Computational Linguistics II: Parsing

Introduction to Parsing

Frank Richter & Jan-Philipp Söhn

fr@sfs.uni-tuebingen.de, jp.soehn@uni-tuebingen.de

November 27th, 2006

Warm-Up

Ambiguity in Grammar

Def. **ambiguity**:
A sentence can be assigned more than one syntactic tree.

Two types of ambiguity:

- Spurious ambiguity
  - All trees describe the same semantics
  - e.g. 2+5+4 $\Rightarrow$ (2+5)+4 or 2+(5+4)
- Essential ambiguity
  - At least two trees differ in semantics
  - e.g. 2-5-4 $\Rightarrow$ (2-5)-4 or 2-(5-4)

Ambiguity in Grammar

Def. **ambiguity**:
A sentence can be assigned more than one syntactic tree.

Two types of ambiguity:

- Spurious ambiguity
  - All trees describe the same semantics
  - e.g. 2+5+4 $\Rightarrow$ (2+5)+4 or 2+(5+4)
- Essential ambiguity
  - At least two trees differ in semantics
  - e.g. 2-5-4 $\Rightarrow$ (2-5)-4 or 2-(5-4)

Ambiguity in Grammar

Def. **ambiguity**:
A sentence can be assigned more than one syntactic tree.

Two types of ambiguity:

- Spurious ambiguity
  - All trees describe the same semantics
  - e.g. 2+5+4 $\Rightarrow$ (2+5)+4 or 2+(5+4)
- Essential ambiguity
  - At least two trees differ in semantics
  - e.g. 2-5-4 $\Rightarrow$ (2-5)-4 or 2-(5-4)

Ambiguity in Grammar

Def. **ambiguity**:
A sentence can be assigned more than one syntactic tree.

Two types of ambiguity:

- Spurious ambiguity
  - All trees describe the same semantics
  - e.g. 2+5+4 $\Rightarrow$ (2+5)+4 or 2+(5+4)
- Essential ambiguity
  - At least two trees differ in semantics
  - e.g. 2-5-4 $\Rightarrow$ (2-5)-4 or 2-(5-4)

Ambiguity in Grammar

Def. **ambiguity**:
A sentence can be assigned more than one syntactic tree.

Two types of ambiguity:

- Spurious ambiguity
  - All trees describe the same semantics
  - e.g. 2+5+4 $\Rightarrow$ (2+5)+4 or 2+(5+4)
- Essential ambiguity
  - At least two trees differ in semantics
  - e.g. 2-5-4 $\Rightarrow$ (2-5)-4 or 2-(5-4)

Ambiguity in Grammar

Def. **ambiguity**:
A sentence can be assigned more than one syntactic tree.

Two types of ambiguity:

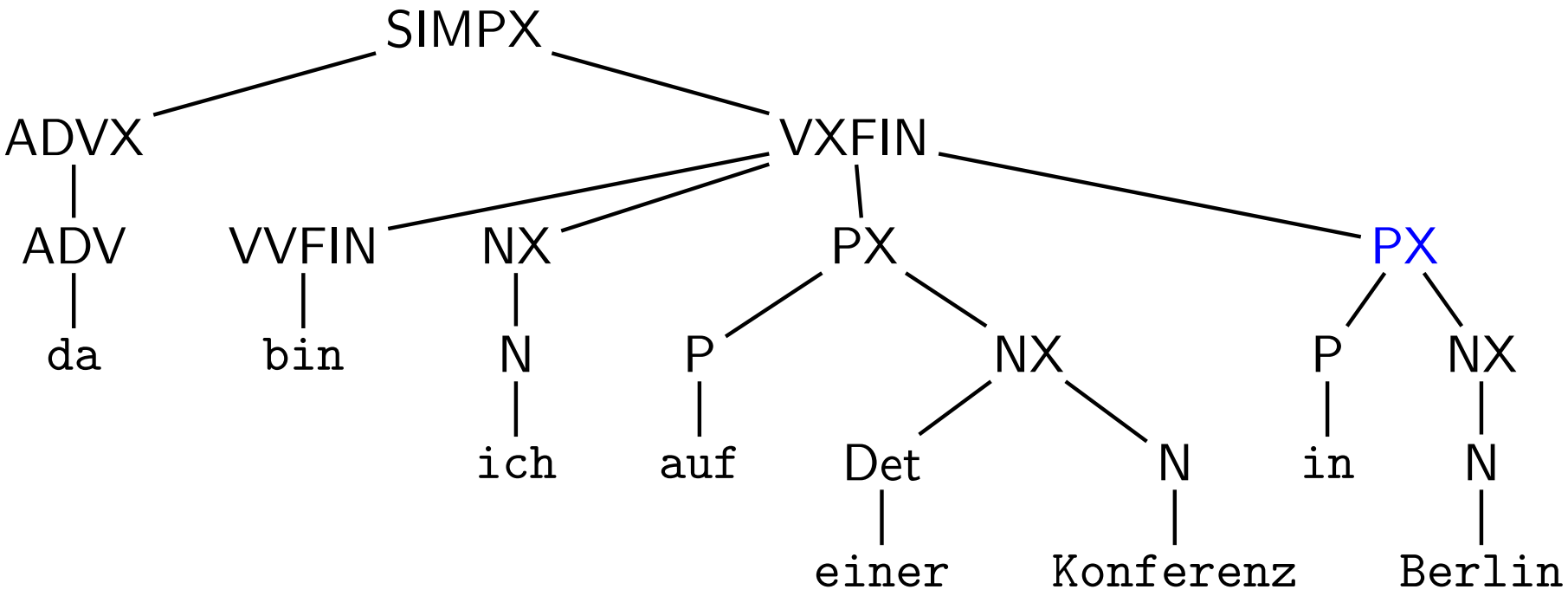
- Spurious ambiguity
  - All trees describe the same semantics
  - e.g. 2+5+4 $\Rightarrow$ (2+5)+4 or 2+(5+4)
- Essential ambiguity
  - At least two trees differ in semantics
  - e.g. 2-5-4 $\Rightarrow$ (2-5)-4 or 2-(5-4)

