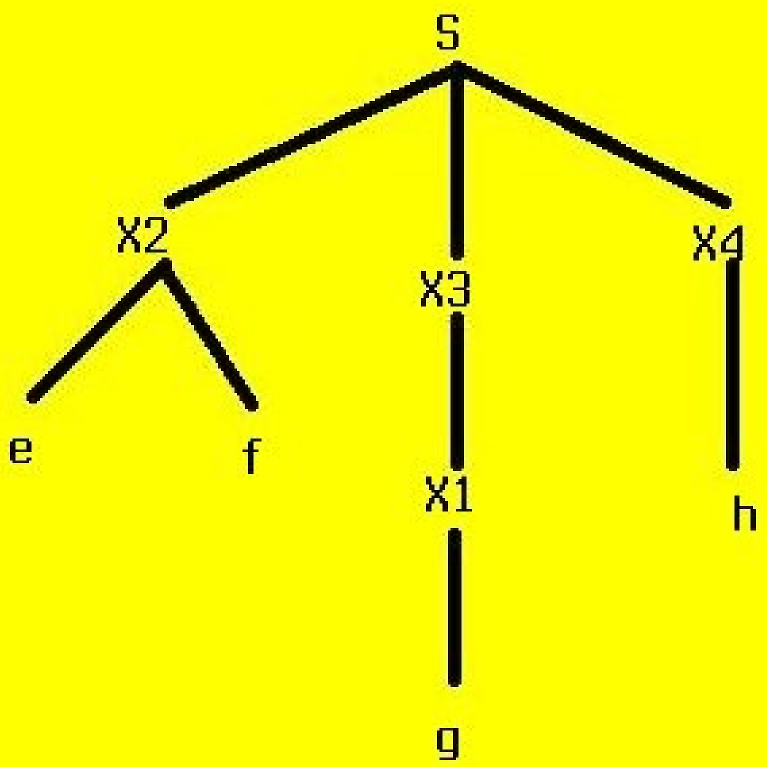
Left-corner parsing with look-ahead
Example

Grammar G with rules:

S -> X2 X3 X4 **X2 -> e f**

X3 -> X1 **X1 -> g**

X4 -> h



Left-corner parsing with look-ahead
Example

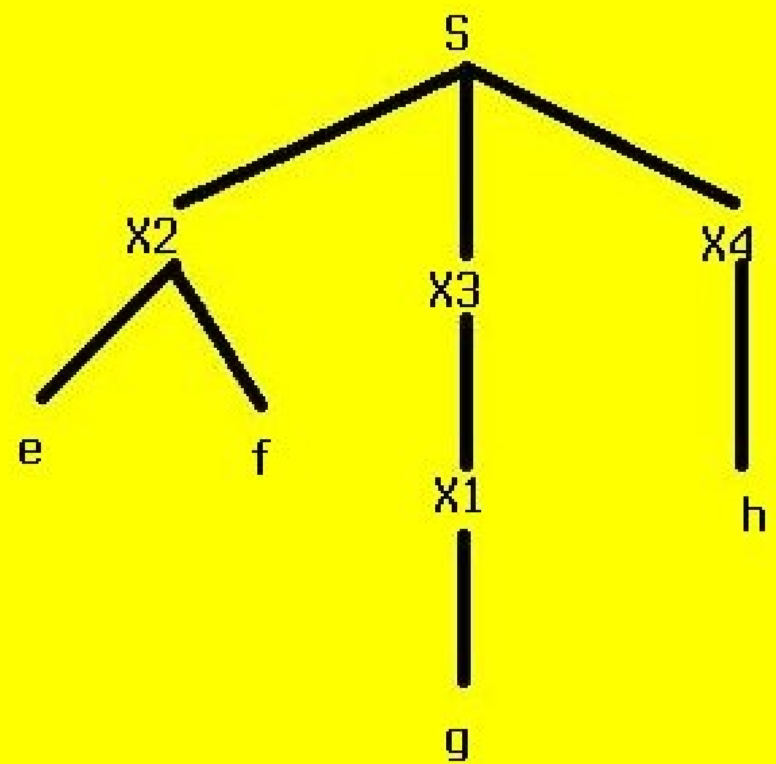
Grammar G with rules:

S -> X2 X3 X4   **X2 -> e f**

X3 -> X1   **X1 -> g**

X4 -> h

LINK(G) = {<S,S>, <X1, X1>,
<X2, X2>, <X3, X3>, <X4, X4>,
<S, X2>, <S, e>, <X2, e>,
<X1, g>, <X3, X1>, <X3, g>,
<X4, h>



Left-corner parsing with look-ahead
Example

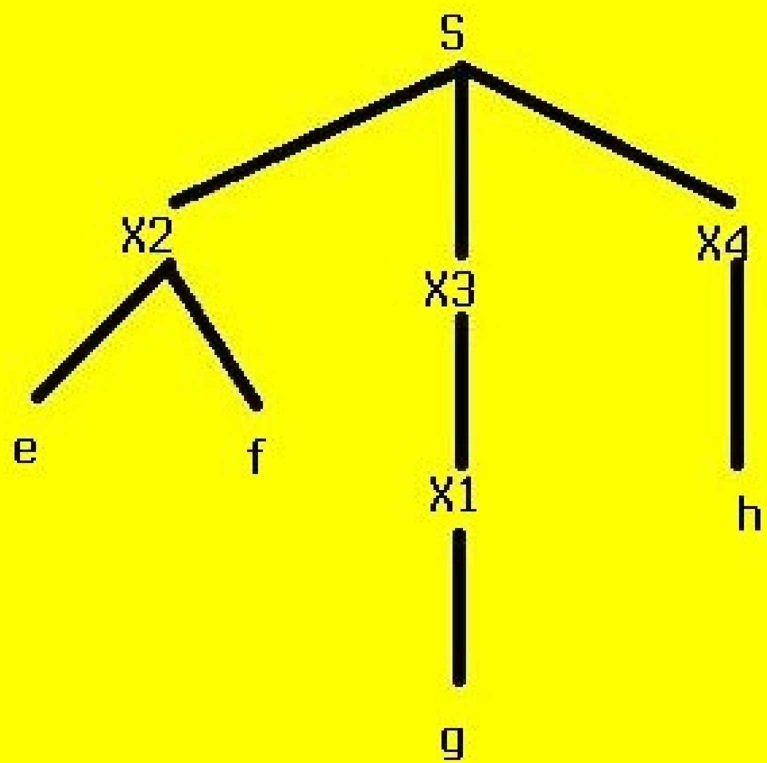
Grammar G with rules:

S -> X2 X3 X4   X2 -> e f

X3 -> X1   X1 -> g

X4 -> h

LINK(G) = {<S,S>, <X1, X1>,
<X2, X2>, <X3, X3>, <X4, X4>,
<S, X2>, <S, e>, <X2, e>,
<X1, g>, <X3, X1>, <X3, g>,
<X4, h>



=> strings like 'fghe' or 'hefg' needn't be parsed at all!

Left-corner parsing with look-ahead
Modifying the procedures

only necessary change: $REDUCE_{LC/LA}$

Preconditions:

1) There is a rule k_0 -> k_1 ... k_n in R or k_1 is part of k_0 for an arbitrary lexical category k_0

2) first(CATEGORIES) $\in$ (N U T)

3) <first(CATEGORIES), k_0> $\in$ LINK(G)

Input: SENTENCE with first = k_1; CATEGORIES; CONSTITUENTS; STRUCTURE

Output: pop(SENTENCE); push(k_2 ... k_n t, CATEGORIES); push(k_0, CONSTITUENTS); structure1(STRUCTURE)

Questions?

Comparison to other approaches

Comparison to other approaches

Drawback of top-down:

Comparison to other approaches

Drawback of top-down:

- ignores what the actual input string looks like most of the time

Comparison to other approaches

Drawback of top-down:

- ignores what the actual input string looks like most of the time

Drawback of bottom-up:

Comparison to other approaches

Drawback of top-down:

- ignores what the actual input string looks like most of the time

Drawback of bottom-up:

- we don't know what we're trying to build at the moment

Comparison to other approaches

Drawback of top-down:

- ignores what the actual input string looks like most of the time

Drawback of bottom-up:

- we don't know what we're trying to build at the moment

=> Left-corner can handle these... examples follow!

