Computational Linguistics II: Parsing

CSGs, Turing Machines, LBAs
Outlook: Other Grammar Formalisms

Frank Richter & Jan-Philipp Söhn

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

The Big Picture

hierarchy	grammar	machine	other
type 3	reg. grammar	D/NFA	reg. expressions
det. cf.	LR(k) grammar	DPDA	
type 2	CFG	PDA	
type 1	CSG	LBA	
type 0	unrestricted grammar	Turing machine	

Context Sensitive Grammars (1)

Definition

A grammar $\langle N, T, P, S \rangle$ is *context-sensitive* iff every production in P is of the form $x_1 A x_2 \rightarrow x_1 y x_2$, with $x_1, x_2 \in \Sigma^*$, $y \in \Sigma^+$, $A \in N$ and the possible exception of $C \rightarrow \epsilon$ in case C does not occur on the righthand side of a rule in P.

Definition

A grammar $\langle N, T, P, S \rangle$ is *monotonic* iff for every production $l \rightarrow r \in P$, $|l| \leq |r|$.

Context Sensitive Grammars (2)

We will not prove the following important theorem:

Theorem

(I) For every monotonic grammar G_M there is a context-sensitive grammar G_S such that $L(G_M) = L(G_S)$.

(II) For every context-sensitive grammar G_S there is a monotonic grammar G_M such that $L(G_S) = L(G_M)$.

Remark:
The languages generated by monotonic and context-sensitive grammars are generally referred to as context-sensitive languages. Their grammars are called Type 1 grammars.

Kuroda Normal Form (1)

A Type 1 grammar $\langle N, T, P, S \rangle$ is in *Kuroda Normal Form* iff all productions in P are of one of the following forms:

1. $A \rightarrow a$
2. $A \rightarrow B$
3. $A \rightarrow BC$
4. $AB \rightarrow CD$.

(A, B and C in N, a in T)

Kuroda Normal Form (2)

Theorem

For every Type 1 grammar G with $\epsilon \notin L(G)$ there is a grammar G' in Kuroda Normal Form such that $L(G) = L(G')$.

Turing Machine

Definition

A septuple $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ is a *Turing Machine* iff

Turing Machine

Definition

A septuple $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ is a *Turing Machine* iff

Q is a finite set (the set of states),

Turing Machine

Definition

A septuple $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ is a *Turing Machine* iff

Q is a finite set (the set of states),

Σ is a finite set (the input alphabet),

Turing Machine

Definition

A septuple $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ is a *Turing Machine* iff

Q is a finite set (the set of states),

Σ is a finite set (the input alphabet),

Γ is a finite set, $\Sigma \subset \Gamma$ (the set of tape symbols),

Turing Machine

Definition

A septuple $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ is a *Turing Machine* iff

Q is a finite set (the set of states),

Σ is a finite set (the input alphabet),

Γ is a finite set, $\Sigma \subset \Gamma$ (the set of tape symbols),

δ is a function from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R, S\}$ (the next move function),

Turing Machine

Definition

A septuple $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ is a *Turing Machine* iff

Q is a finite set (the set of states),

Σ is a finite set (the input alphabet),

Γ is a finite set, $\Sigma \subset \Gamma$ (the set of tape symbols),

δ is a function from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R, S\}$ (the next move function),

$q_0 \in Q$ (the start state),

Turing Machine

Definition

A septuple $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ is a *Turing Machine* iff

Q is a finite set (the set of states),

Σ is a finite set (the input alphabet),

Γ is a finite set, $\Sigma \subset \Gamma$ (the set of tape symbols),

δ is a function from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R, S\}$ (the next move function),

$q_0 \in Q$ (the start state),

$\Box \in (\Gamma - \Sigma)$ (the blank), and

Turing Machine

Definition

A septuple $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ is a *Turing Machine* iff

Q is a finite set (the set of states),

Σ is a finite set (the input alphabet),

Γ is a finite set, $\Sigma \subset \Gamma$ (the set of tape symbols),

δ is a function from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R, S\}$ (the next move function),

$q_0 \in Q$ (the start state),

$\Box \in (\Gamma - \Sigma)$ (the blank), and

$F \subseteq Q$ (the set of final states).

Configuration of a Turing Machine

Definition

A *configuration of a Turing Machine* $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ is a word $w \in \Gamma^* Q \Gamma^*$.