Ambiguity in Grammar

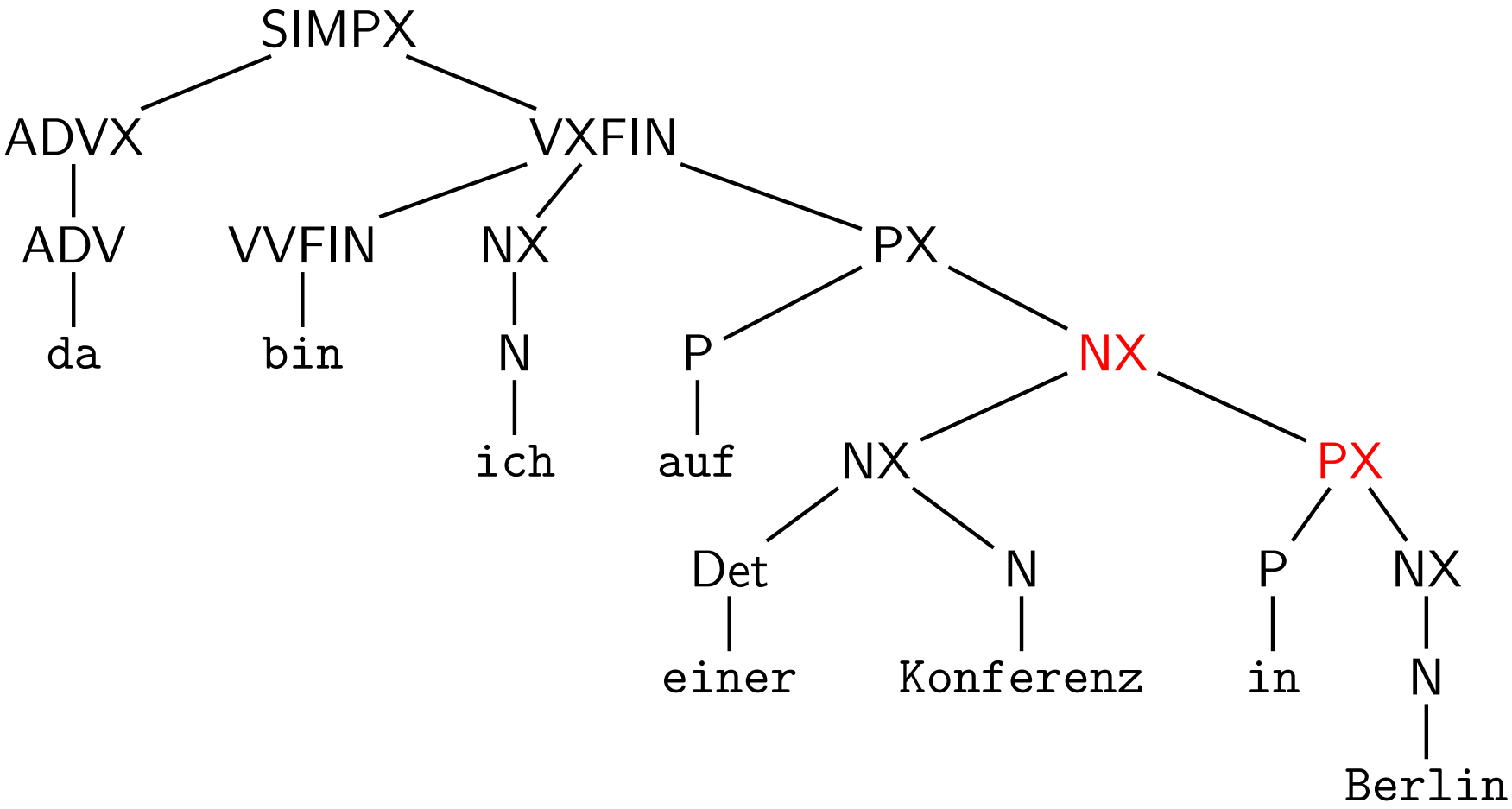
Does the following sentence have 1 or 2 semantic interpretations?

```
da bin ich auf einer Konferenz in Berlin
```

Ambiguity in Grammar



Ambiguity in Grammar



Linearization of parse trees

The only information necessary for reconstructing a tree is the sequence of rules and the notation:

- Prefix notation (i.e. leftmost derivation): each node is listed by listing its number followed by prefix listings of its daughters.
- Postfix notation (i.e. rightmost derivation): each node is listed by listing all daughters in postfix notation followed by node's number.
- infix notation (i.e. left-corner derivation): each node is surrounded by lists of its left and right daughters. The number of daughters which belong to the left list is determined in advance.

Linearization of parse trees

The only information necessary for reconstructing a tree is the sequence of rules and the notation:

- Prefix notation (i.e. leftmost derivation): each node is listed by listing its number followed by prefix listings of its daughters.
- Postfix notation (i.e. rightmost derivation): each node is listed by listing all daughters in postfix notation followed by node's number.
- infix notation (i.e. left-corner derivation): each node is surrounded by lists of its left and right daughters. The number of daughters which belong to the left list is determined in advance.

