Dependenzparsing

Vorlesung “Computerlinguistische Techniken”

Alexander Koller

15. Dezember 2014

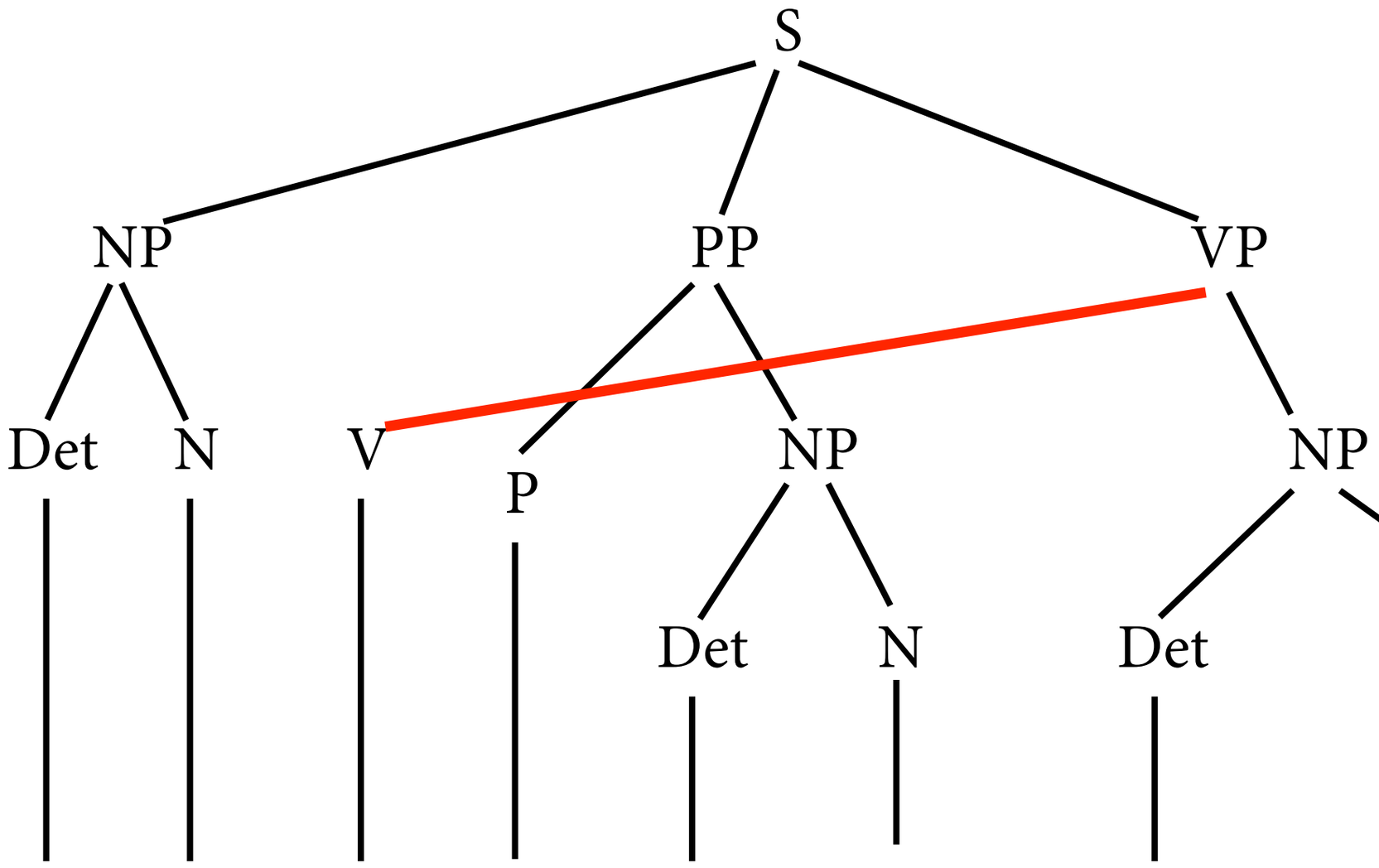
Was bisher geschah

- Verschiedene Parsingalgorithmen für kfGs
- Statistische Modelle:
  - n-Gramm-Modelle
  - HMMs
- Nächster Schritt: statistisches Parsing.
  - heute: Dependenzparsing
  - Januar: Probabilistische kfGs

Diskontinuität

- Versteckte Annahme von kfGs:
  - jede Konstituente erzeugt einen zusammenhängenden Substring
  - Kind eines Knotens im Parsebaum darf nicht in andere Teilbäume “hineinragen”
- Annahme ist für Englisch weitgehend erfüllt.
  Für Sprachen mit freier Wortstellung (z.B. Deutsch) aber nicht!

Diskontinuität: Ein Beispiel



Konsequenzen

- Jeder rein kfG-basierte Parser wird für Deutsch Fehler machen, weil er die richtigen Analysen nicht darstellen kann.