Move of a Turing Machine

We define the moves, $\vdash$ of a Turing machine from one configuration to the next:

Definition

$$a_1 \ldots a_m q b_1 \ldots b_n \vdash \begin{cases} a_1 \ldots a_m q' c b_2 \ldots b_n, \\ \quad \delta(q, b_1) = (q', c, S), m \geq 0, n \geq 1 \\ a_1 \ldots a_m c q' b_2 \ldots b_n, \\ \quad \delta(q, b_1) = (q', c, R), m \geq 0, n \geq 2 \\ a_1 \ldots a_{m-1} q' a_m c b_2 \ldots b_n, \\ \quad \delta(q, b_1) = (q', c, L), m \geq 1, n \geq 1 \\ a_1 \ldots a_m c q' \Box, \\ \quad \delta(q, b_1) = (q', c, R), m \geq 0, n = 1 \\ q' \Box c b_2 \ldots b_n, \\ \quad \delta(q, b_1) = (q', c, L), m = 0, n \geq 1 \end{cases}$$

Language Accepted by a TM

Definition

The language $L(M)$ accepted by a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ is

$$L(M) = \{x \in \Sigma^* | q_0 x \vdash^* \alpha q \beta; \alpha, \beta \in \Gamma^*; q \in F\}.$$

Linear Bounded Automata

A *linear bounded automaton* is a nondeterministic Turing machine satisfying the following conditions:

1. Its input alphabet includes two special symbols, the left and right endmarkers.
2. The LBA has no moves left from the left endmarker or right from the right endmarker, nor may it print another symbol over them.

Linear Bounded Automata

A *linear bounded automaton* is a nondeterministic Turing machine satisfying the following conditions:

1. Its input alphabet includes two special symbols, the left and right endmarkers.
2. The LBA has no moves left from the left endmarker or right from the right endmarker, nor may it print another symbol over them.

Theorem
The languages accepted by LBAs are exactly the languages generated by CSGs.

German VPs (simplified)

der Arzt gibt dem Patienten die Pille

der Arzt gibt die Pille dem Patienten

dem Patienten gibt der Arzt die Pille

dem Patienten gibt die Pille der Arzt

die Pille gibt der Arzt dem Patienten

die Pille gibt dem Patienten der Arzt

der Arzt gibt ihm die Pille

die Pille gibt ihm der Arzt

dem Patienten gibt sie der Arzt

die Pille gibt er ihm

der Arzt gibt sie ihm

dem Patienten gibt er sie

der Arzt sagt dem Patienten daß er krank ist

dem Patienten sagt der Arzt daß er krank ist

CF Production Rules for German VPs

$$
\begin{aligned}
VP &\rightarrow V\ NP_{dat}\ NP_{akk} \\
VP &\rightarrow V\ NP_{akk}\ NP_{dat} \\
VP &\rightarrow V\ NP_{nom}\ NP_{akk} \\
VP &\rightarrow V\ NP_{akk}\ NP_{nom} \\
VP &\rightarrow V\ NP_{nom}\ NP_{dat} \\
VP &\rightarrow V\ NP_{dat}\ NP_{nom} \\
VP &\rightarrow V\ NPP_{dat}\ NP_{akk} \\
VP &\rightarrow V\ NPP_{dat}\ NP_{nom} \\
VP &\rightarrow V\ NPP_{akk}\ NP_{nom} \\
VP &\rightarrow V\ NPP_{nom}\ NPP_{dat} \\
VP &\rightarrow V\ NPP_{akk}\ NPP_{dat} \\
VP &\rightarrow V\ NPP_{nom}\ NPP_{akk} \\
VP &\rightarrow V\ NP_{dat}\ S \\
VP &\rightarrow V\ NP_{nom}\ S
\end{aligned}
$$

