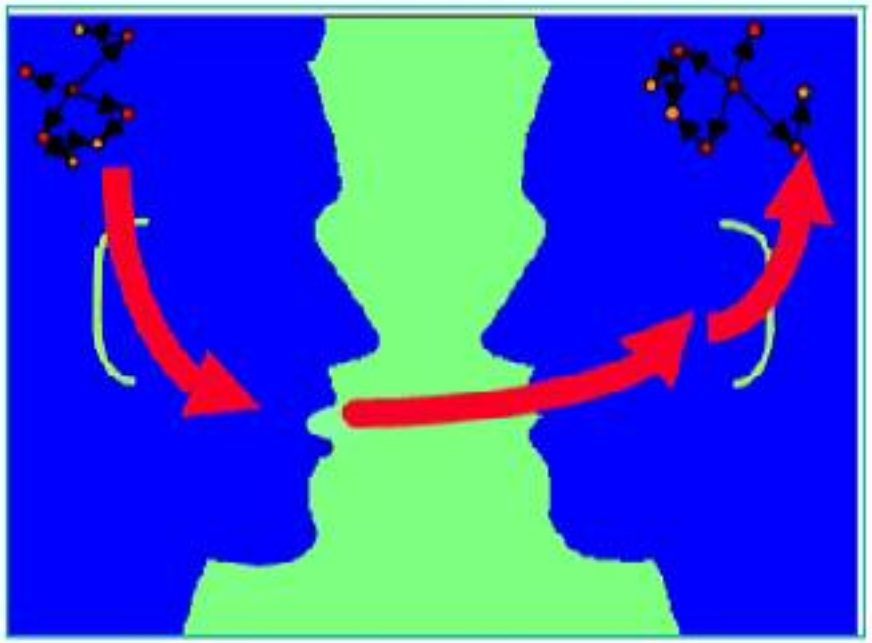
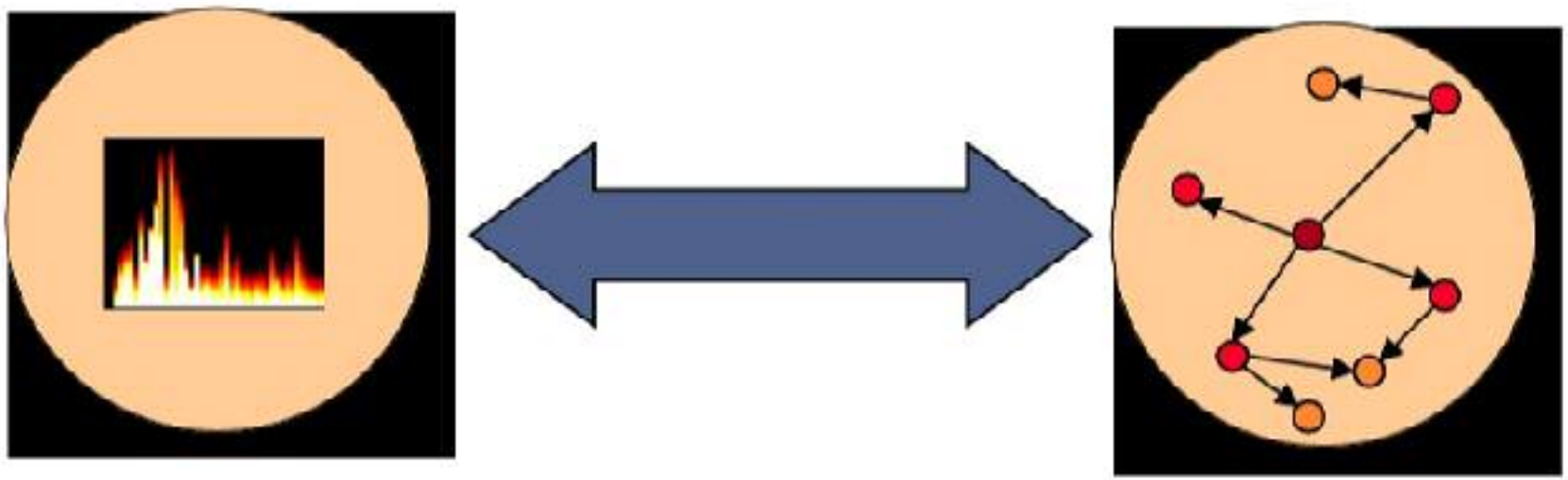
Die Rolle der Grammatik, Ersetzungsregelgrammatiken, und Parsing

Hans Uszkoreit

WHAT HAPPENS IN BETWEEN?



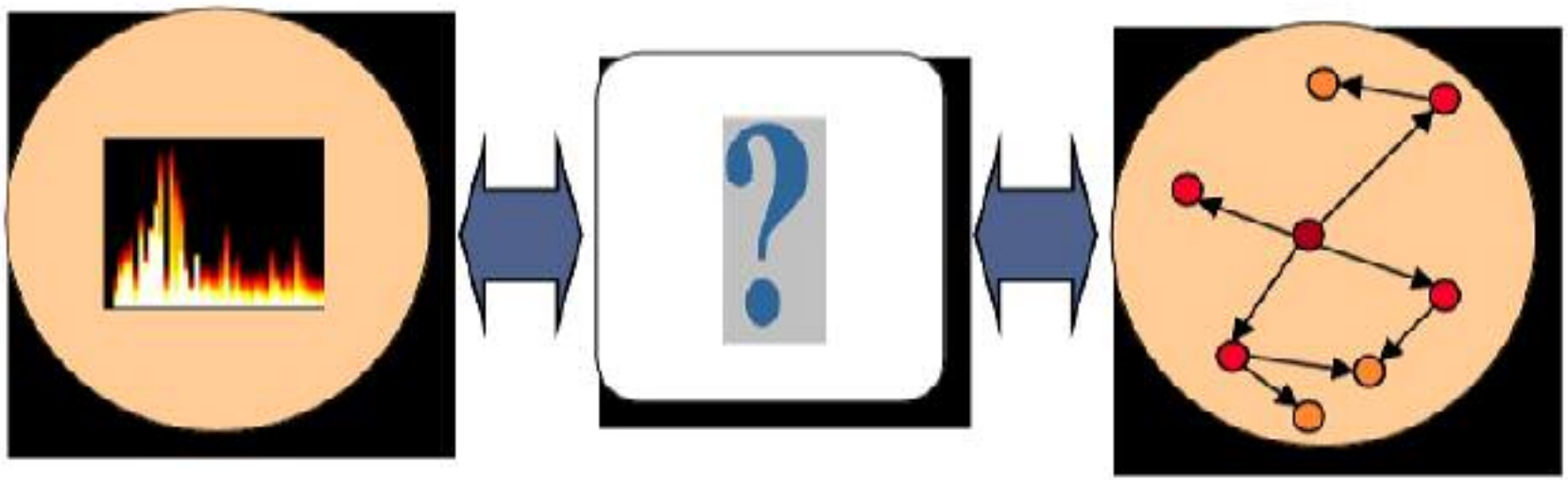
WHAT HAPPENS IN BETWEEN?



sound waves

activation of concepts

WHAT HAPPENS IN BETWEEN?

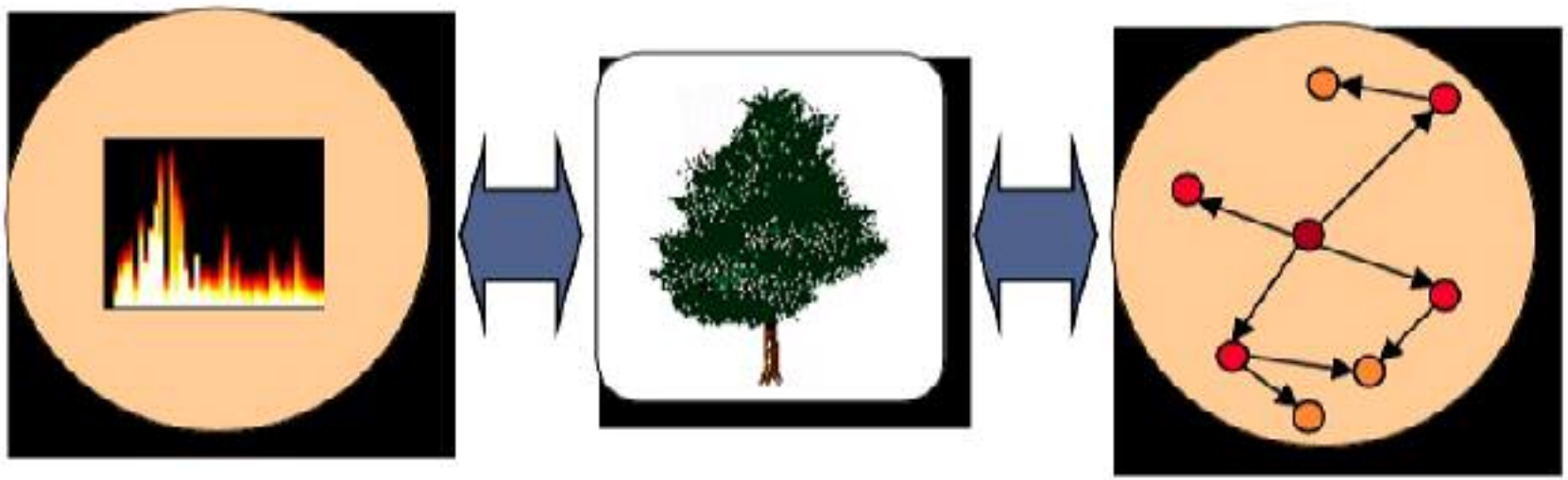


sound waves

Grammar

activation of concepts

WHAT HAPPENS IN BETWEEN?