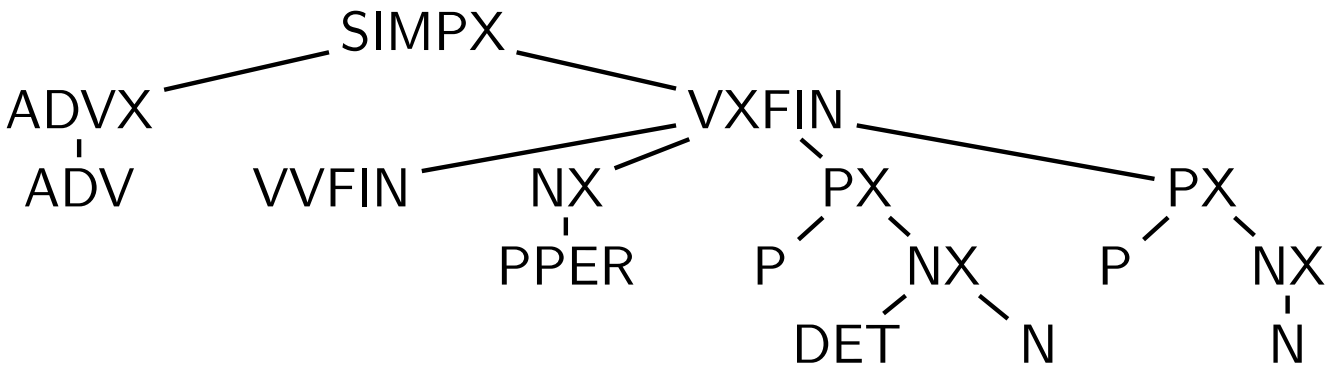
Linearization of parse trees

The only information necessary for reconstructing a tree is the sequence of rules and the notation:

- Prefix notation (i.e. leftmost derivation): each node is listed by listing its number followed by prefix listings of its daughters.
- Postfix notation (i.e. rightmost derivation): each node is listed by listing all daughters in postfix notation followed by node's number.
- infix notation (i.e. left-corner derivation): each node is surrounded by lists of its left and right daughters. The number of daughters which belong to the left list is determined in advance.

An Example

1: SIMPX → ADVX VXFIN
2: ADVX → ADV
3: VXFIN → VVFIN NX PX PX
4: NX → DET N
5: NX → N | PPER
6: PX → P NX



Classification of Parsing Algorithms

Direction of Processing the Input

- Unidirectional
  - left – right
  - right – left
- Bidirectional

Direction of Processing the Input

- Unidirectional
  - left – right
  - right – left
- Bidirectional

Direction of Processing the Input

- Unidirectional
  - left – right
  - right – left
- Bidirectional

Direction of Processing the Input

- Unidirectional
  - left – right
  - right – left
- Bidirectional

Rule Invocation Strategy

- Unidirectional
  - Top-down
    - starts from start symbol, expands non-terminals
    - describes "production" side
  - Bottom-up
    - starts from the terminals, replaces righthand side of rules by mothernodes
    - describes "recognition / perception" side
- Bidirectional

Rule Invocation Strategy

- Unidirectional
  - Top-down
    - starts from start symbol, expands non-terminals
    - describes "production" side
  - Bottom-up
    - starts from the terminals, replaces righthand side of rules by mothernodes
    - describes "recognition / perception" side
- Bidirectional

Rule Invocation Strategy

- Unidirectional
  - Top-down
    - starts from start symbol, expands non-terminals
    - describes "production" side
  - Bottom-up
    - starts from the terminals, replaces righthand side of rules by mothernodes
    - describes "recognition / perception" side
- Bidirectional

Rule Invocation Strategy

- Unidirectional
  - Top-down
    - starts from start symbol, expands non-terminals
    - describes "production" side
  - Bottom-up
    - starts from the terminals, replaces righthand side of rules by mothernodes
    - describes "recognition / perception" side
- Bidirectional

Rule Invocation Strategy

- Unidirectional
  - Top-down
    - starts from start symbol, expands non-terminals
    - describes "production" side
  - Bottom-up
    - starts from the terminals, replaces righthand side of rules by mothernodes
    - describes "recognition / perception" side
- Bidirectional

Rule Invocation Strategy

- Unidirectional
  - Top-down
    - starts from start symbol, expands non-terminals
    - describes "production" side
  - Bottom-up
    - starts from the terminals, replaces righthand side of rules by mothernodes
    - describes "recognition / perception" side
- Bidirectional

Rule Invocation Strategy

- Unidirectional
  - Top-down
    - starts from start symbol, expands non-terminals
    - describes "production" side
  - Bottom-up
    - starts from the terminals, replaces righthand side of rules by mothernodes
    - describes "recognition / perception" side
- Bidirectional