Comparison to other approaches

Example TD vs LC

Comparison to other approaches
Example TD vs LC

Grammar: S -> NP VP NP -> det N NP -> PN VP -> IV

det -> the N -> robber PN -> Vincent IV -> died

Comparison to other approaches
Example TD vs LC

Grammar: S -> NP VP   NP -> det N NP -> PN VP -> IV

det -> the   N -> robber PN -> Vincent IV -> died

Input sentence: Vincent died.

Comparison to other approaches
Example TD vs LC

Grammar: S -> NP VP   NP -> det N   NP -> PN   VP -> IV

det -> the   N -> robber   PN -> Vincent   IV -> died

Input sentence: Vincent died.

Top-down:

Comparison to other approaches
Example TD vs LC

Grammar: S -> NP VP   NP -> det N NP -> PN VP -> IV

det -> the   N -> robber PN -> Vincent   IV -> died

Input sentence: Vincent died.

Top-down: S -> NP VP

Comparison to other approaches
Example TD vs LC

Grammar: S -> NP VP   NP -> det N   NP -> PN   VP -> IV

det -> the   N -> robber   PN -> Vincent   IV -> died

Input sentence: Vincent died.

Top-down: S -> NP VP -> det N VP

Comparison to other approaches
Example TD vs LC

Grammar: S -> NP VP   NP -> det N   NP -> PN   VP -> IV

det -> the   N -> robber   PN -> Vincent   IV -> died

Input sentence: Vincent died.

Top-down: S -> NP VP -> det N VP -> DEAD END!

Vincent isn't det, det cannot be expanded

=> need to backtrack ;-(

Comparison to other approaches
Example TD vs LC

Grammar: S -> NP VP   NP -> det N NP -> PN VP -> IV

det -> the   N -> robber PN -> Vincent IV -> died

Input sentence: Vincent died.

Top-down: S -> NP VP -> det N VP -> DEAD END!

Vincent isn't det, det cannot be expanded

=> need to backtrack ;-(

Left-corner:

Comparison to other approaches
Example TD vs LC

Grammar: S -> NP VP NP -> det N NP -> PN VP -> IV

det -> the N -> robber PN -> Vincent IV -> died

Input sentence: Vincent died.

Top-down: S -> NP VP -> det N VP -> DEAD END!

Vincent isn't det, det cannot be expanded

=> need to backtrack ;-(

Left-corner: predict S (TD)

Comparison to other approaches
Example TD vs LC

Grammar: S -> NP VP   NP -> det N   NP -> PN   VP -> IV

det -> the   N -> robber   PN -> Vincent   IV -> died

Input sentence: Vincent died.

Top-down: S -> NP VP -> det N VP -> DEAD END!

Vincent isn't det, det cannot be expanded

=> need to backtrack ;-(

Left-corner: predict S (TD) -> recognize PN (BU)

Comparison to other approaches
Example TD vs LC

Grammar: S -> NP VP NP -> det N NP -> PN VP -> IV

det -> the N -> robber PN -> Vincent IV -> died

Input sentence: Vincent died.

Top-down: S -> NP VP -> det N VP -> DEAD END!

Vincent isn't det, det cannot be expanded

=> need to backtrack ;-(

Left-corner: predict S (TD) -> recognize PN (BU) -> select rule 'NP -> PN'

Comparison to other approaches
Example TD vs LC

Grammar: S -> NP VP NP -> det N NP -> PN VP -> IV

det -> the N -> robber PN -> Vincent IV -> died

Input sentence: Vincent died.

Top-down: S -> NP VP -> det N VP -> DEAD END!
Vincent isn't det, det cannot be expanded
=> need to backtrack ;-(

Left-corner: predict S (TD) -> recognize PN (BU) -> select rule 'NP -> PN'
-> select rule 'S -> NP VP'

Comparison to other approaches
Example TD vs LC

Grammar: S -> NP VP NP -> det N NP -> PN VP -> IV

det -> the N -> robber PN -> Vincent IV -> died

Input sentence: Vincent died.

Top-down: S -> NP VP -> det N VP -> DEAD END!