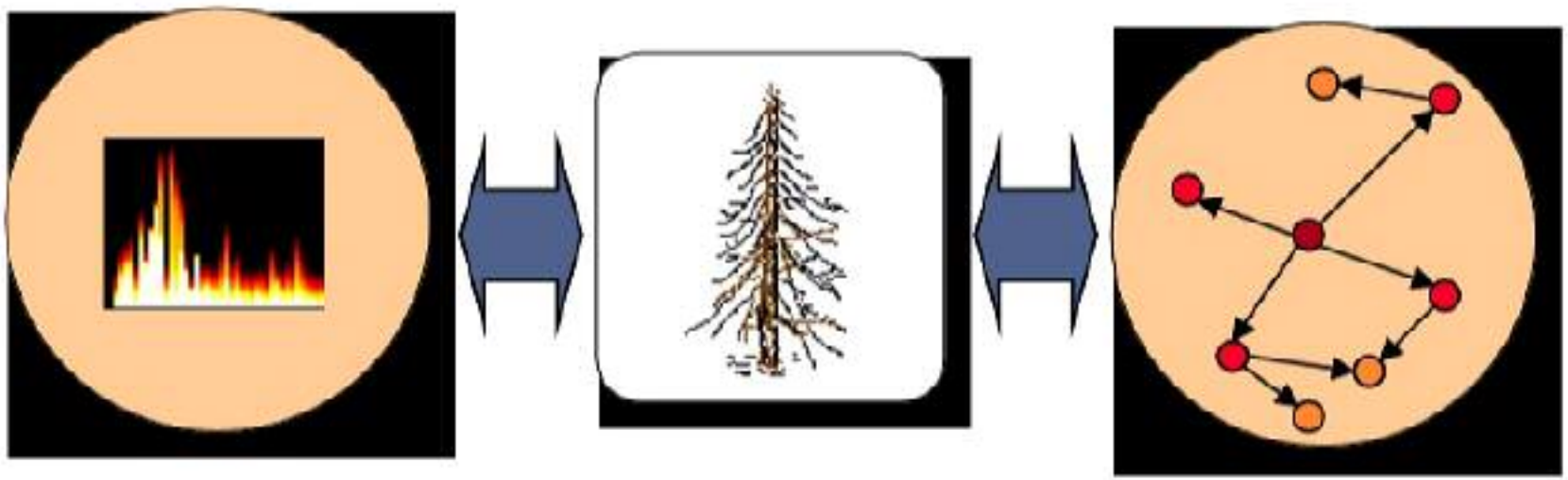


sound waves

Grammar

activation of concepts

WHAT HAPPENS IN BETWEEN?

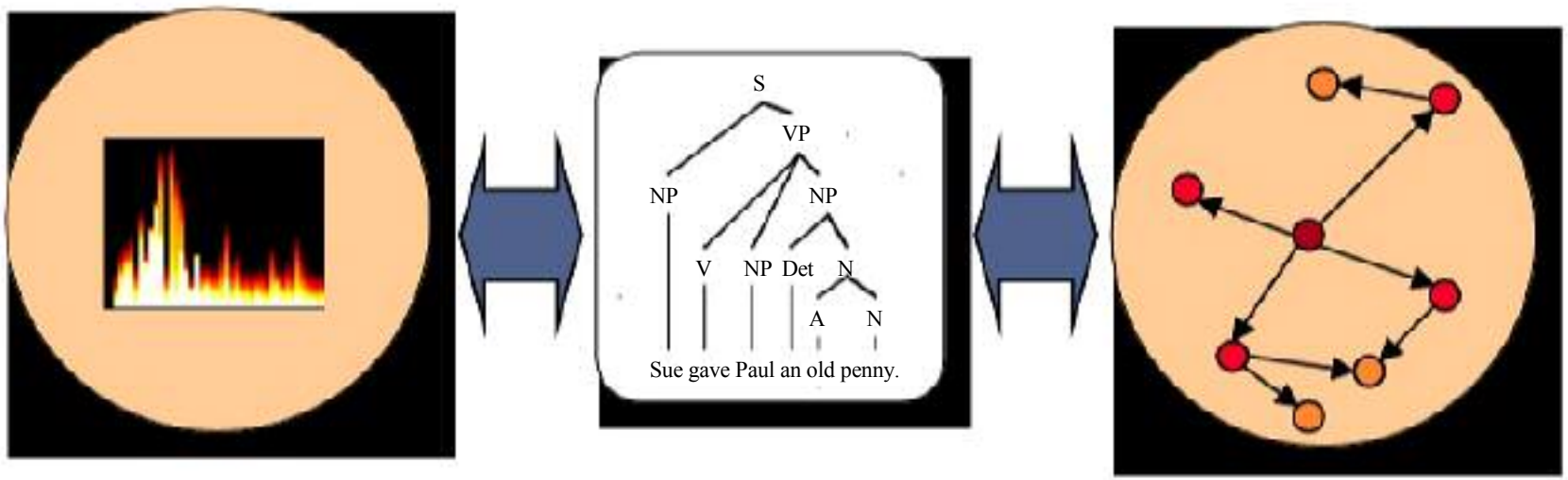


sound waves

Grammar

activation of concepts

WHAT HAPPENS IN BETWEEN?