Rule Invocation Strategy

- Unidirectional
  - Top-down
    - starts from start symbol, expands non-terminals
    - describes "production" side
  - Bottom-up
    - starts from the terminals, replaces righthand side of rules by mothernodes
    - describes "recognition / perception" side
- Bidirectional

Top-Down Parsing

- start state: start symbol
- internal administration: stores sentential form
  e.g. `the man with the green hat bought` NP PP
- two actions: *predict* and *match*
- predict: select a non-terminal symbol in sentential form and replace it by a righthand side of a rule
  e.g. `the man with the green hat bought` NP PP $\Rightarrow$ `the man with the green hat bought` DET N PP
- match: compare a terminal symbol with the next word in the input string
  e.g. DET N PP VP + `the man with the green hat bought a book`

Top-Down Parsing

- start state: start symbol
- internal administration: stores sentential form
  e.g. `the man with the green hat bought` NP PP
- two actions: *predict* and *match*
- predict: select a non-terminal symbol in sentential form and replace it by a righthand side of a rule
  e.g. `the man with the green hat bought` NP PP $\Rightarrow$ `the man with the green hat bought` DET N PP
- match: compare a terminal symbol with the next word in the input string
  e.g. DET N PP VP + `the man with the green hat bought a book`

Top-Down Parsing

- start state: start symbol
- internal administration: stores sentential form
  e.g. `the man with the green hat bought` NP PP
- two actions: *predict* and *match*
- predict: select a non-terminal symbol in sentential form and replace it by a righthand side of a rule
  e.g. `the man with the green hat bought` NP PP $\Rightarrow$ `the man with the green hat bought` DET N PP
- match: compare a terminal symbol with the next word in the input string
  e.g. DET N PP VP + `the man with the green hat bought a book`

Top-Down Parsing

- start state: start symbol
- internal administration: stores sentential form
  e.g. `the man with the green hat bought` NP PP
- two actions: *predict* and *match*
- predict: select a non-terminal symbol in sentential form and replace it by a righthand side of a rule
  e.g. `the man with the green hat bought` NP PP $\Rightarrow$ `the man with the green hat bought` DET N PP
- match: compare a terminal symbol with the next word in the input string
  e.g. DET N PP VP + `the man with the green hat bought a book`

Top-Down Parsing

- start state: start symbol
- internal administration: stores sentential form
  e.g. `the man with the green hat bought` NP PP
- two actions: *predict* and *match*
- predict: select a non-terminal symbol in sentential form and replace it by a righthand side of a rule
  e.g. `the man with the green hat bought` NP PP $\Rightarrow$ `the man with the green hat bought` DET N PP
- match: compare a terminal symbol with the next word in the input string
  e.g. DET N PP VP + `the man with the green hat bought a book`

Bottom-Up Parsing

- start state: input string
- internal administration: stores sentential form already processed
  e.g. admin: NP P DET N input: `bought a book`
- two actions: *shift* and *reduce*
- shift: move the next word from the input string to the internal administration
  e.g. NP P DET N $\Rightarrow$ NP P DET N `bought`
- reduce: replace the (rightmost) sequence of symbols in the internal administration by a lefthand side symbol of a rule
  e.g. NP P NP $\Rightarrow$ NP PP

Bottom-Up Parsing

- start state: input string
- internal administration: stores sentential form already processed
  e.g. admin: NP P DET N input: `bought a book`
- two actions: *shift* and *reduce*
- shift: move the next word from the input string to the internal administration
  e.g. NP P DET N $\Rightarrow$ NP P DET N `bought`
- reduce: replace the (rightmost) sequence of symbols in the internal administration by a lefthand side symbol of a rule
  e.g. NP P NP $\Rightarrow$ NP PP