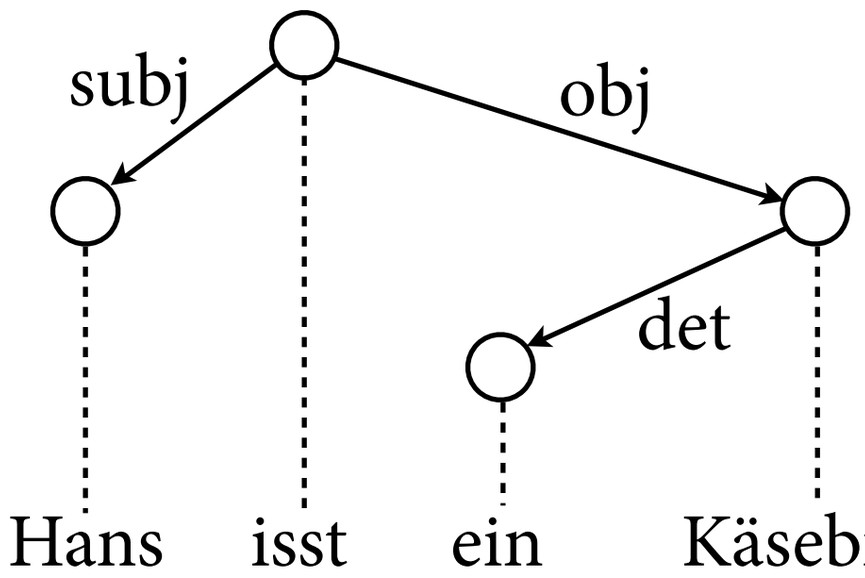
- Auswege:
  - Mechanismen für freie Wortstellung in kfGs einbauen (z.B. UCFGs). Hierfür kein W.modell bekannt; Parsingkomplexität steigt.
  - Anderen Grammatikformalismus auswählen!

Dependenzbäume

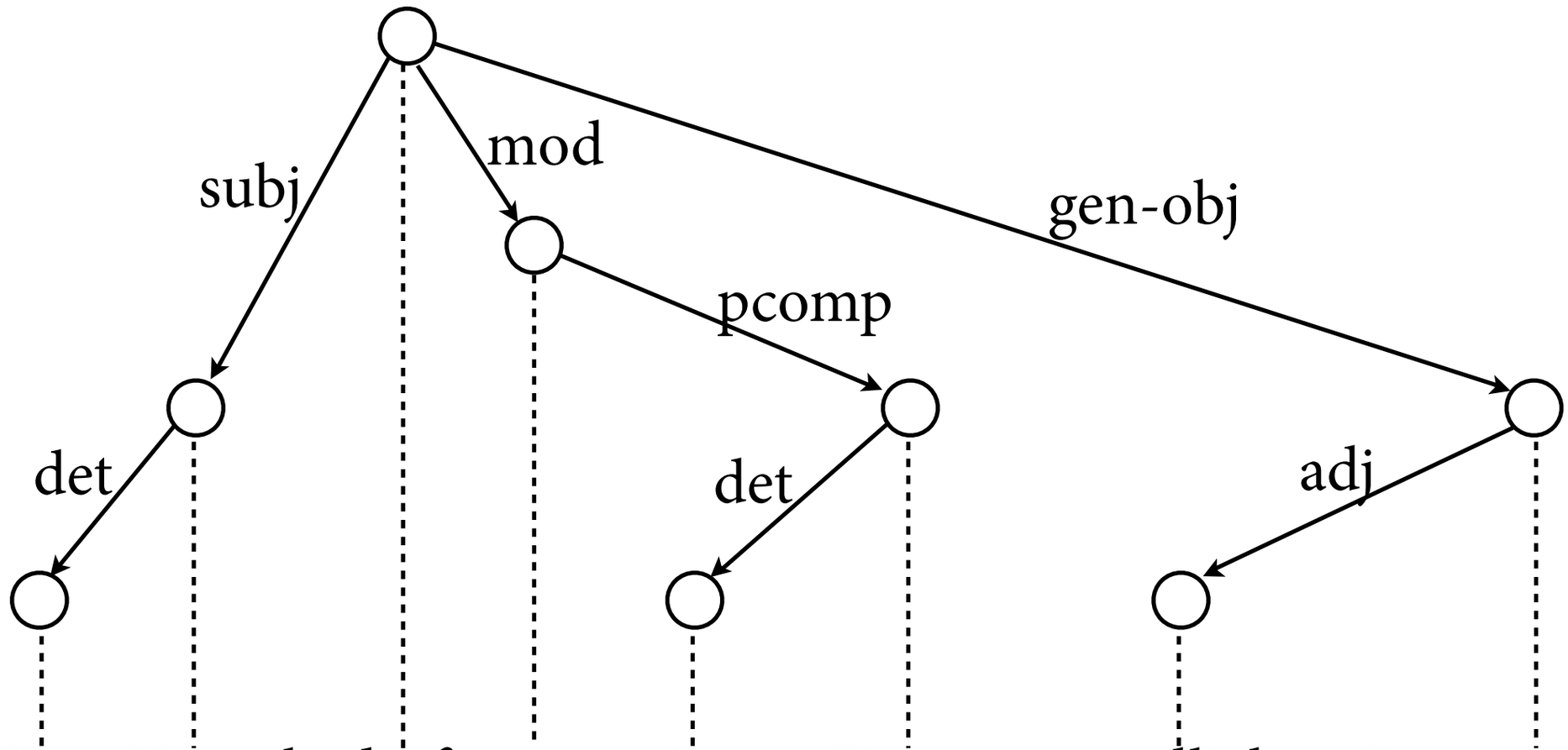
- Grundidee:
  - Grammatische Struktur nicht durch Konstituenten darstellen, sondern durch Abhängigkeiten zwischen Wörtern.
  - Knoten eines Dependenzbaums sind Wörter, Kanten sind Dependenzen.
  - Lexikon gibt Valenzen der Wörter an: welche Wörter will ich als Dependenten?