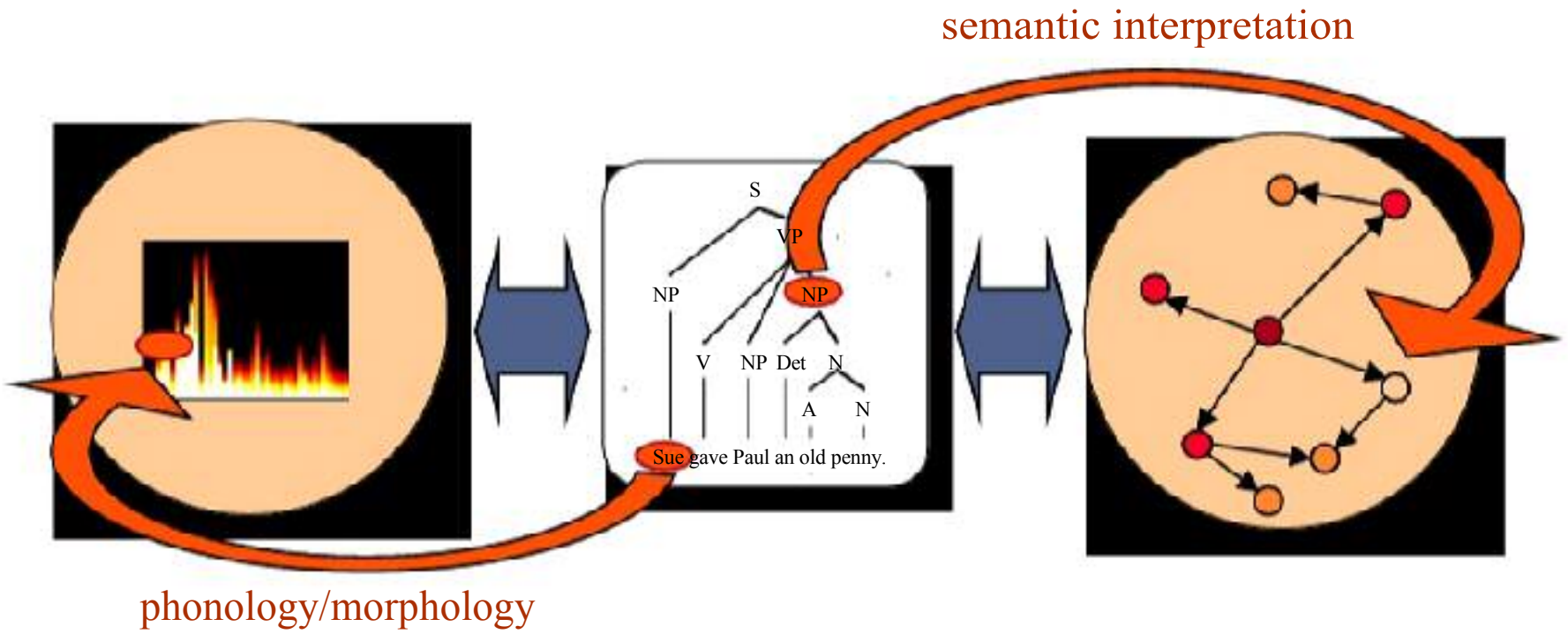


sound waves

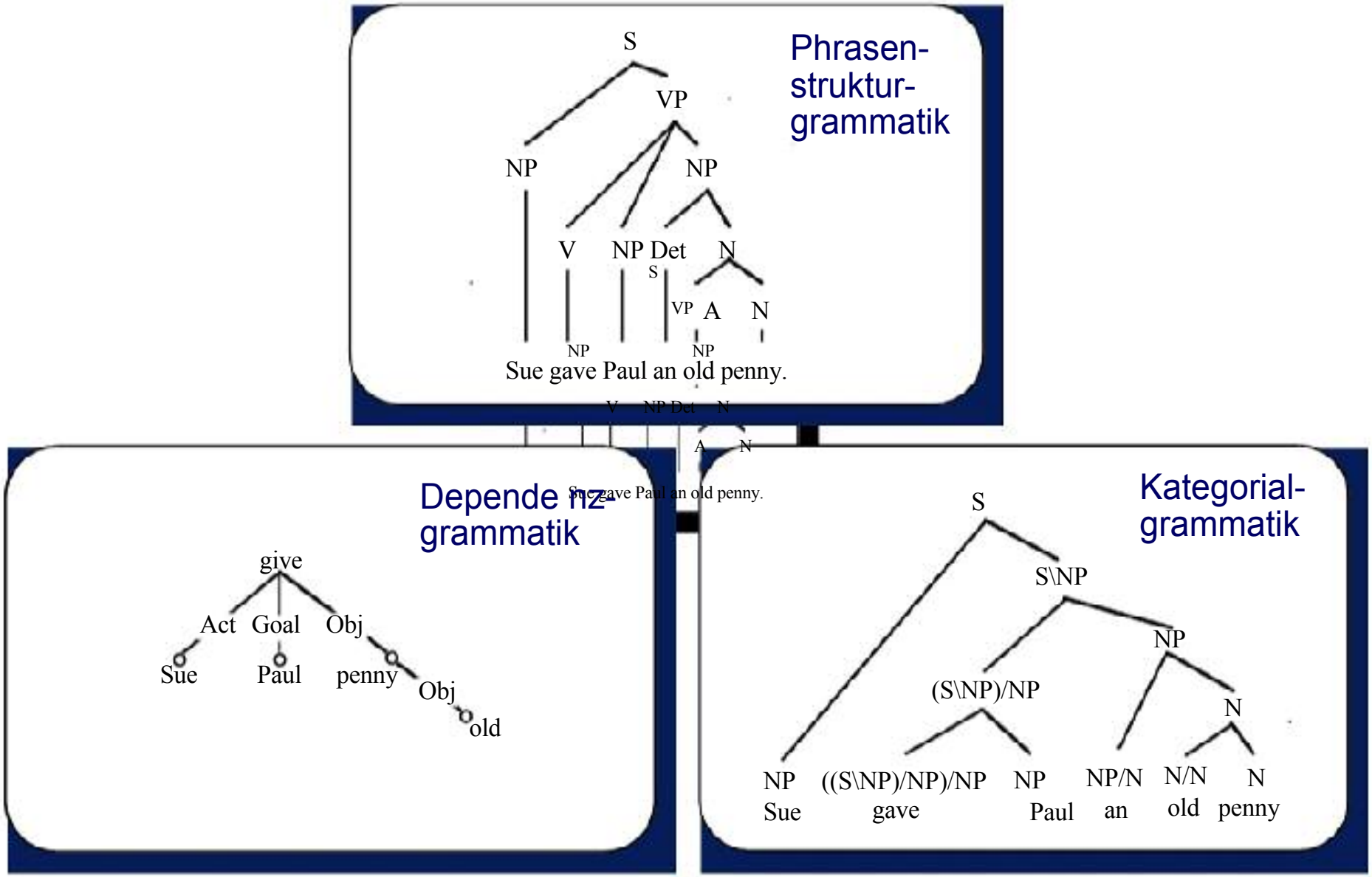
Grammar

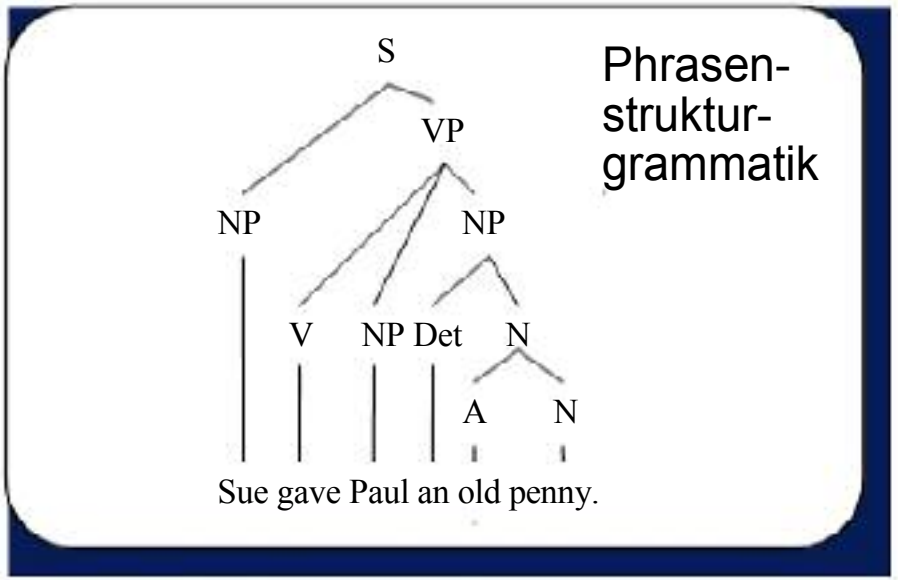
activation of concepts

WHAT HAPPENS IN BETWEEN?