Generalizations

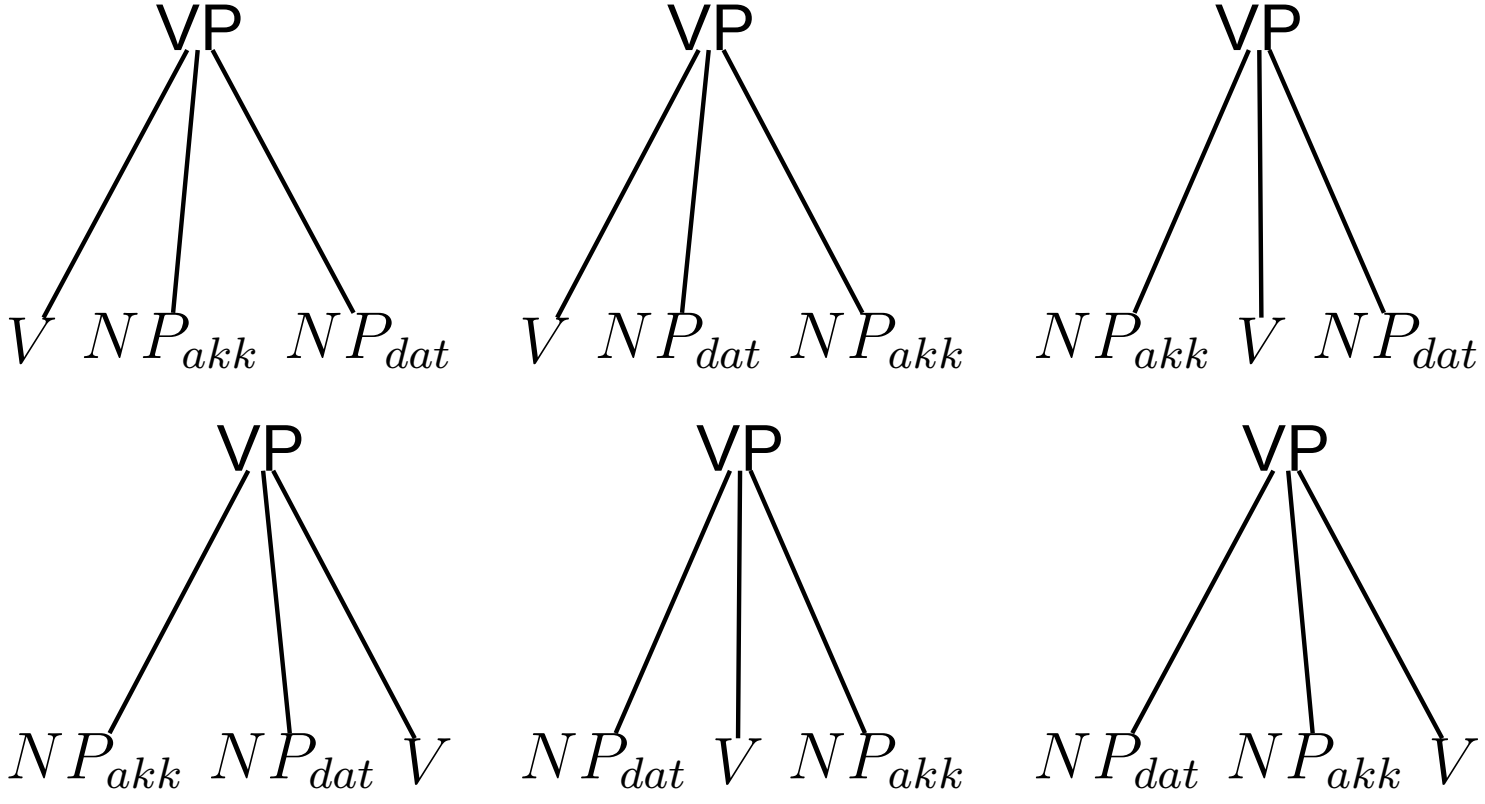
- simple definite NPs come in no particular order
- pronominal NPs precede definite NPs
- pronominal nominative subjects precede dative objects
- pronominal nominative subjects precede accusative objects
- pronominal accusative objects precede pronominal dative objects
- definite NPs precede sentential objects

ID/LP Grammars

- distinguish two different types of information in phrase structure rules: immediate dominance (ID) and linear precedence (LP)
- idea: split ID information from LP information
- ID rules only determine the number of daughters and their syntactic category
- LP rules determine the sequence of nodes in local trees
- ID rule: $VP \rightarrow V, NP_{akk}, NP_{dat}$
- LP rule: $NPP_{nom} \prec NPP_{dat}$
- LP rules apply **globally**

ID/LP: An Example

ID rule: $VP \rightarrow V, NP_{akk}, NP_{dat}$
possible trees:



The German VP Revisited

$$
\begin{aligned}
VP &\rightarrow V\ NP_{dat}\ NP_{akk} \\
VP &\rightarrow V\ NP_{akk}\ NP_{dat} \\
VP &\rightarrow V\ NP_{nom}\ NP_{akk} \\
VP &\rightarrow V\ NP_{akk}\ NP_{nom} \\
VP &\rightarrow V\ NP_{nom}\ NP_{dat} \\
VP &\rightarrow V\ NP_{dat}\ NP_{nom} \\
VP &\rightarrow V\ NPP_{dat}\ NP_{akk} \\
VP &\rightarrow V\ NPP_{dat}\ NP_{nom} \\
VP &\rightarrow V\ NPP_{akk}\ NP_{nom} \\
VP &\rightarrow V\ NPP_{nom}\ NPP_{dat} \\
VP &\rightarrow V\ NPP_{akk}\ NPP_{dat} \\
VP &\rightarrow V\ NPP_{nom}\ NPP_{akk} \\
VP &\rightarrow V\ NP_{dat}\ S \\
VP &\rightarrow V\ NP_{nom}\ S
\end{aligned}
$$

The German VP Revisited

$$
\begin{array}{lcl}
VP & \rightarrow & V\ NP_{dat}\ NP_{akk} \\
VP & \rightarrow & V\ NP_{akk}\ NP_{dat} \\
VP & \rightarrow & V\ NP_{nom}\ NP_{akk} \\
VP & \rightarrow & V\ NP_{akk}\ NP_{nom} \\
VP & \rightarrow & V\ NP_{nom}\ NP_{dat} \\
VP & \rightarrow & V\ NP_{dat}\ NP_{nom} \\
VP & \rightarrow & V\ NPP_{dat}\ NP_{akk} \\
VP & \rightarrow & V\ NPP_{dat}\ NP_{nom} \\
VP & \rightarrow & V\ NPP_{akk}\ NP_{nom} \\
VP & \rightarrow & V\ NPP_{nom}\ NPP_{dat} \\
VP & \rightarrow & V\ NPP_{akk}\ NPP_{dat} \\
VP & \rightarrow & V\ NPP_{nom}\ NPP_{akk} \\
VP & \rightarrow & V\ NP_{dat}\ S \\
VP & \rightarrow & V\ NP_{nom}\ S
\end{array}
$$

$$\Rightarrow$$

$$
\begin{array}{lcl}
VP & \rightarrow & V, NP_{dat}, NP_{akk} \\
VP & \rightarrow & V, NP_{nom}, NP_{akk} \\
VP & \rightarrow & V, NP_{nom}, NP_{dat} \\
VP & \rightarrow & V, NPP_{dat}, NP_{nom} \\
VP & \rightarrow & V, NPP_{akk}, NP_{nom} \\
VP & \rightarrow & V, NPP_{akk}, NPP_{nom} \\
VP & \rightarrow & V, NPP_{dat}, NPP_{nom} \\
VP & \rightarrow & V, NPP_{akk}, NPP_{dat} \\
VP & \rightarrow & V, NPP_{dat}, S \\
VP & \rightarrow & V, NPP_{akk}, S
\end{array}
$$

$$NPP_{nom} \prec NPP_{akk} \prec NPP_{dat} \prec NP_{nom/akk}$$

$$NP_{nom/dat} \prec S, \quad V \prec XP$$

Equivalence of Grammars

- two grammars are **weakly equivalent** iff they describe the same language
- two grammars are **strongly equivalent** iff they are weakly equivalent and assign the same structure(s) to each sentence

- for each ID/LP grammar, there is a strongly equivalent CFG
- for each CFG, there is a weakly equivalent ID/LP grammar
- there are CFGs with a strongly equivalent ID/LP grammar; these grammars possess the ECPO property
- ECPO = exhaustive constant partial ordering

A non-ECPO Grammar

CFG:

$$S_{V2} \rightarrow NP\ V_{fin}\ NP\ V_{pp}$$
$$S_{Vend} \rightarrow KON\ NP\ NP\ V_{pp}\ V_{fin}$$

a weakly equivalent ID/LP grammar:

$$S_{V2} \rightarrow NP_K,\ VX_1,\ NP_Y,\ V_{pp},$$
$$S_{Vend} \rightarrow KON,\ NP,\ NP,\ V_{pp},\ VX_2,$$
$$VX_1 \rightarrow V_{fin}, \qquad NP_K \rightarrow NP,$$
$$VX_2 \rightarrow V_{fin}, \qquad NP_Y \rightarrow NP,$$

$$VX_1 \prec V_{pp}, \qquad NP_K \prec VX_1, \qquad KON \prec NP \prec V_{pp}$$
$$V_{pp} \prec VX_2, \qquad VX_1 \prec NP_Y$$