- Historisch alter Grammatikansatz:
  - Tesniere, posthum 1953
  - Prager Schule während des Kalten Kriegs
  - heute in CL sehr aktuell

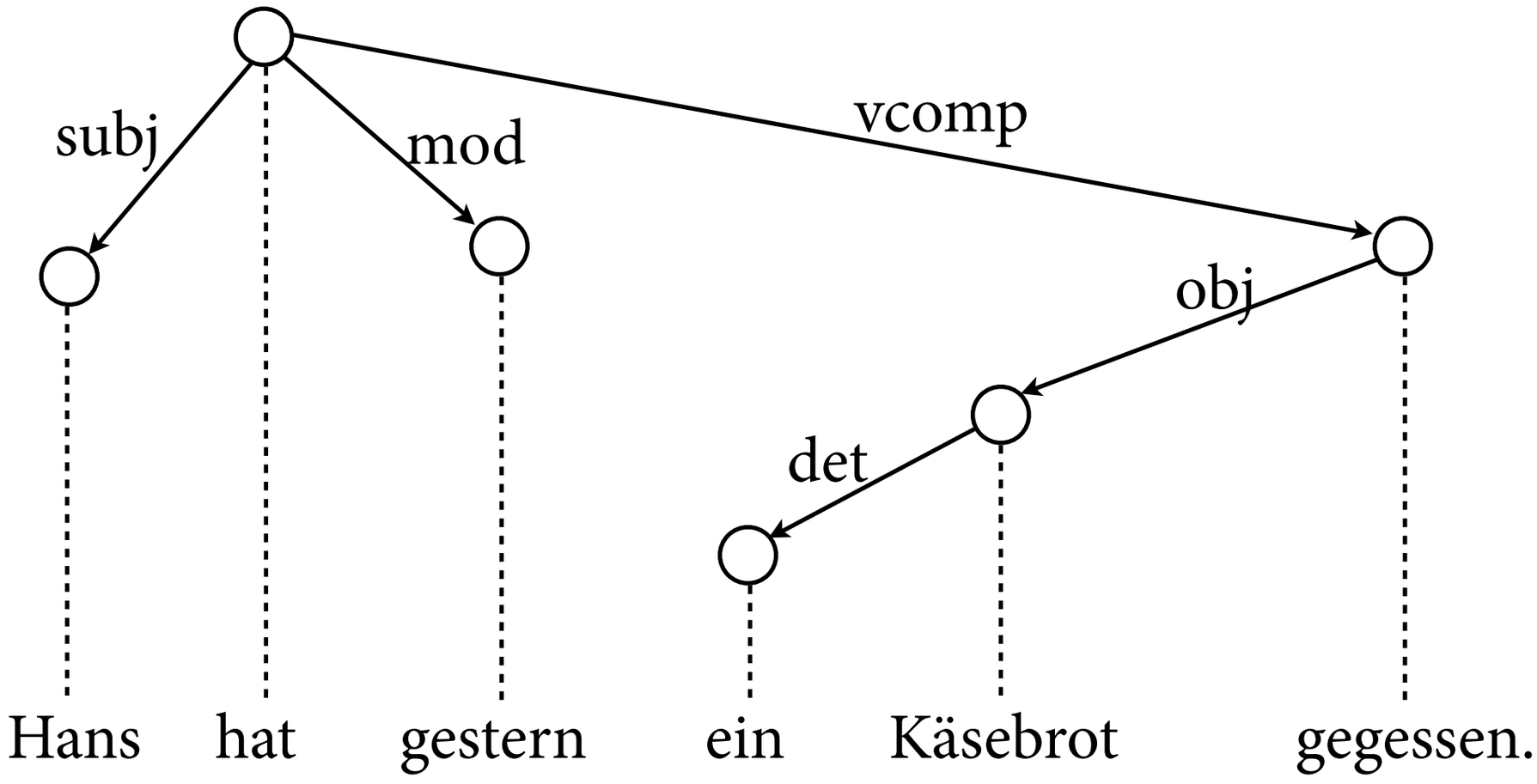
Dependenzbäume: Beispiele



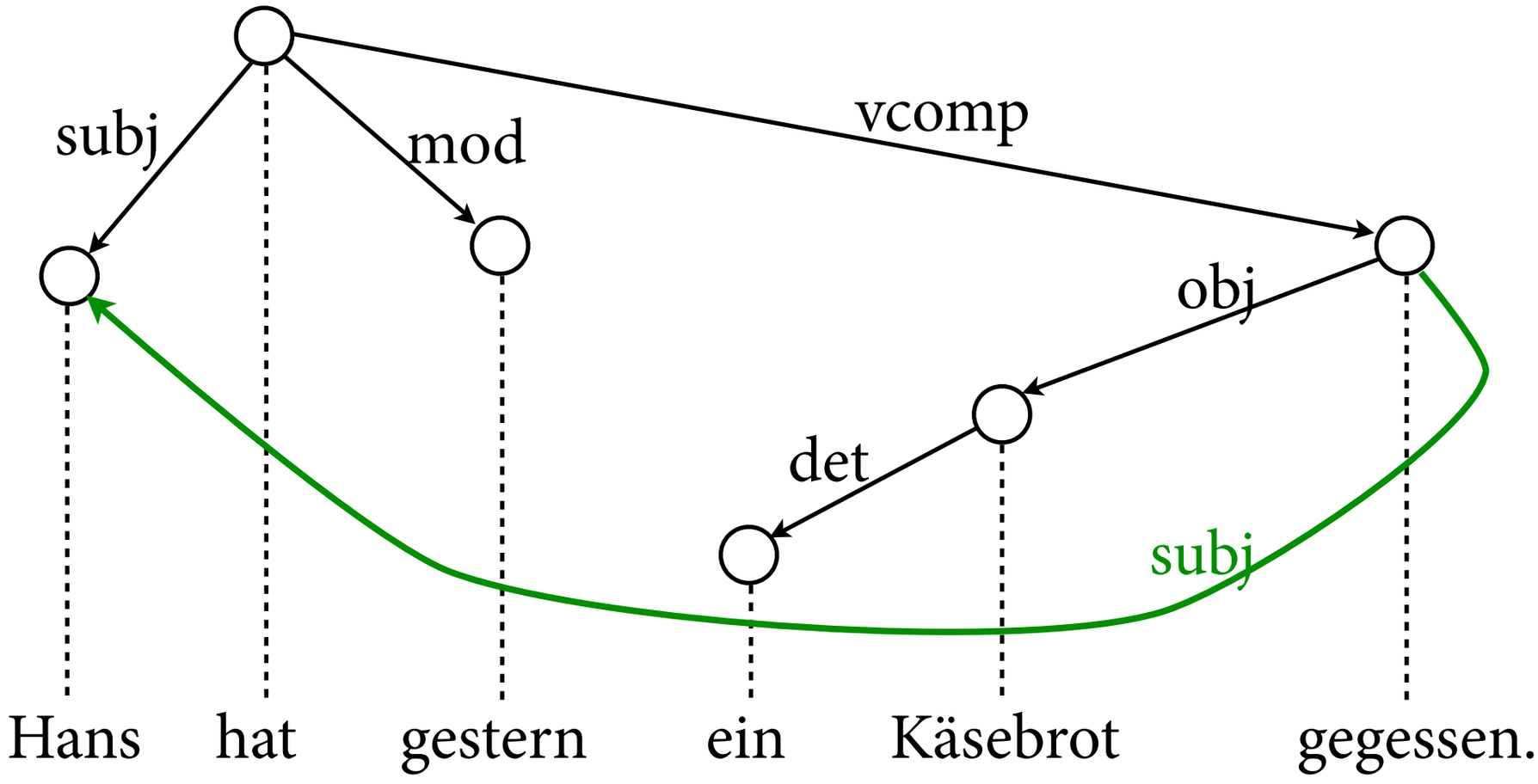
Dependenzbäume: Beispiele



Dependenzbäume: Beispiele

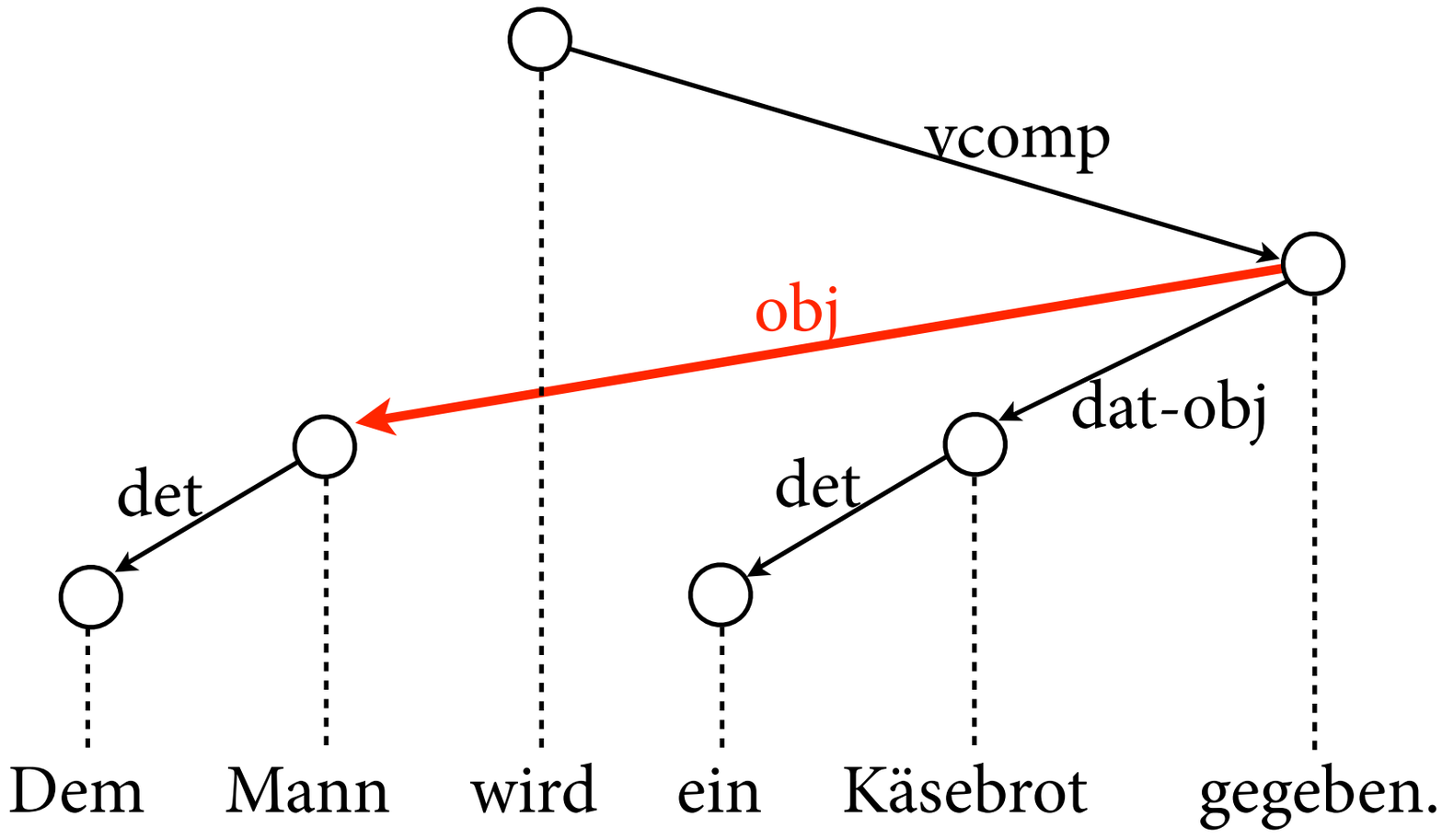


Dependenzbäume: Beispiele



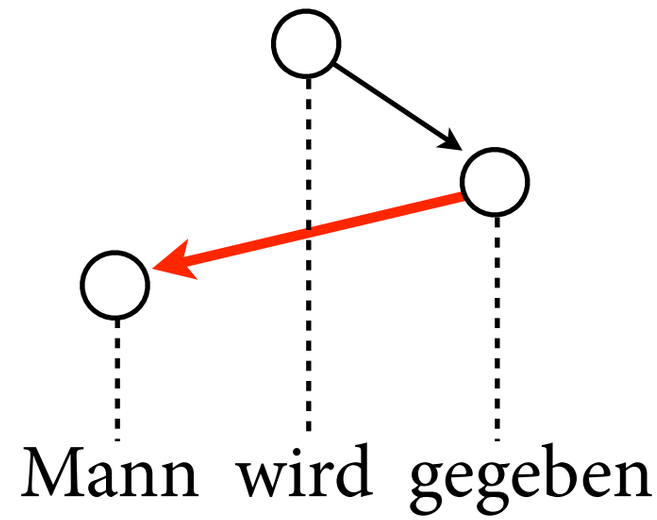
Manche Dependenztheorien erlauben Graphen, die keine Bäume sind.

Dependenzbäume: Beispiele



Kreuzende Kanten

- Dependenzbaum kann *kreuzende Kanten* enthalten: kreuzen die *Projektionslinie* eines anderen Wortes.



- Kreuzende Kanten sind (erlaubtes!) Äquivalent zu diskont. Konstituenten.

Projektivität

- Dependenzbäume ohne kreuzende Kanten heißen *projektiv*; ansonsten nichtprojektiv.

- Nichtprojektive Dependenzbäume sind erlaubt, aber projektive sind einfacher, und Parsing wird einfacher.

- Formales Resultat (Hays-Gaifman):
  Projektive Dependenzbäume und kontextfreie Parsebäume ineinander übersetzbar.

Dependenzparsing

- Problem: Wie findet man für einen Satz
  - (gegeben eine Dependenzgrammatik) grammatische Korrektheit? (= Wortproblem)
  - (gegeben eine DG) alle korrekten Dependenzbäume? (= Parsingproblem)
  - den “besten” Dependenzbaum?

- Man kann Dependenzgrammatiken definieren. Wir gehen hier aber einen anderen Weg.