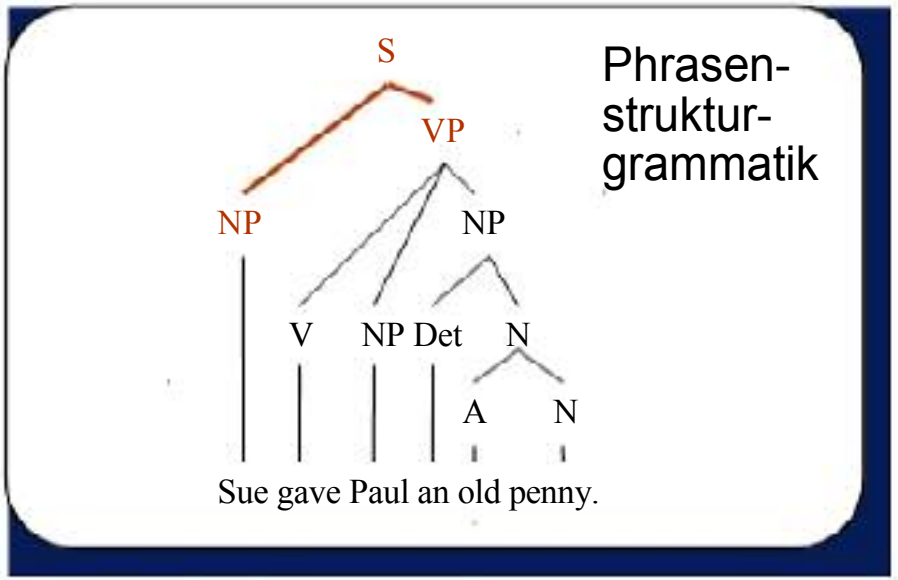


THREE TRADITIONS

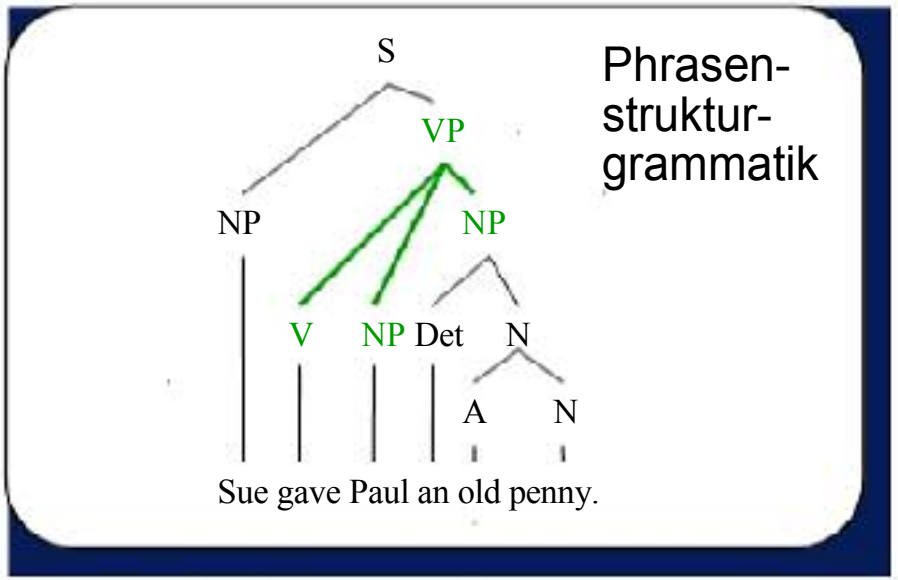




S → NP VP

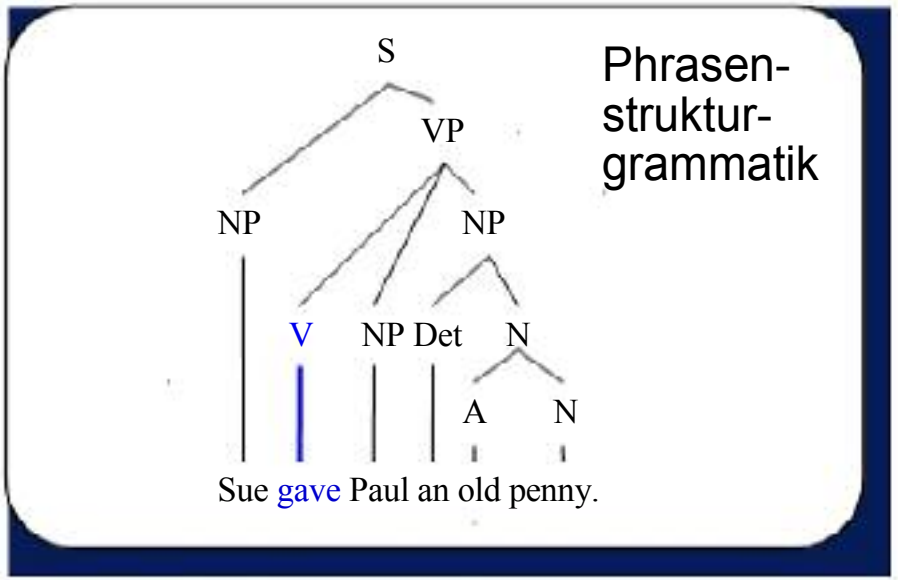


S → NP VP



S → NP VP

VP → V NP NP

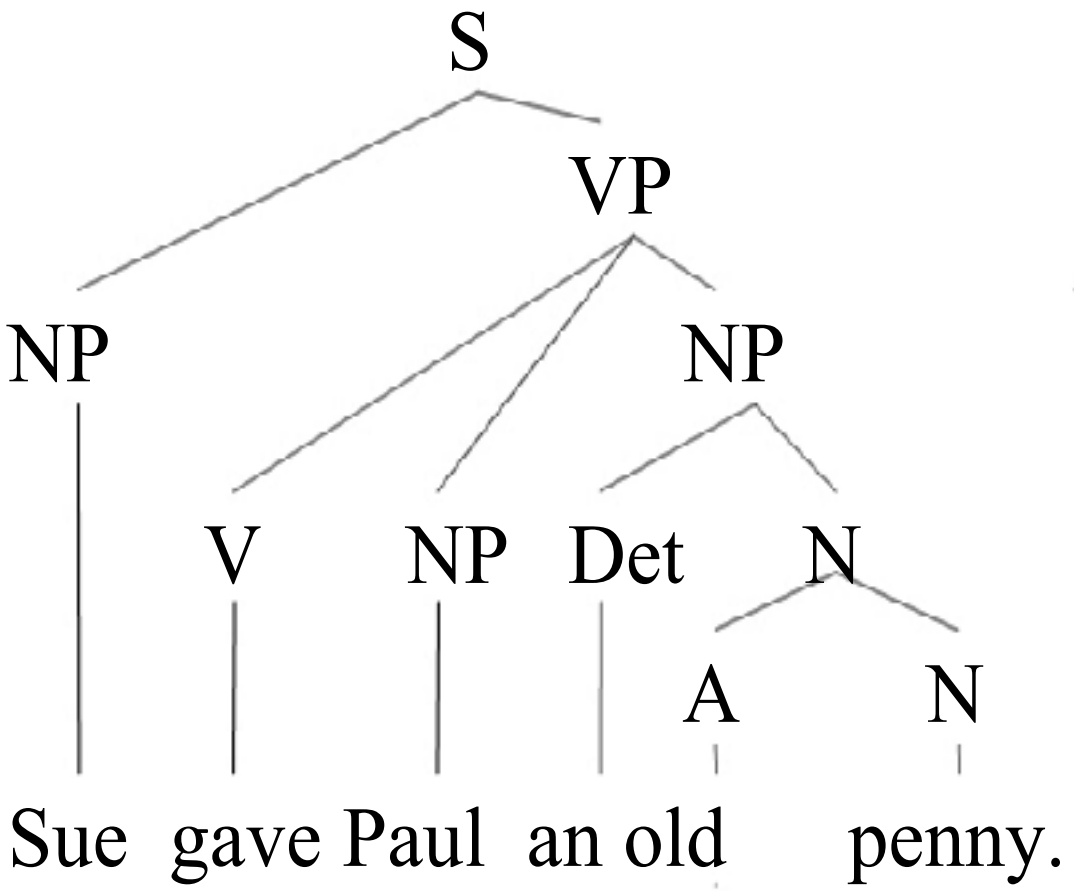


S → NP VP
VP → V NP NP

V → gave

Symbolkonventionen

	Einzelsymbole	Ketten
nichtterminale	A, B, C, ...	... , X, Y, Z
terminale	a, b, c, ...	... , x, y, z
unspezifiziert	α, β, γ, ...	... , φ, χ, ψ, ω
Startsymbol	S	
die leere Kette	e	
Zahlen	..., i, j, k, l, m, n, ...	



Formale Grammatik

Eine Sprache L über einem Alphabet (Vokabular) Σ ist eine Teilmenge von Σ^*.

Eine formale Grammatik G_L für eine Sprache L ist ein Quadrupel $(V_N, V_T, \{S\}, P)$.

V_N - nichtterminales Vokabular, Hilfsalphabet

V_T - terminales Vokabular
($V_T \cap V_N = \emptyset$, $L \subseteq V_T^*$, $V = V_T \cup V_N$)

{S} - Einermenge mit dem Startsymbol, Axiomenmenge

P - Menge der Produktionen, Regelmenge
Menge von Regeln der Form $\omega_1 \varphi \omega_2 \rightarrow \omega_1 \psi \omega_2$
meist geschrieben als $\varphi \rightarrow \psi$

Ableitung

Relation "folgt":

Wenn G = (V_N, V_T, {S}, P), dann folgt ψ aus φ nach G genau dann, wenn es Ketten φ_1, φ_2, χ, ω gibt, so daß $\varphi = \varphi_1, \chi, \varphi_2$ und $\psi = \varphi_1, \omega, \varphi_2$ und $\chi \rightarrow \omega \in P$.

Notation: $! \#_G "$

Ableitung:

Eine Folge von Ketten $\varphi_1, \varphi_1, ..., \varphi_n$ ist eine Ableitung nach G g.d.w. $!_i \#_G !_{i+1}$

für alle i, $1 \leq i \leq n$.

Wenn es eine Ableitung nach G von φ nach ψ gibt, können wir das so schreiben: $! \#^*_G "$

Die Relation ableitbar ist transitiv und wird als reflexiv definiert.

Die erzeugte Sprache

Die Sprache L: Eine Kette ω ist nach G_L in der Sprache L g.d.w. die folgenden drei Bedingungen erfüllt sind:

1. $\omega \square V_{T^*}$
2. $S \overset{*}{\underset{G}{"}} !$
3. Es gibt kein χ, so daß $!\# \overset{*}{\underset{G}{"}}$ und $\omega \neq \chi$.

Man sagt auch G_L erzeugt die Sprache L. Die von G erzeugte Sprache L wird auch als L(G) geschrieben.

Schwache Äquivalenz: Zwei Grammatiken G_1 und G_2 sind schwach äquivalent, wenn sie dieselbe Sprache erzeugen.

Typen von Grammatiken

Typ 0 (unbeschränkte Ersetzungsregelsysteme):

Jede Grammatik, die die Definition einer formalen Grammatik erfüllt, ist vom Typ 0. Typ

1 (kontextsensitive Grammatiken):

Jede Produktion hat die Form $\varphi A \psi \rightarrow \varphi \omega \psi$, wobei $A \square V_T$, $\omega \neq \varepsilon$. Typ 2

(kontextfreie Grammatiken):

Jede Produktion hat die Form $A \rightarrow \omega$, wobei $\omega \neq \varepsilon$. Typ

3 (reguläre Grammatiken):

Jede Produktion hat die Form $A \rightarrow x\,B$ oder $A \rightarrow x$, wobei $x \neq \varepsilon$.

Formal grammars, formal languages and their corresponding automata

Chomsky hierarchy	Grammars	Languages	Minimal Automaton
Type-0	Unrestricted	Recursively enumerable	Turing machine
n/a	(no common name)	Recursive	Decider
Type-1	Context-sensitive	Context-sensitive	Linear-bounded Automaton
Type-2	Context-free	Context-free	Pushdown Automaton
Type-3	Regular	Regular	Finite-State Automaton

Eine kurze Chronologie der Kontroverse

- 57-70: Chomsky, Postal u.a. behaupten, das KFGs are unzureichend für natürliche Sprachen
- 63: Gilbert Harrman verteidigt Phrasenstrukturgrammatik
- 71&73: Peters und Ritchie zeigen, daß TG alle Sprachen vom Typ 0 erzeugt
- 79: Gazdar schlägt eine Zwei-Stufen PSG (GPSG) vor, die adäquat für das Englische ist
- 82: Gazdar & Pullum zeigen, daß alle bisherigen"Beweise" für die Nichtkontextfreiheit fehlerhaft sind
- 82: Uszkoreit und Peters beweisen, daß die damalige GPSG alle Sprachen vom Typ 0 erzeugt, dieser Umstand wird von GKPS durch den finite-closure constraint beseitigt
- 85: Shieber beweist, daß Zürcher Deutsch nicht KF ist, Culy zeigt das gleiche für Bambara
- 85: ähnliche Behauptungen von Higginbotham and Langendoen & Postal für das Englische wurden durch Pullum erfolgreich widerlegt