Bottom-Up Parsing

- start state: input string
- internal administration: stores sentential form already processed
  e.g. admin: NP P DET N input: `bought a book`
- two actions: *shift* and *reduce*
- shift: move the next word from the input string to the internal administration
  e.g. NP P DET N $\Rightarrow$ NP P DET N `bought`
- reduce: replace the (rightmost) sequence of symbols in the internal administration by a lefthand side symbol of a rule
  e.g. NP P NP $\Rightarrow$ NP PP

Bottom-Up Parsing

- start state: input string
- internal administration: stores sentential form already processed
  e.g. admin: NP P DET N input: `bought a book`
- two actions: *shift* and *reduce*
- shift: move the next word from the input string to the internal administration
  e.g. NP P DET N $\Rightarrow$ NP P DET N `bought`
- reduce: replace the (rightmost) sequence of symbols in the internal administration by a lefthand side symbol of a rule
  e.g. NP P NP $\Rightarrow$ NP PP

Bottom-Up Parsing

- start state: input string
- internal administration: stores sentential form already processed
  e.g. admin: NP P DET N input: `bought a book`
- two actions: *shift* and *reduce*
- shift: move the next word from the input string to the internal administration
  e.g. NP P DET N $\Rightarrow$ NP P DET N `bought`
- reduce: replace the (rightmost) sequence of symbols in the internal administration by a lefthand side symbol of a rule
  e.g. NP P NP $\Rightarrow$ NP PP

Bottom-Up Parsing II

Bottom-up process can be regarded as production with a "reversed" grammar:

ADVX VXFIN	$\rightarrow$	SIMPX
ADV	$\rightarrow$	ADVX
VVFIN NX PX PX	$\rightarrow$	VXFIN
DET N	$\rightarrow$	NX
N | PPER	$\rightarrow$	NX
P NX	$\rightarrow$	PX

- we need a new terminal symbol: SIMPX $\rightarrow$!
- and a new start symbol:
  Start $\rightarrow$ `da habe ich eine Konferenz in Berlin`

Non-Deterministic Automata

- both parsing methods have two components: one makes the substitutions and stores the parse tree, the other decides which rule to use next
- the substituting component can be defined as a **non-deterministic automaton** (NDA)
- it is non-deterministic because in some states, it has more than one possible rule to choose from, e.g. `NX` $\longrightarrow$ `DET N` or `NX` $\longrightarrow$ `PPER`
- the decision which rule is chosen is made by the control component
- control strategies vary and can be very complex
- one more component: book keeping component for the parse tree

Non-Deterministic Automata

- both parsing methods have two components: one makes the substitutions and stores the parse tree, the other decides which rule to use next
- the substituting component can be defined as a **non-deterministic automaton** (NDA)
- it is non-deterministic because in some states, it has more than one possible rule to choose from, e.g. `NX` $\longrightarrow$ `DET N` or `NX` $\longrightarrow$ `PPER`
- the decision which rule is chosen is made by the control component
- control strategies vary and can be very complex
- one more component: book keeping component for the parse tree

Non-Deterministic Automata

- both parsing methods have two components: one makes the substitutions and stores the parse tree, the other decides which rule to use next
- the substituting component can be defined as a **non-deterministic automaton** (NDA)
- it is non-deterministic because in some states, it has more than one possible rule to choose from, e.g. `NX` $\longrightarrow$ `DET N` or `NX` $\longrightarrow$ `PPER`
- the decision which rule is chosen is made by the control component
- control strategies vary and can be very complex
- one more component: book keeping component for the parse tree

Non-Deterministic Automata

- both parsing methods have two components: one makes the substitutions and stores the parse tree, the other decides which rule to use next
- the substituting component can be defined as a **non-deterministic automaton** (NDA)
- it is non-deterministic because in some states, it has more than one possible rule to choose from, e.g. `NX` $\longrightarrow$ `DET N` or `NX` $\longrightarrow$ `PPER`
- the decision which rule is chosen is made by the control component
- control strategies vary and can be very complex
- one more component: book keeping component for the parse tree