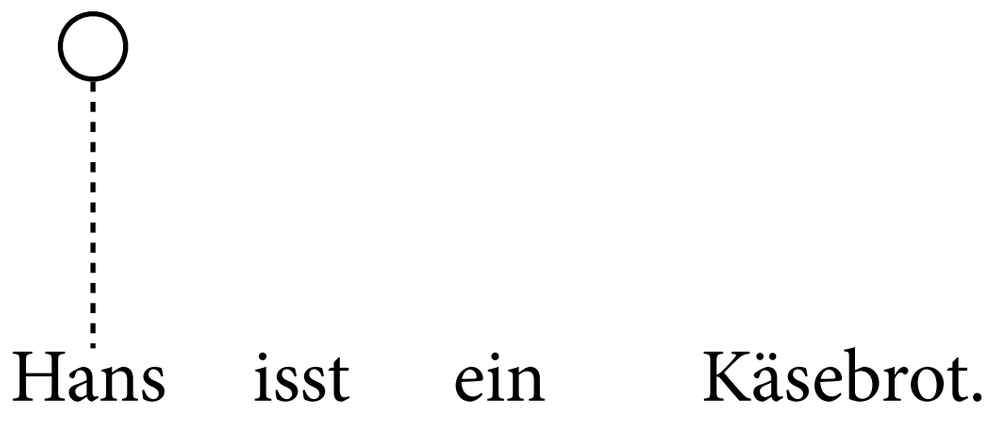
Parsing ohne Grammatik

- Angenommen, wir hätten ein Orakel, das uns für jedes Paar von Wörtern sagt, ob das eine Kind des anderen sein möchte.
  - Dann können wir ohne formale Grammatik “richtige” Dependenzbäume ausrechnen.

Hans isst ein Käsebrot.

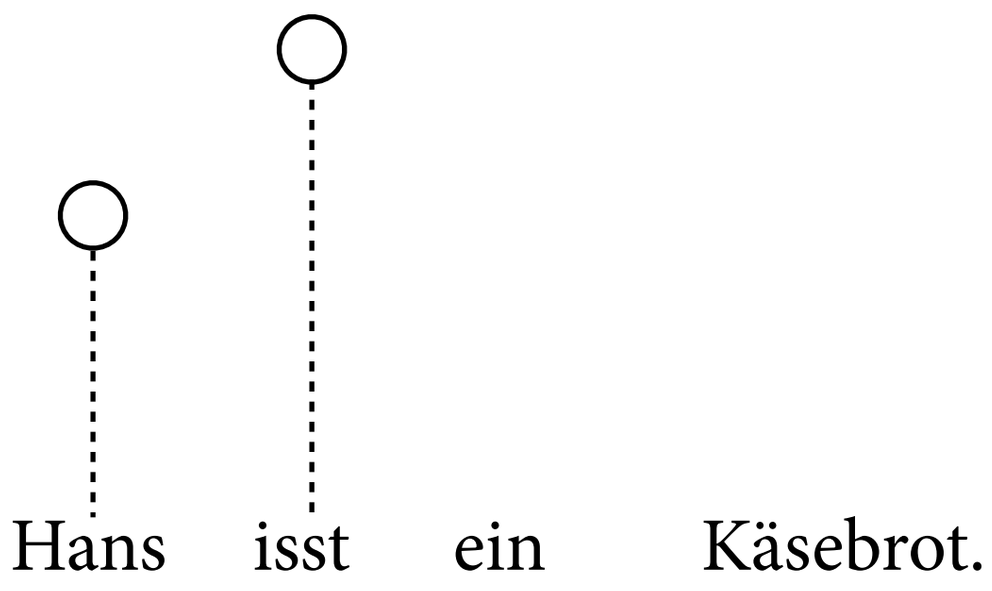
Parsing ohne Grammatik

- Angenommen, wir hätten ein Orakel, das uns für jedes Paar von Wörtern sagt, ob das eine Kind des anderen sein möchte.
  - Dann können wir ohne formale Grammatik “richtige” Dependenzbäume ausrechnen.



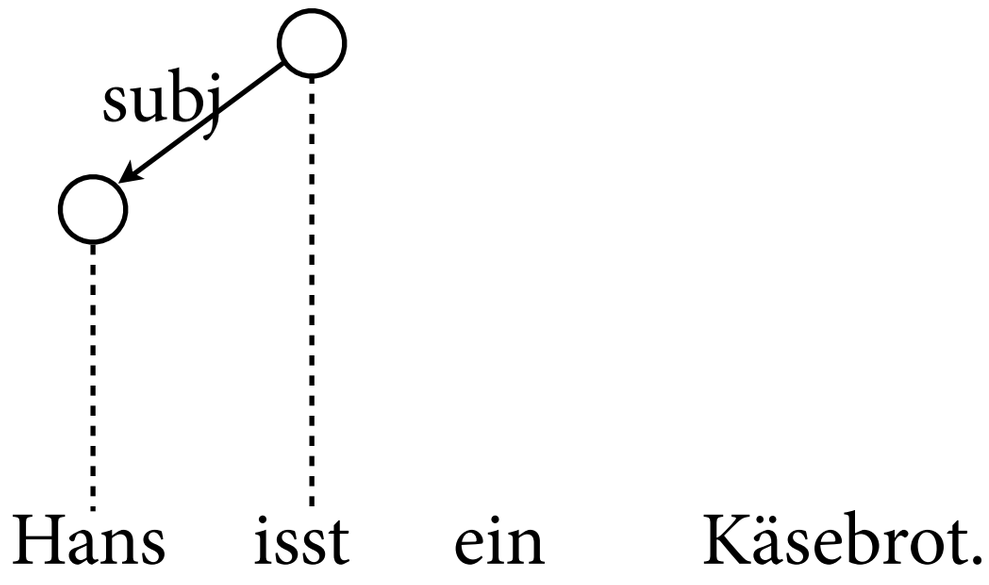
Parsing ohne Grammatik

- Angenommen, wir hätten ein Orakel, das uns für jedes Paar von Wörtern sagt, ob das eine Kind des anderen sein möchte.
  - Dann können wir ohne formale Grammatik “richtige” Dependenzbäume ausrechnen.