Jan säit das mer em Hans es huus hälfed aastriiche

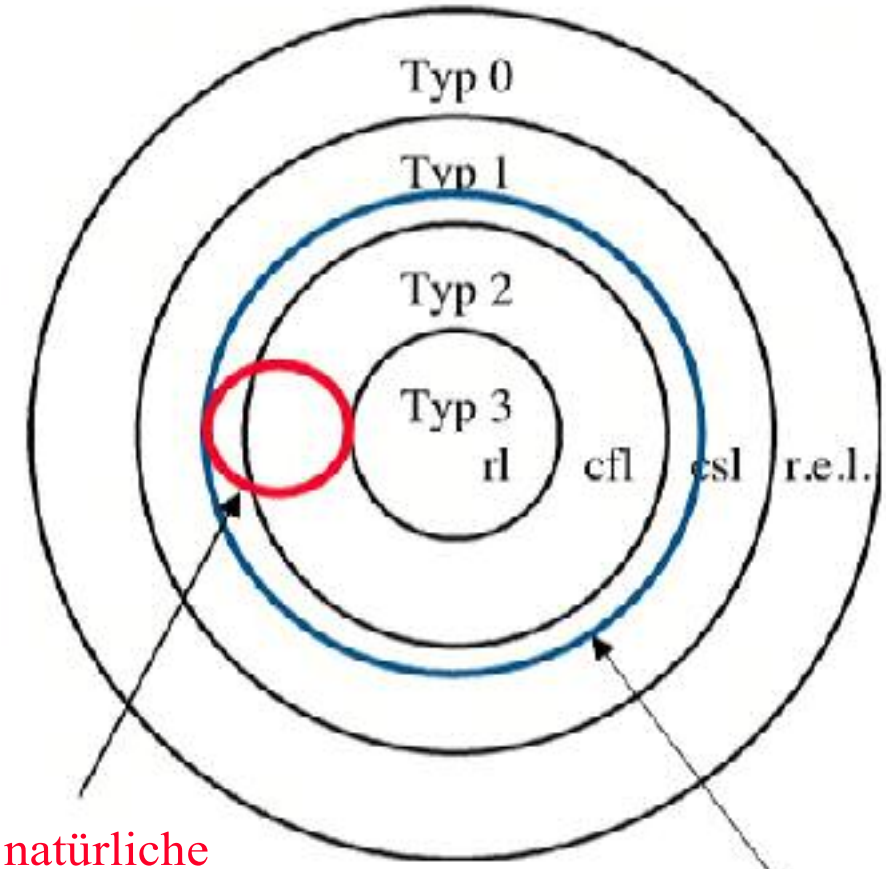
...dat Jan Piet Marie de kinderen zag helpen laten zwemmen

daß Jan Piet Marie den Kindern sah helfen lassen schwimmen

daß Jan Piet Marie helfen sah, die Kinder schwimmen zu lassen
daß Jan Piet Marie die Kinder schwimmen zu lassen helfen sah

Jan säit das mer em Hans es huus hälfed aastriiche

Computational Complexity of Human Language



- Typ 0: recursively enumerable sets
- Typ 1: contextsensitive languages
- Typ 2: context-free languages
- Typ 3: regular languages

Ableitung

S

NP VP

DET ADJ N VP

DET ADJ N V NP

DET ADJ N V DET ADJ N

ein kleines Mädchen sucht ein kleines Mädchen

Bäume

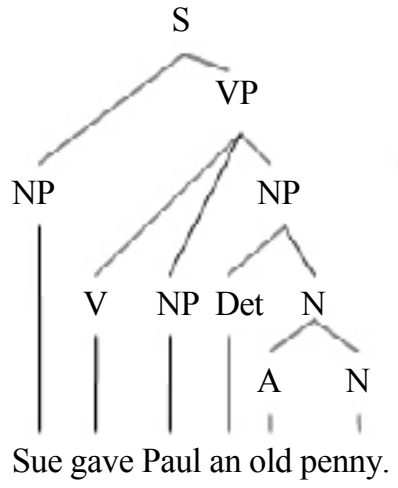
der Begriff des syntaktischen Strukturbaums

kodierte Information

1. Die hierarchische Organisation der Teile eines Satzes in Konstituenten.

2. Der Zugehörigkeit jeder Konstituente in eine syntaktische (linguistische) Klasse (Kategorie).

3. Die lineare Abfolge der Konstituenten.



Bäume II

Relationen: unmittelbare Dominanz - Dominanz
unmittelbare Präzedenz - Präzedenz

Konstituentenstrukturbaum: Quintupel (N, Q, D, P, L)

N - endliche Menge von Knoten
Q - endliche Menge von Etiketten
D - schwache Teilordnung in N x N, die Dominanzrelation
(reflexiv, transitiv und antisymmetrisch)
P - starke Teilordnung in N x N, die Präzedenzrelation
(irreflexiv, transitiv und asymmetrisch)
L - Funktion von N in Q, die Etikettierfunktion

Bedingungen:

Wurzelbedingung
Exklusivitätsbedingung
Kreuzungsfreiheit

Wurzelbedingung

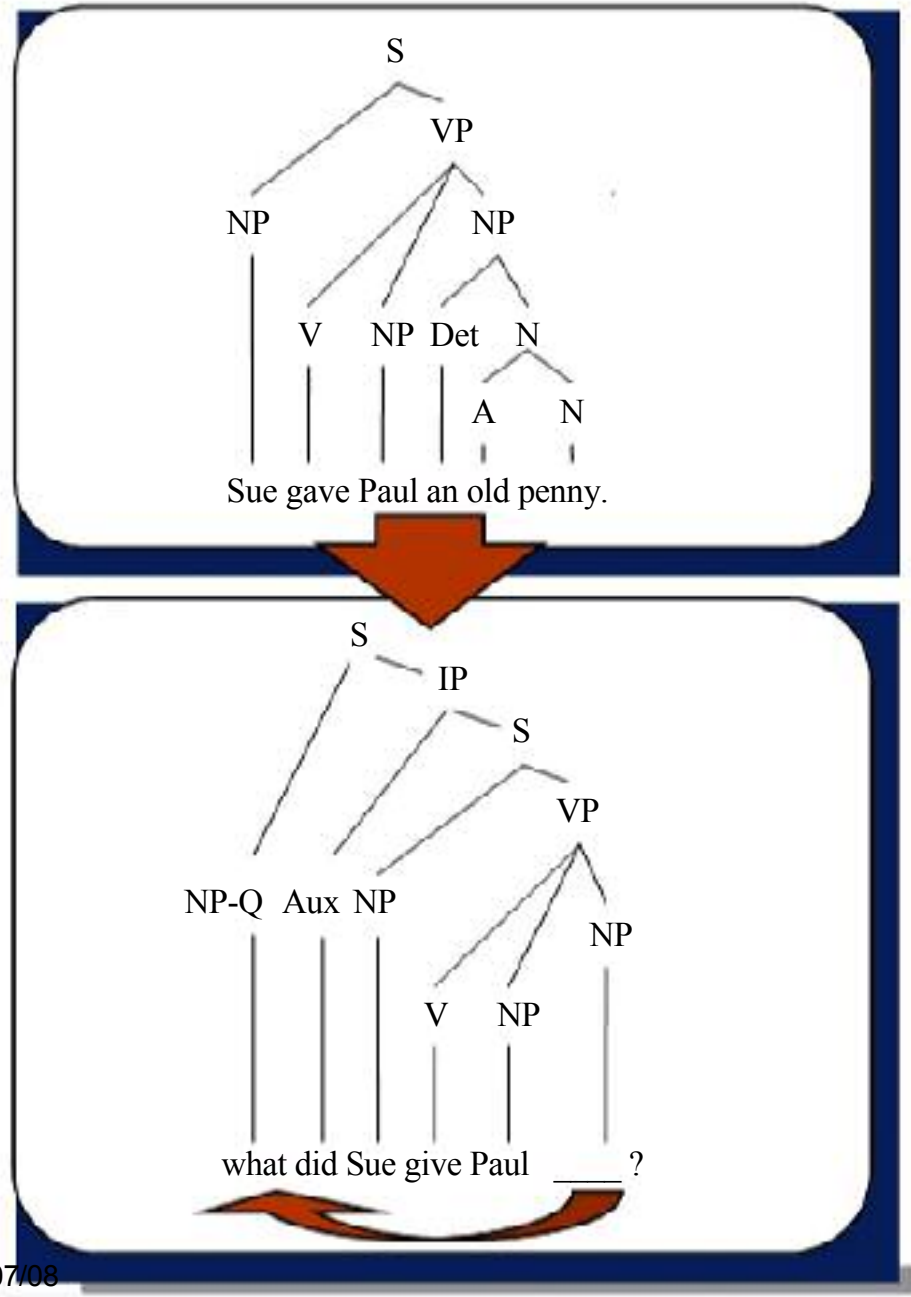
Es gibt genau einen Knoten, der alle Knoten des Baumes dominiert.

Exklusivitätsbedingung

Für jegliche zwei Knoten x und y gilt, entweder D(x,y) oder D(y,x)
oder P(x,y) oder P(y,x)

Kreuzungsfreiheit

Wenn P(x,y) dann gilt für alle x', die von x dominiert werden D(x,x') und für alle y', die von y dominiert werden D(y, y') dass auch x' y' vorausgeht [P(x', y')]



Transformations-grammatik

Grammatische Verarbeitung

- Je nach Verarbeitungsrichtung sprechen wir von

  grammatischer Analyse,
  wenn wir sprachliche Eingabe mithilfe einer Grammatik analysieren,
  oder von

  grammatischer Generierung,
  wenn wir mithilfe einer Grammatik aus eine Bedeutungsrepräsentation eine sprachliche Ausgabe erzeugen.