Vincent isn't det, det cannot be expanded

=> need to backtrack ;-(

Left-corner: predict S (TD) -> recognize PN (BU) -> select rule 'NP -> PN'

-> select rule 'S -> NP VP' -> MATCH!

Comparison to other approaches
Example TD vs LC

Grammar: S -> NP VP NP -> det N NP -> PN VP -> IV

det -> the N -> robber PN -> Vincent IV -> died

Input sentence: Vincent died.

Top-down: S -> NP VP -> det N VP -> DEAD END!
Vincent isn't det, det cannot be expanded
=> need to backtrack ;-(

Left-corner: predict S (TD) -> recognize PN (BU) -> select rule 'NP -> PN'
-> select rule 'S -> NP VP' -> MATCH!
-> input: died – predict VP (TD)

Comparison to other approaches
Example TD vs LC

Grammar: S -> NP VP NP -> det N NP -> PN VP -> IV

det -> the N -> robber PN -> Vincent IV -> died

Input sentence: Vincent died.

Top-down: S -> NP VP -> det N VP -> DEAD END!
Vincent isn't det, det cannot be expanded
=> need to backtrack ;-(

Left-corner: predict S (TD) -> recognize PN (BU) -> select rule 'NP -> PN'
-> select rule 'S -> NP VP' -> MATCH!
-> input: died – predict VP (TD) -> recognize IV (BU)

Comparison to other approaches
Example TD vs LC

Grammar: S -> NP VP NP -> det N NP -> PN VP -> IV

det -> the N -> robber PN -> Vincent IV -> died

Input sentence: Vincent died.

Top-down: S -> NP VP -> det N VP -> DEAD END!

Vincent isn't det, det cannot be expanded

=> need to backtrack ;-(

Left-corner: predict S (TD) -> recognize PN (BU) -> select rule 'NP -> PN'

-> select rule 'S -> NP VP' -> MATCH!

-> input: died – predict VP (TD) -> recognize IV (BU)

-> select rule 'VP -> IV'

Comparison to other approaches
Example TD vs LC

Grammar: S -> NP VP NP -> det N NP -> PN VP -> IV

det -> the N -> robber PN -> Vincent IV -> died

Input sentence: Vincent died.

Top-down: S -> NP VP -> det N VP -> DEAD END!

Vincent isn't det, det cannot be expanded

=> need to backtrack ;-(

Left-corner: predict S (TD) -> recognize PN (BU) -> select rule 'NP -> PN'

-> select rule 'S -> NP VP' -> MATCH!

-> input: died – predict VP (TD) -> recognize IV (BU)

-> select rule 'VP -> IV' -> MATCH!

Comparison to other approaches
Example TD vs LC

Grammar: S -> NP VP NP -> det N NP -> PN VP -> IV

det -> the N -> robber PN -> Vincent IV -> died

Input sentence: Vincent died.

Top-down: S -> NP VP -> det N VP -> DEAD END!

Vincent isn't det, det cannot be expanded

=> need to backtrack ;-(

Left-corner: predict S (TD) -> recognize PN (BU) -> select rule 'NP -> PN'

-> select rule 'S -> NP VP' -> MATCH!

-> input: died – predict VP (TD) -> recognize IV (BU)

-> select rule 'VP -> IV' -> MATCH! => successful parse

Comparison to other approaches

Example BU vs LC

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP   NP -> det N   NP -> PN   VP -> IV
VP -> TV NP   TV -> plant   IV -> died   det -> the
N -> plant

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP   NP -> det N   NP -> PN   VP -> IV
VP -> TV NP   TV -> plant   IV -> died   det -> the
N -> plant

Input sentence: the plant died

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP NP -> det N NP -> PN VP -> IV
VP -> TV NP TV -> plant IV -> died det -> the
N -> plant

Input sentence: the plant died

Bottom-up:

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP   NP -> det N   NP -> PN   VP -> IV
VP -> TV NP   TV -> plant   IV -> died   det -> the
N -> plant

Input sentence: the plant died

Bottom-up: the plant died

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP   NP -> det N   NP -> PN   VP -> IV
VP -> TV NP   TV -> plant   IV -> died   det -> the
N -> plant