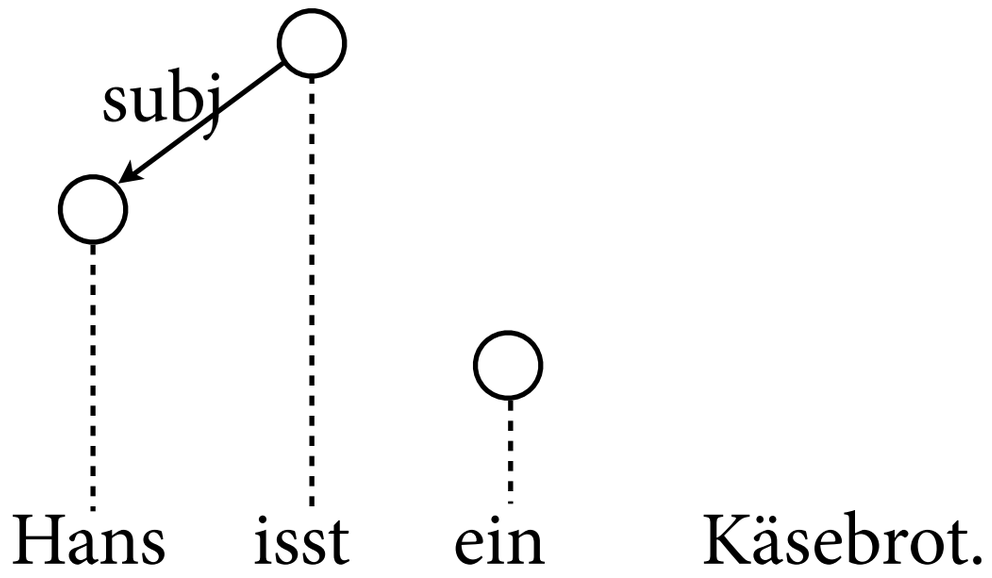
Parsing ohne Grammatik

- Angenommen, wir hätten ein Orakel, das uns für jedes Paar von Wörtern sagt, ob das eine Kind des anderen sein möchte.
  - Dann können wir ohne formale Grammatik “richtige” Dependenzbäume ausrechnen.



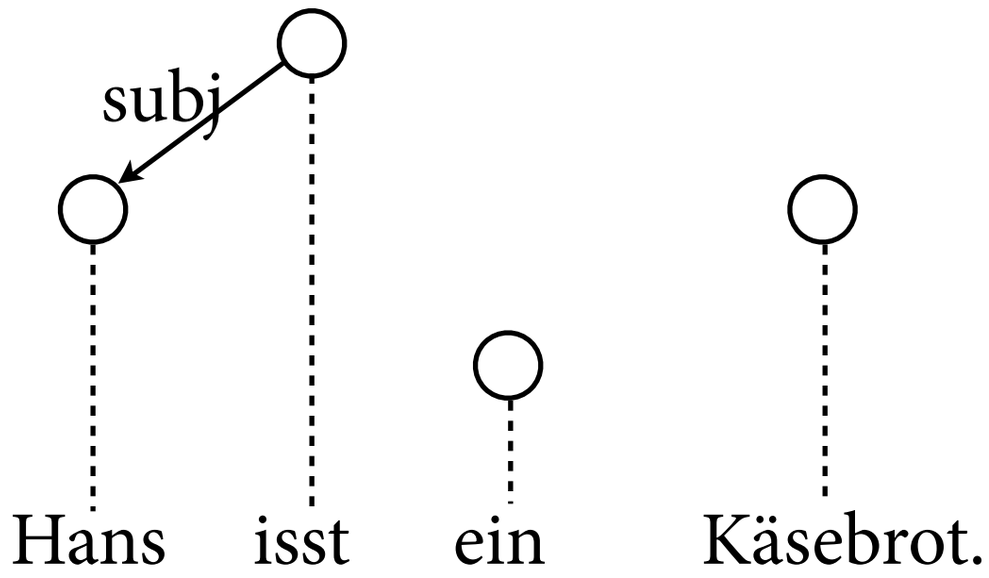
Parsing ohne Grammatik

- Angenommen, wir hätten ein Orakel, das uns für jedes Paar von Wörtern sagt, ob das eine Kind des anderen sein möchte.
  - Dann können wir ohne formale Grammatik “richtige” Dependenzbäume ausrechnen.