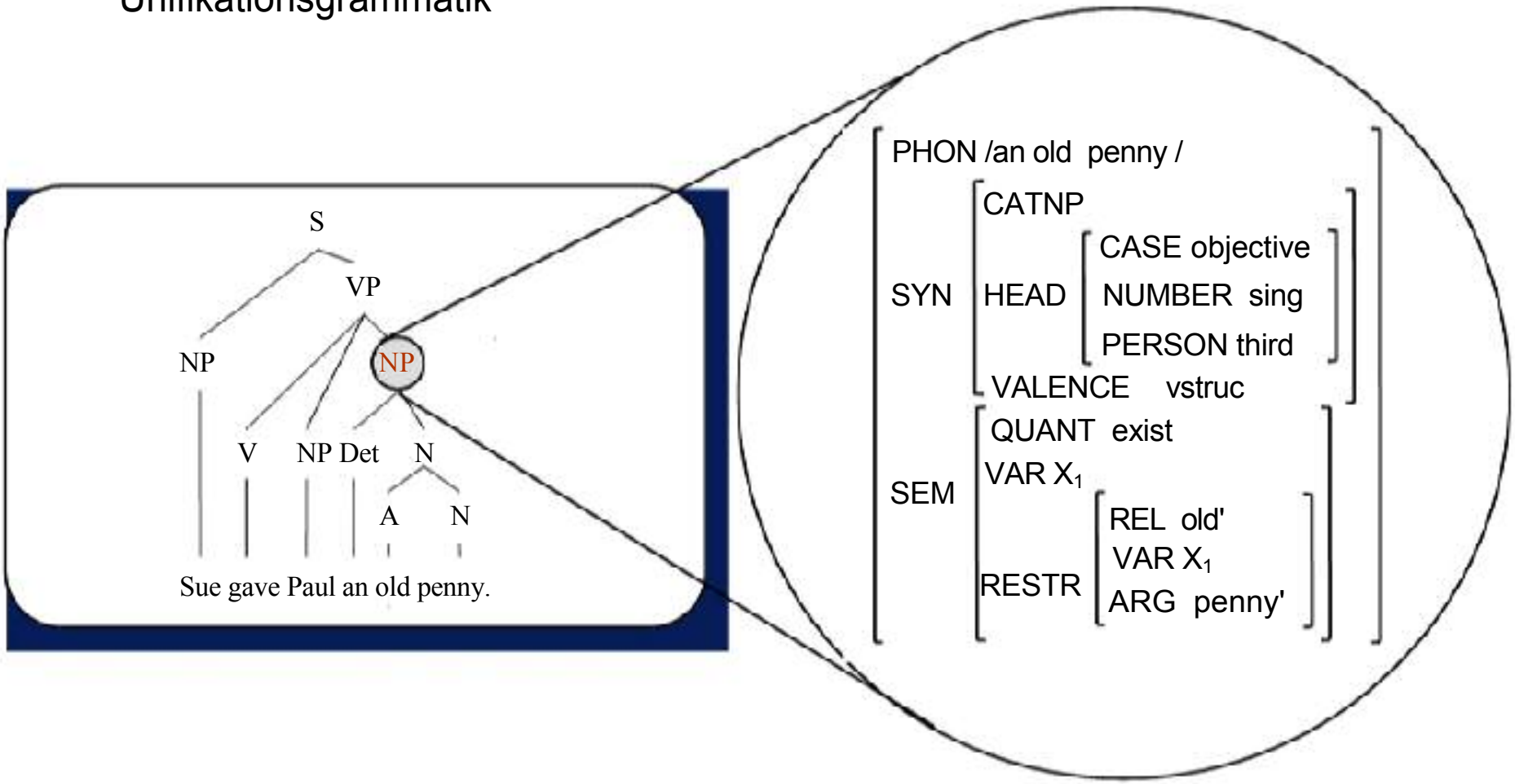
Warum syntaktische Analyse?

- Einen Hund hat dieser Mann gebissen.
- Ein Hund hat diesen Mann gebissen.

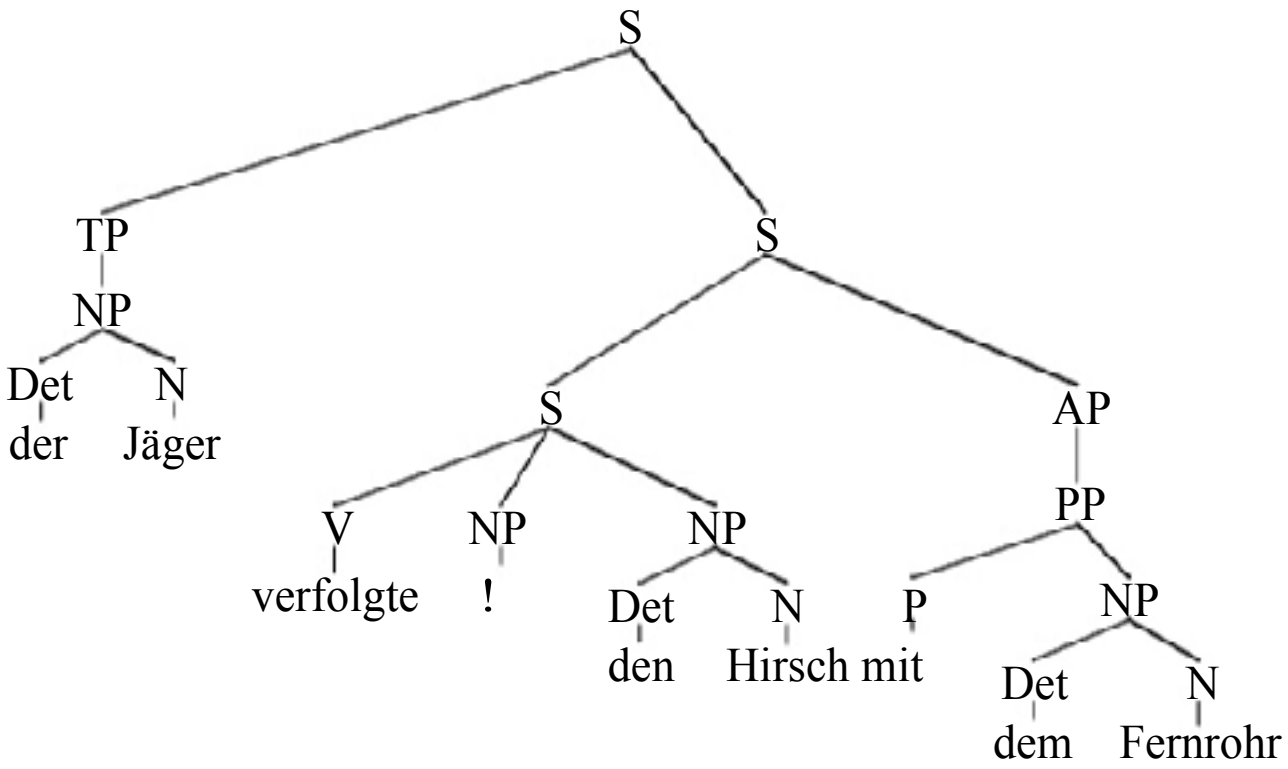
- This man has bitten a dog.
- A dog this man has bitten.
- A dog has bitten this man.

- Peter versprach Paul, die Akten zu bearbeiten.
- Peter überredete Paul, die Akten zu bearbeiten

Unifikationsgrammatik



Strukturbaum

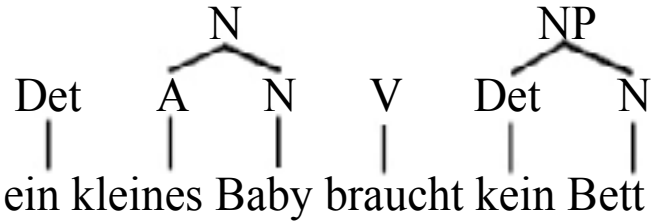


Bottom-Up Parsing

S ! NP VP	A ! kleines
NP ! Det N	A ! schönes
N ! A N	A ! warmes
N ! N PP	Det ! ein
VP ! V NP	Det ! kein
	Det ! sein
	N ! Baby
	N ! Bett
	N ! Kissen
	P ! ohne
	V ! braucht
	V ! hat

Bottom-Up Parsing

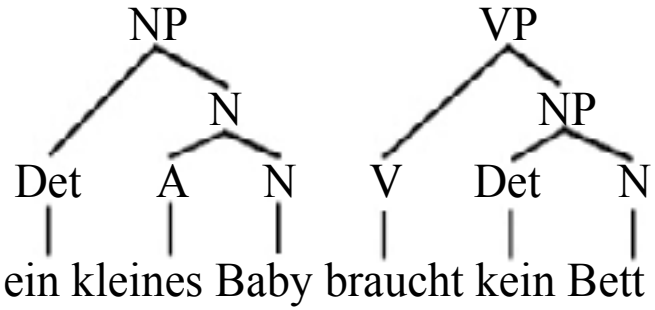
S ! NP VP	A ! kleines
NP ! Det N	A ! schönes
N ! A N	A ! warmes
N ! N PP	Det ! ein
VP ! V NP	Det ! kein
	Det ! sein
	N ! Baby
	N ! Bett
	N ! Kissen
	P ! ohne
	V ! braucht
	V ! hat



Bottom-Up Parsing

S ! NP VP
NP ! Det N
N ! A N
N ! N PP
VP ! V NP

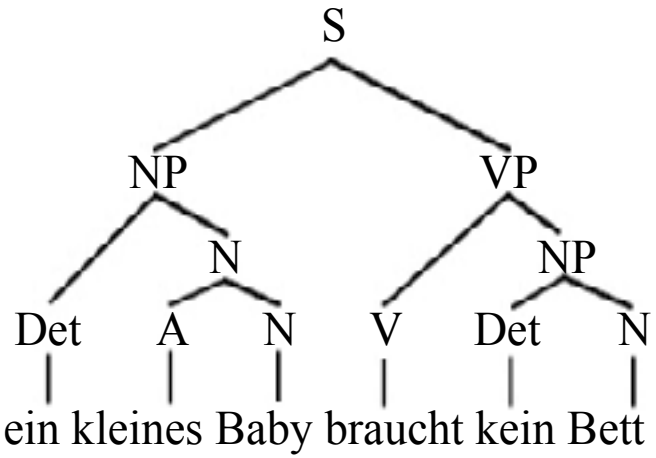
A ! kleines
A ! schönes
A ! warmes
Det ! ein
Det ! kein
Det ! sein
N ! Baby
N ! Bett
N ! Kissen
P ! ohne
V ! braucht
V ! hat