Input sentence: the plant died

Bottom-up: the plant died -> det plant died

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP   NP -> det N   NP -> PN   VP -> IV
VP -> TV NP   TV -> plant   IV -> died   det -> the
N -> plant

Input sentence: the plant died

Bottom-up: the plant died -> det plant died -> det TV died

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP   NP -> det N   NP -> PN   VP -> IV
VP -> TV NP   TV -> plant   IV -> died   det -> the
N -> plant

Input sentence: the plant died

Bottom-up: the plant died -> det plant died -> det TV died -> det TV IV

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP   NP -> det N   NP -> PN   VP -> IV
VP -> TV NP   TV -> plant   IV -> died   det -> the
N -> plant

Input sentence: the plant died

Bottom-up: the plant died -> det plant died -> det TV died -> det TV IV ->
det TV VP

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP NP -> det N NP -> PN VP -> IV
VP -> TV NP TV -> plant IV -> died det -> the
N -> plant

Input sentence: the plant died

Bottom-up: the plant died -> det plant died -> det TV died -> det TV IV ->
det TV VP -> DEAD END!

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP   NP -> det N   NP -> PN   VP -> IV
VP -> TV NP   TV -> plant   IV -> died   det -> the
N -> plant

Input sentence: the plant died

Bottom-up: the plant died -> det plant died -> det TV died -> det TV IV ->
det TV VP -> DEAD END! => need to backtrack ;-(

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP NP -> det N NP -> PN VP -> IV
VP -> TV NP TV -> plant IV -> died det -> the
N -> plant

Input sentence: the plant died

Bottom-up: the plant died -> det plant died -> det TV died -> det TV IV ->
det TV VP -> DEAD END! => need to backtrack ;-(

Left-corner:

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP NP -> det N NP -> PN VP -> IV
VP -> TV NP TV -> plant IV -> died det -> the
N -> plant

Input sentence: the plant died

Bottom-up: the plant died -> det plant died -> det TV died -> det TV IV ->
det TV VP -> DEAD END! => need to backtrack ;-(

Left-corner: predict S (TD)

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP NP -> det N NP -> PN VP -> IV
VP -> TV NP TV -> plant IV -> died det -> the
N -> plant

Input sentence: the plant died

Bottom-up: the plant died -> det plant died -> det TV died -> det TV IV -> det TV VP -> DEAD END! => need to backtrack ;-(

Left-corner: predict S (TD) -> recognize det (BU)

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP   NP -> det N   NP -> PN   VP -> IV
VP -> TV NP   TV -> plant   IV -> died   det -> the
N -> plant

Input sentence: the plant died

Bottom-up: the plant died -> det plant died -> det TV died -> det TV IV ->
det TV VP -> DEAD END! => need to backtrack ;-(

Left-corner: predict S (TD) -> recognize det (BU)
-> select rule 'NP -> det N'

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP   NP -> det N   NP -> PN   VP -> IV
VP -> TV NP   TV -> plant   IV -> died   det -> the
N -> plant

Input sentence: the plant died

Bottom-up: the plant died -> det plant died -> det TV died -> det TV IV ->
det TV VP -> DEAD END! => need to backtrack ;-(

Left-corner: predict S (TD) -> recognize det (BU)
-> select rule 'NP -> det N' -> recognize N (BU)

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP   NP -> det N   NP -> PN   VP -> IV
VP -> TV NP   TV -> plant   IV -> died   det -> the
N -> plant

Input sentence: the plant died

Bottom-up: the plant died -> det plant died -> det TV died -> det TV IV ->
det TV VP -> DEAD END! => need to backtrack ;-(

Left-corner: predict S (TD) -> recognize det (BU)
-> select rule 'NP -> det N' -> recognize N (BU) -> MATCH!