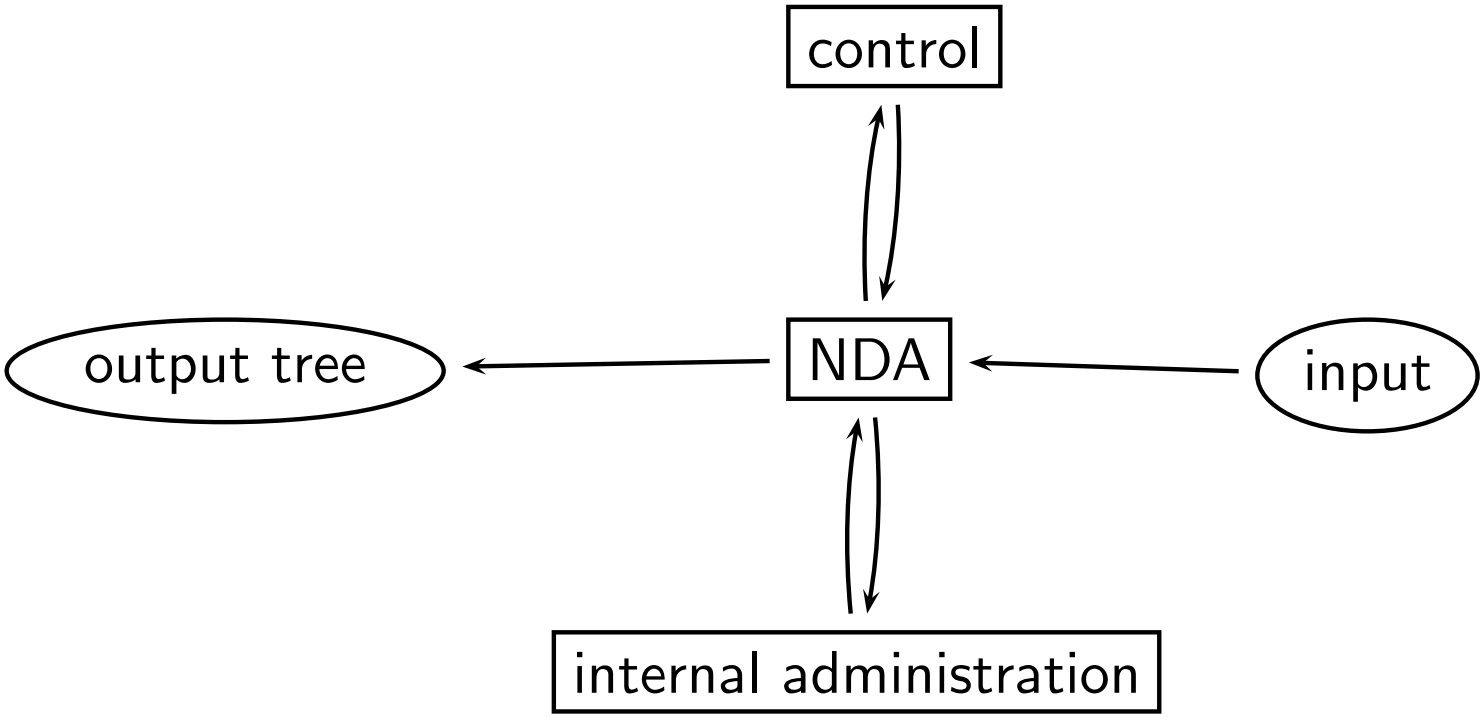
Non-Deterministic Automata

- both parsing methods have two components: one makes the substitutions and stores the parse tree, the other decides which rule to use next
- the substituting component can be defined as a **non-deterministic automaton** (NDA)
- it is non-deterministic because in some states, it has more than one possible rule to choose from, e.g. `NX` $\longrightarrow$ `DET N` or `NX` $\longrightarrow$ `PPER`
- the decision which rule is chosen is made by the control component
- control strategies vary and can be very complex
- one more component: book keeping component for the parse tree