Bottom-Up Parsing

S ! NP VP
NP ! Det N
N ! A N
N ! N PP
VP ! V NP

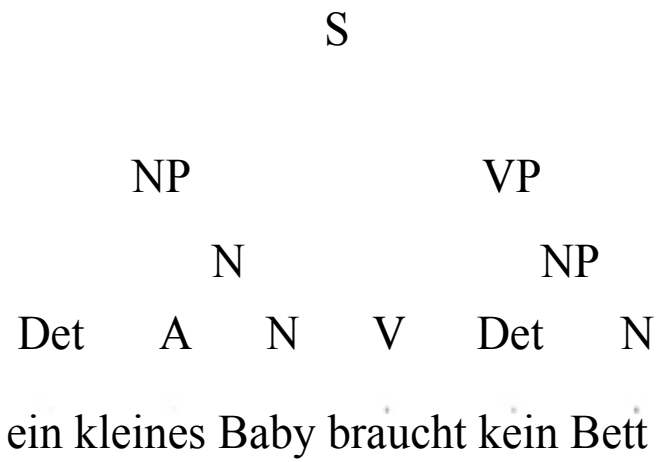
A ! kleines
A ! schönes
A ! warmes
Det ! ein
Det ! kein
Det ! sein
N ! Baby
N ! Bett
N ! Kissen
P ! ohne
V ! braucht
V ! hat



Top-Down Parsing

S ! NP VP
NP ! Det N
N ! A N
N ! N PP
VP ! V NP

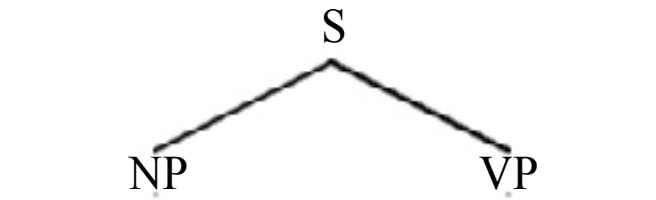
A ! kleines
A ! schönes
A ! warmes
Det ! ein
Det ! kein
Det ! sein
N ! Baby
N ! Bett
N ! Kissen
P ! ohne
V ! braucht
V ! hat



Top-Down Parsing

S ! NP VP
NP ! Det N
N ! A N
N ! N PP
VP ! V NP

A ! kleines
A ! schönes
A ! warmes
Det ! ein
Det ! kein
Det ! sein
N ! Baby
N ! Bett
N ! Kissen
P ! ohne
V ! braucht
V ! hat

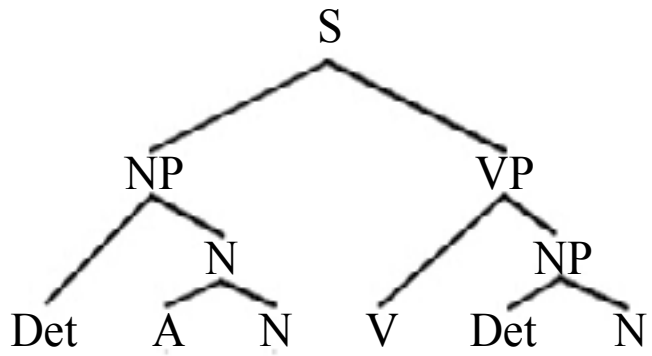


ein kleines Baby braucht kein Bett

Top-Down Parsing

S ! NP VP
NP ! Det N
N ! A N
N ! N PP
VP ! V NP

A ! kleines
A ! schönes
A ! warmes
Det ! ein
Det ! kein
Det ! sein
N ! Baby
N ! Bett
N ! Kissen
P ! ohne
V ! braucht
V ! hat

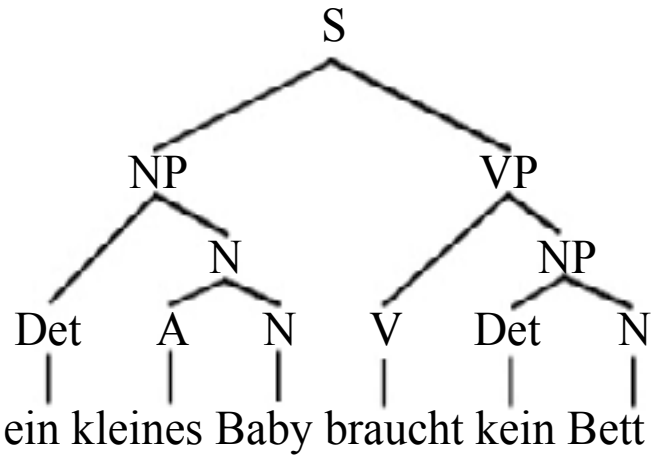


ein kleines Baby braucht kein Bett

Top-Down Parsing

S ! NP VP
NP ! Det N
N ! A N
N ! N PP
VP ! V NP

A ! kleines
A ! schönes
A ! warmes
Det ! ein
Det ! kein
Det ! sein
N ! Baby
N ! Bett
N ! Kissen
P ! ohne
V ! braucht
V ! hat



Beispiel

S → NP VP
NP → Det N
N → N PP
NP → PN
VP → V
VP → V NP
VP → V NP NP
PP → P NP

Det → der
N → Junge
PN → Berlin
P → aus
V → gab

Det → den
N → Ball
PN → Peter
P → für
V → sah

Det → das
N → Mädchen
PN → Gerda
P → ohne
V → schlief

PN → Bonn
P → nach
V → stahl

Beispiel

S → NP VP	Det → der	Det → den	Det → das	
NP → Det N	N → Junge	N → Ball	N → Mädchen	
N → N PP	PN → Berlin	PN → Peter	PN → Gerda	PN → Bonn
NP → PN	P → aus	P → für	P → ohne	P → nach
VP → V	V → gab	V → sah	V → schlief	V → stahl
VP → V NP				

Der Junge aus Berlin gab Peter den Ball.

Beispiel: Lokale Ambiguität

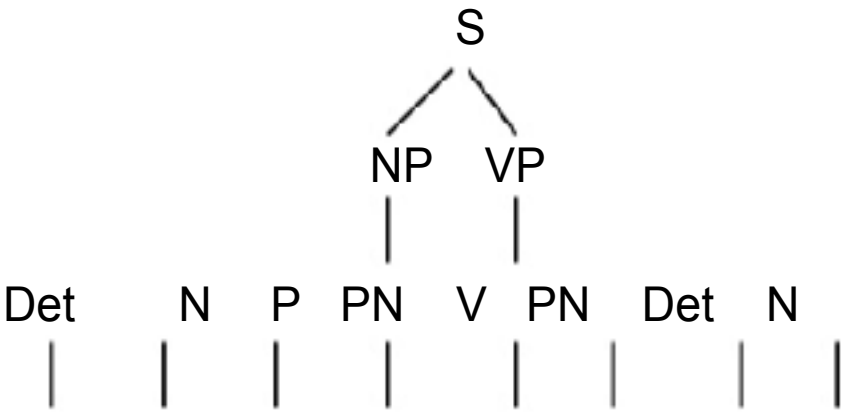
S → NP VP
NP → Det N
N → N PP
NP → PN
VP → V
VP → V NP
VP → V NP NP
PP → P NP

Det → der
N → Junge
PN → Berlin
P → aus
V → gab

Det → den
N → Ball
PN → Peter
P → für
V → sah

Det → das
N → Mädchen
PN → Gerda
P → ohne
V → schlief

PN → Bonn
P → nach
V → stahl



Der Junge aus Berlin gab Peter den Ball.

Earley Algorithmus

- 1970 für das Parsen mit kontextfreien Grammatiken von Jay Earley entwickelt.
- Der Algorithmus verknüpft top-down Vorhersage mit bottom-up Überprüfung.
- Als ein Chart-Parser verwendet der Algorithmus eine Tabelle mit erkannten wohlgeformten Teilketten (well-formed substring table).
- Jede wohlgeformte Zeichenkette wird nur einmal gemerkt.
- Der Algorithmus besteht aus den Teilen: (Initialisierung), Vorhersage, Vervollständigung und Einlesen.

Parsing: Chart Parsing, Earley-Algorithm,
top down/bottom up, left-right

Permanent data structure:

grammar (context free phrase structure grammar)

Imput: a string of words w_1 w_2 . . . w_3

Working structures:

Input string - input with position markers 0 w_1 1 w_2 2 ... w_n n

Position - point in the input string

Chart - set of items (edges) Every item is a triple !h, i, A ∀ #. ∃%, where h and i are positions of the input string, and A ! ∀. # a dotted rule. α and β are members of $(V_T ! V_N *.)$

A ! ∀ # is a rule of the grammar. α is the part of the right-hand-side of the rule that has already been applied to the segment of the input string from h to i.

Initially the chart contains for each rule of the form S ! ∀ exactly one item !0, 0, S ∀ . #∃. (initialization)

Algorithm:

For each position i from 0 to n repeat the following steps until no more items can be produced:

Scanner (consumption of input)

If w_i = a add for each item !h, i -1, A ∀ #.a ∃% a new item !h, i , A ∀ # a. ∃%.

Completer (completion of constituents)

For each pair of items of the form !h, i, A ∀ #. ∃ and !k , h, B ∀ # .A ∃ % add a new entry !k , i, B ∀ #A . ∃% to the chart if it is not already present in the chart.

Predictor (top down prediction of constituents)

For each item !h, i, A ∀ #.B ∃% add for each rule B ! ∀ a new item !i, i, B ∀ . #∃ to the chart if it is not already present in the chart.

If the chart contains at least one item !0, n , S ∀ #. ∃ return success, else return failure.