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP NP -> det N NP -> PN VP -> IV
VP -> TV NP TV -> plant IV -> died det -> the
N -> plant

Input sentence: the plant died

Bottom-up: the plant died -> det plant died -> det TV died -> det TV IV ->
det TV VP -> DEAD END! => need to backtrack ;-(

Left-corner: predict S (TD) -> recognize det (BU)
-> select rule 'NP -> det N' -> recognize N (BU) -> MATCH!
-> select rule 'S -> NP VP'

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP   NP -> det N   NP -> PN   VP -> IV
VP -> TV NP   TV -> plant   IV -> died   det -> the
N -> plant

Input sentence: the plant died

Bottom-up: the plant died -> det plant died -> det TV died -> det TV IV -> det TV VP -> DEAD END! => need to backtrack ;-(

Left-corner: predict S (TD) -> recognize det (BU)
-> select rule 'NP -> det N' -> recognize N (BU) -> MATCH!
-> select rule 'S -> NP VP' -> input: died – predict VP (TD)

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP   NP -> det N   NP -> PN   VP -> IV
VP -> TV NP   TV -> plant   IV -> died   det -> the
N -> plant

Input sentence: the plant died

Bottom-up: the plant died -> det plant died -> det TV died -> det TV IV -> det TV VP -> DEAD END! => need to backtrack ;-(

Left-corner: predict S (TD) -> recognize det (BU)
-> select rule 'NP -> det N' -> recognize N (BU) -> MATCH!
-> select rule 'S -> NP VP' -> input: died – predict VP (TD) -> recognize IV (BU)

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP   NP -> det N   NP -> PN   VP -> IV
VP -> TV NP   TV -> plant   IV -> died   det -> the
N -> plant

Input sentence: the plant died

Bottom-up: the plant died -> det plant died -> det TV died -> det TV IV -> det TV VP -> DEAD END! => need to backtrack ;-(

Left-corner: predict S (TD) -> recognize det (BU)
-> select rule 'NP -> det N' -> recognize N (BU) -> MATCH!
-> select rule 'S -> NP VP' -> input: died – predict VP (TD) ->
recognize IV (BU) -> select rule 'VP -> IV'

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP   NP -> det N   NP -> PN   VP -> IV
VP -> TV NP   TV -> plant   IV -> died   det -> the
N -> plant

Input sentence: the plant died

Bottom-up: the plant died -> det plant died -> det TV died -> det TV IV -> det TV VP -> DEAD END! => need to backtrack ;-(

Left-corner: predict S (TD) -> recognize det (BU)
-> select rule 'NP -> det N' -> recognize N (BU) -> MATCH!
-> select rule 'S -> NP VP '-> input: died – predict VP (TD) -> recognize IV (BU) -> select rule 'VP -> IV'
-> MATCH!

Comparison to other approaches
Example BU vs LC

Grammar: S -> NP VP NP -> det N NP -> PN VP -> IV
VP -> TV NP TV -> plant IV -> died det -> the
N -> plant

Input sentence: the plant died

Bottom-up: the plant died -> det plant died -> det TV died -> det TV IV ->
det TV VP -> DEAD END! => need to backtrack ;-(

Left-corner: predict S (TD) -> recognize det (BU)
-> select rule 'NP -> det N' -> recognize N (BU) -> MATCH!
-> select rule 'S -> NP VP' -> input: died – predict VP (TD) ->
recognize IV (BU) -> select rule 'VP -> IV'
-> MATCH! => successful parse

Comparison to other approaches
Conclusion and outlook

Comparison to other approaches
Conclusion and outlook

- left-corner diminishes risk of having to backtrack after a series of wrong moves

Comparison to other approaches
Conclusion and outlook

- left-corner diminishes risk of having to backtrack after a series of wrong moves
- but: also combines some of the problems TD and BU have => hardly used in practice

Comparison to other approaches
Conclusion and outlook

- left-corner parsing might be a good model for the human parser!

Comparison to other approaches
Conclusion and outlook

- left-corner parsing might be a good model for the human parser!

Complexity issues:

Strategy	Left-branching	Center Embedding	Right-branching
TD	O(n)	O(n)	O(1)
BU	O(1)	O(n)	O(n)
LC	O(1)	O(n)	O(1)

table taken from Shravan Vasishth's HSP slides

Questions?

Bibliography

- Naumann, Sven and Langer, Haben 1994. *Parsing. Eine Einführung in die maschinelle Analyse natürlicher Sprache*. B.G. Teubner Stuttgart
- a very short section from the Grune & Jacobs book
- http://www.coli.uni-saarland.de/~kris/nlp-with-prolog/html/node53.html
- Shravan Vasishth's slides for the Human Sentence Processing seminar from last semester

Thanks for your attention!