Non-Deterministic Automata

- both parsing methods have two components: one makes the substitutions and stores the parse tree, the other decides which rule to use next
- the substituting component can be defined as a **non-deterministic automaton** (NDA)
- it is non-deterministic because in some states, it has more than one possible rule to choose from, e.g. `NX` $\longrightarrow$ `DET N` or `NX` $\longrightarrow$ `PPER`
- the decision which rule is chosen is made by the control component
- control strategies vary and can be very complex
- one more component: book keeping component for the parse tree

Non-Deterministic Automata II



Non-Deterministic Automata III

- the NDA is derived from the grammar
- for top-down parsing, the moves are given by the rules, the administration at the beginning contains the start symbol
- there are many control mechanisms, some are independent of the grammar (e.g. undo the last step), some use tables extracted in advance from the grammar (if I choose `NX` $\longrightarrow$ `NX1 KON NX1` and my next word is *the*, is there a way that `NX1` produces a string that begins with *the*?)
- **parser generator**: program that constructs the control mechanism and the NDA for a specific parsing algorithm

Non-Deterministic Automata III

- the NDA is derived from the grammar
- for top-down parsing, the moves are given by the rules, the administration at the beginning contains the start symbol
- there are many control mechanisms, some are independent of the grammar (e.g. undo the last step), some use tables extracted in advance from the grammar (if I choose `NX` $\longrightarrow$ `NX1 KON NX1` and my next word is *the*, is there a way that `NX1` produces a string that begins with *the*?)
- **parser generator**: program that constructs the control mechanism and the NDA for a specific parsing algorithm

Non-Deterministic Automata III

- the NDA is derived from the grammar
- for top-down parsing, the moves are given by the rules, the administration at the beginning contains the start symbol
- there are many control mechanisms, some are independent of the grammar (e.g. undo the last step), some use tables extracted in advance from the grammar (if I choose `NX` $\longrightarrow$ `NX1 KON NX1` and my next word is *the*, is there a way that `NX1` produces a string that begins with *the*?)
- **parser generator**: program that constructs the control mechanism and the NDA for a specific parsing algorithm

Non-Deterministic Automata III

- the NDA is derived from the grammar
- for top-down parsing, the moves are given by the rules, the administration at the beginning contains the start symbol
- there are many control mechanisms, some are independent of the grammar (e.g. undo the last step), some use tables extracted in advance from the grammar (if I choose `NX` $\longrightarrow$ `NX1 KON NX1` and my next word is *the*, is there a way that `NX1` produces a string that begins with *the*?)
- **parser generator**: program that constructs the control mechanism and the NDA for a specific parsing algorithm

Searching Strategy

Parsing can be regarded as search: parsing as a graph from start state to end state ⇒ look for the shortest / best path

How does the control mechanism decide on the next move?

- depth-first (backtracking) or breadth-first?
- Deterministic or non-deterministic
- ...

Searching Strategy

Parsing can be regarded as search: parsing as a graph from start state to end state ⇒ look for the shortest / best path

How does the control mechanism decide on the next move?

- depth-first (backtracking) or breadth-first?
- Deterministic or non-deterministic
- ...

Searching Strategy

Parsing can be regarded as search: parsing as a graph from start state to end state ⇒ look for the shortest / best path

How does the control mechanism decide on the next move?

- depth-first (backtracking) or breadth-first?
- Deterministic or non-deterministic
- ...

Parsers and Parser Generators

	Parser abstract machine 1st order	**Parser Generator** abstract machine 2nd order
Data	syntax lexicon	parsing algorithm
Processing	parsing algorithm	generating algorithm
Input	sentence	syntax lexicon
Output	structure or FALSE	parser

Putting all together

	Top-down	Bottom-up
Non-directional methods	Unger parser	CYK parser
Directional methods	predict/match Depth-first (backtrack) Breadth-first, DCGs	shift/reduce Depth-first (backtrack) Breadth-first (Earley)
	Left-corner	
Linear directional methods		LR(k) SLR(1)
Efficient general directional methods		Tomita