Parsing: Earley-Algorithm		example	

grammar: G_1

input: 0 the 1 little 2 baby 3 needs 4 a 5 bed 6

i	word$_i$	new items	step
0	-	• <0,0, S → .NP VP>	Initial.
		• <0,0, NP → .Det N>	Predictor
		• <0,0, Det → .the>	Predictor
		<0,0, Det → .a>	Predictor
		<0,0, Det → .his>	Predictor
1	the	• <0,1, Det → the.>	Scanner
		• <0,1, NP → Det.N>	Completer
		• <1,1, N → .A N>	Predictor
		• <1,1, A → .little>	Predictor
		<1,1, A → .nice>	Predictor
		<1,1, A → .warm>	Predictor
		<1,1, N → .baby>	Predictor
		<1,1, N → .bed>	Predictor
		<1,1, N → .pillow>	Predictor
2	little	• <1,2, A → little.>	Scanner
		• <1,2, N → A.N>	Completer
		• <2,2, N → .A N>	Predictor
		• <2,2, A → .little>	Predictor
		<2,2, A → .nice>	Predictor
		<2,2, A → .warm>	Predictor
		• <2,2, N → .baby>	Predictor
		<2,2, N → .bed>	Predictor
		<2,2, N → .pillow>	Predictor

i	wordi	new items	step
3	baby	• <2,3, N → baby.>	Scanner
		• <1,3, N → A N.>	Completer
		• <0,3, NP → Det N.>	Completer
		• <0,3, S → NP.VP>	Completer
		• <3,3, VP → .V NP>	Predictor
		<3,3, VP → .V NP PP>	Predictor
		• <3,3, V → .needs	Predictor
		<3,3, V → .has	Predictor
		<3,3, V → .sees	Predictor
4	needs	• <3,4, V → needs.>	Scanner
		• <3,4, VP → V.NP>	Completer
		<3,4, VP → V.NP PP>	Completer
		• <4,4, NP → .Det N>	Predictor
		<4,4, Det → .the>	Predictor
		• <4,4, Det → .a>	Predictor
		<4,4, Det → .his>	Predictor
5	a	• <4,5, Det → a.>	Scanner
		• <4,5, NP → Det.N>	Completer
		<5,5, N → .A N>	Predictor
		<5,5, A → .little>	Predictor
		<5,5, A → .nice>	Predictor
		<5,5, A → .warm>	Predictor
		<5,5, N → .baby>	Predictor
		• <5,5, N → .bed>	Predictor
		<5,5, N → .pillow>	Predictor
6	bed	• <5,6, N → bed.>	Scanner
		• <4,6, NP → Det N.>	Completer
		• <3,6, VP → V NP.>	Completer
		<3,6, VP → V NP.PP>	Completer
		• <0,6, S → NP VP.>	Completer
		<6,6, PP → .P NP>	Predictor
		<6,6, P → .without>	Predictor
		<6,6, P → .for>	Predictor

Initialisierung

Anfangs enthält die Chart für jede Regel S → α genau einen Eintrag 〈0, 0, S →. α〉 .

Position = 0

Für S → NP VP enthält die Chart den folgenden ersten Eintrag: 〈0, 0, S → .NP VP〉

S

⓪ Der ① Junge ② aus ③ Berlin ④ gab ⑤ Peter ⑥ den ⑦ Ball ⑧

Predictor

Vorhersage (Predictor)

Für jeden Eintrag !h, i, A " #.B $% füge für jede Regel

B " & den Eintrag !i, i, B " .&% in die Chart ein, wenn er nicht schon in der Chart steht.

Position = 0

Für 〈0, 0, S → .NP VP〉 und NP → Det N füge ein: 〈0, 0, NP → .Det N〉

S

⓪ Der ① Junge ② aus ③ Berlin ④ gab ⑤ Peter ⑥ den ⑦ Ball ⑧

NP

Predictor

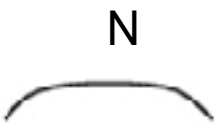
Vorhersage (Predictor)

Für jeden Eintrag !h, i, A " #.B $% füge für jede Regel

B " & den Eintrag !i, i, B " .&% in die Chart ein, wenn er nicht schon in der Chart steht.

Position = 2

Für 〈1, 2, N → N.PP〉 und PP → P NP füge ein: 〈2, 2, PP → .P NP〉



⓪ Der ① Junge ② aus ③ Berlin ④ gab ⑤ Peter ⑥ den ⑦ Ball ⑧

Predictor

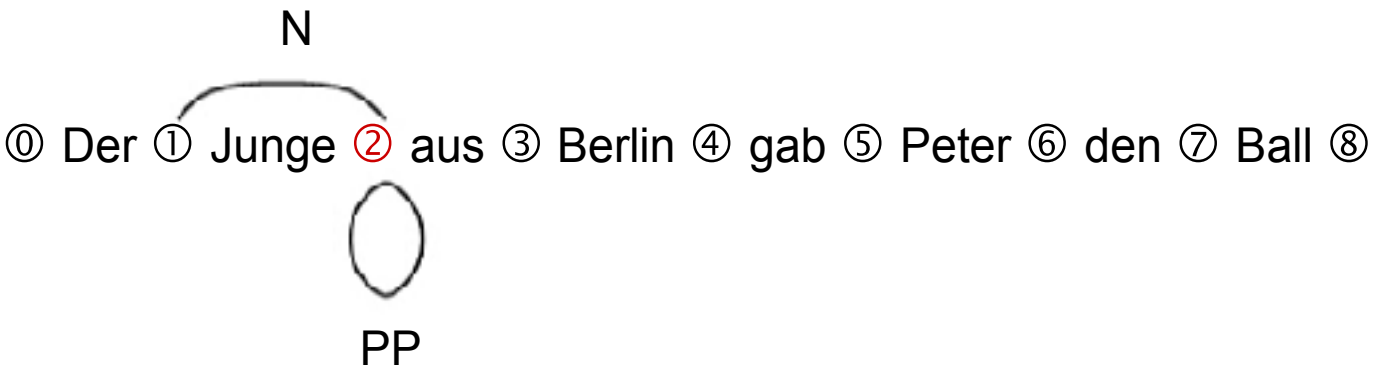
Vorhersage (Predictor)

Für jeden Eintrag !h, i, A " #.B $% füge für jede Regel

B " & den Eintrag !i, i, B " .&% in die Chart ein, wenn er nicht schon in der Chart steht.

Position = 2

Für 〈1, 2, N → N.PP〉 und PP → P NP füge ein: 〈2, 2, PP → .P NP〉



Predictor

Vorhersage (Predictor)

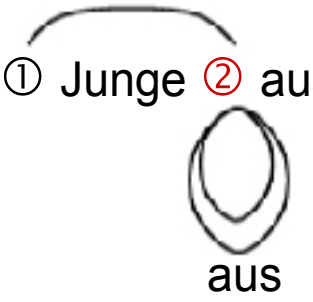
Für jeden Eintrag !h, i, A " #.B $% füge für jede Regel

B " & den Eintrag !i, i, B " .&% in die Chart ein, wenn er nicht schon in der Chart steht.

Position = 2

Für 〈1, 2, N → N.PP〉 und PP → P NP füge ein: 〈2, 2, PP → .P NP〉 Für

〈2, 2, PP → .P NP〉 und P → aus füge ein: 〈2, 2, P → .aus〉

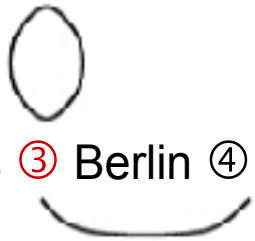


Eingabeverarbeitung (Scanner)

Wenn wi = a füge für jeden Eintrag !h, i -1, A " #.a $%
einen Eintrag !h, i , A " # a. $%.

Position = 3

Für w_i = Berlin und 〈3, 3, PN → .Berlin〉 füge ein: 〈3, 4, PN → Berlin.〉



⓪ Der ① Junge ② aus ③ Berlin ④ gab ⑤ Peter ⑥ den ⑦ Ball ⑧

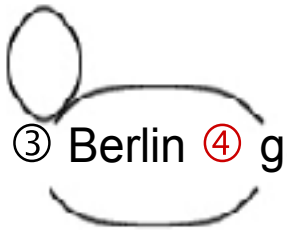
Completer

Vervollständigung (Completer)

Für jedes Paar von Einträgen der Form !h, i, A " #. $ und
!k , h, B " % .A & $ füge den Eintrag !k , i, B " %A .&$ in
die Chart ein, wenn er nicht schon in der Chart steht.

Position = 4

Für 〈3, 4, PN → Berlin.〉 und 〈3, 3, NP → .PN〉 füge ein: 〈3, 4, NP → PN.〉



⓪ Der ① Junge ② aus ③ Berlin ④ gab ⑤ Peter ⑥ den ⑦ Ball ⑧

Completer

Vervollständigung (Completer)

Für jedes Paar von Einträgen der Form !h, i, A " #. $ und
!k , h, B " % .A & $ füge den Eintrag !k , i, B " %A .&$ in
die Chart ein, wenn er nicht schon in der Chart steht.

Position = 4

Für 〈3, 4, PN → Berlin.〉 und 〈3, 3, NP → .PN〉 füge ein: 〈3, 4, NP → PN.〉

Für 〈3, 4, NP → PN.〉 und 〈2, 3, PP → P .NP〉 füge ein: 〈2, 4, PP → P NP.〉

⓪ Der ① Junge ② aus ③ Berlin ④ gab ⑤ Peter ⑥ den ⑦ Ball ⑧