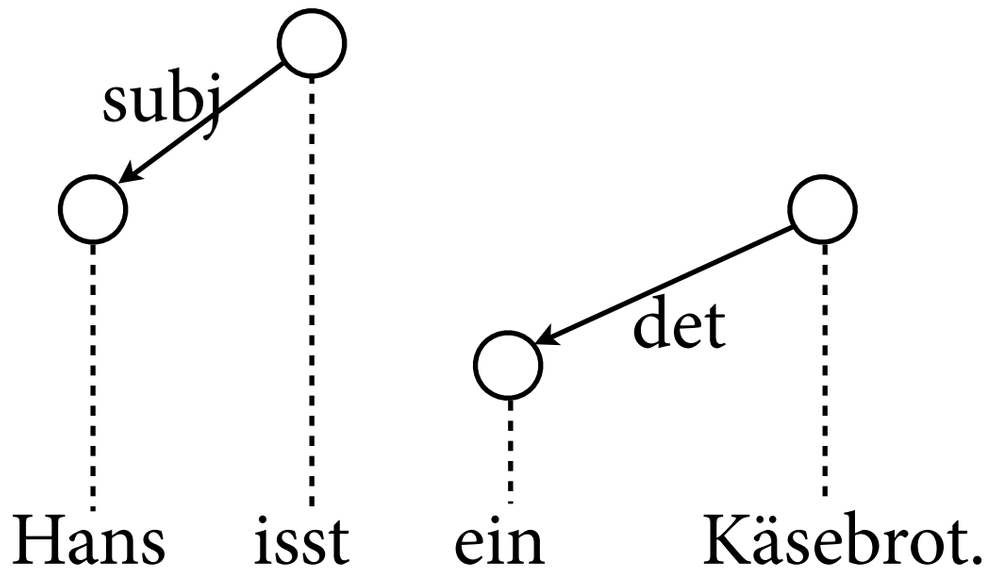
Parsing ohne Grammatik

- Angenommen, wir hätten ein Orakel, das uns für jedes Paar von Wörtern sagt, ob das eine Kind des anderen sein möchte.
  - Dann können wir ohne formale Grammatik “richtige” Dependenzbäume ausrechnen.



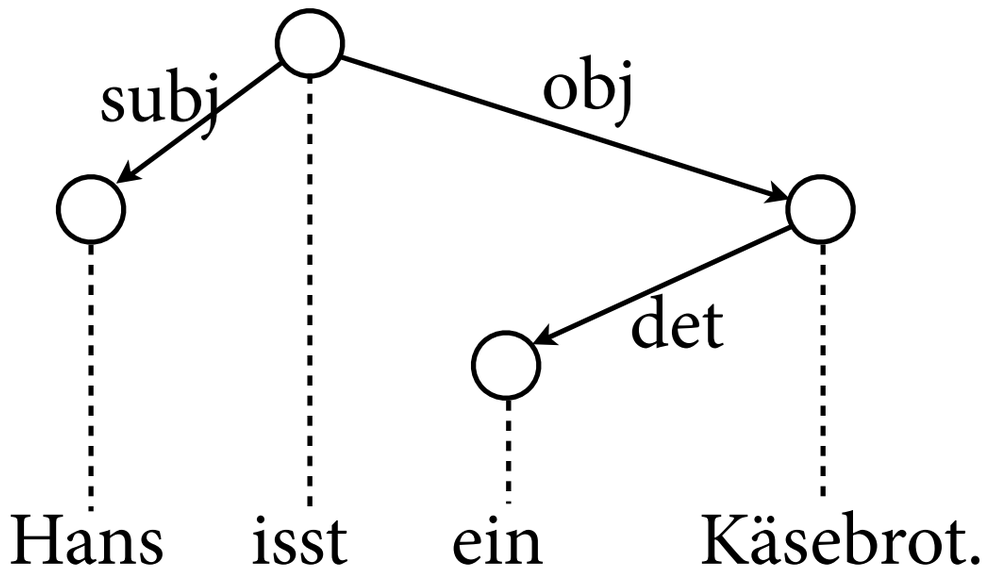
Parsing ohne Grammatik

- Angenommen, wir hätten ein Orakel, das uns für jedes Paar von Wörtern sagt, ob das eine Kind des anderen sein möchte.
  - Dann können wir ohne formale Grammatik “richtige” Dependenzbäume ausrechnen.



Parsing ohne Grammatik

- Angenommen, wir hätten ein Orakel, das uns für jedes Paar von Wörtern sagt, ob das eine Kind des anderen sein möchte.
  - Dann können wir ohne formale Grammatik “richtige” Dependenzbäume ausrechnen.



Nivres Dependenzparser

- Idee von Joakim Nivre (2003):
  - Parser liest Satz von links nach rechts und führt dabei Parseroperationen aus ⇒ massiv nichtdeterministisch.
  - In jedem Schritt fragt man *Klassifikator*, welche Operation ausgeführt werden soll, auf Grundlage von *Features* ⇒ deterministischer Parser.

- In Urform gehen nur projektive Bäume.
  - Erweiterung auf “pseudo-projektives Parsing”
  - Alternative: MST-Parser, volle Nichtprojektivität

Parser-Items

- Nivres Parser manipuliert für Satz w_1 ... w_n Items der Form (σ, τ, h, d):
  - τ ist sortierte Liste der Zahlen j, ..., n für ein j
    = noch nicht gelesene Eingabewörter
  - σ ist ein Stack, der Zahlen < j enthält
    = Wurzeln von Teilbäumen, die wir schon gelesen haben
  - h(i) ist Vater des i-ten Worts; falls wir noch keinen Vater von i kennen, ist h(i) = 0 (“Kind der Wurzel”)
  - d(i) ist Label der Kante (h(i), i); falls h(i) = 0, ist d(i) = r0

Parser-Items

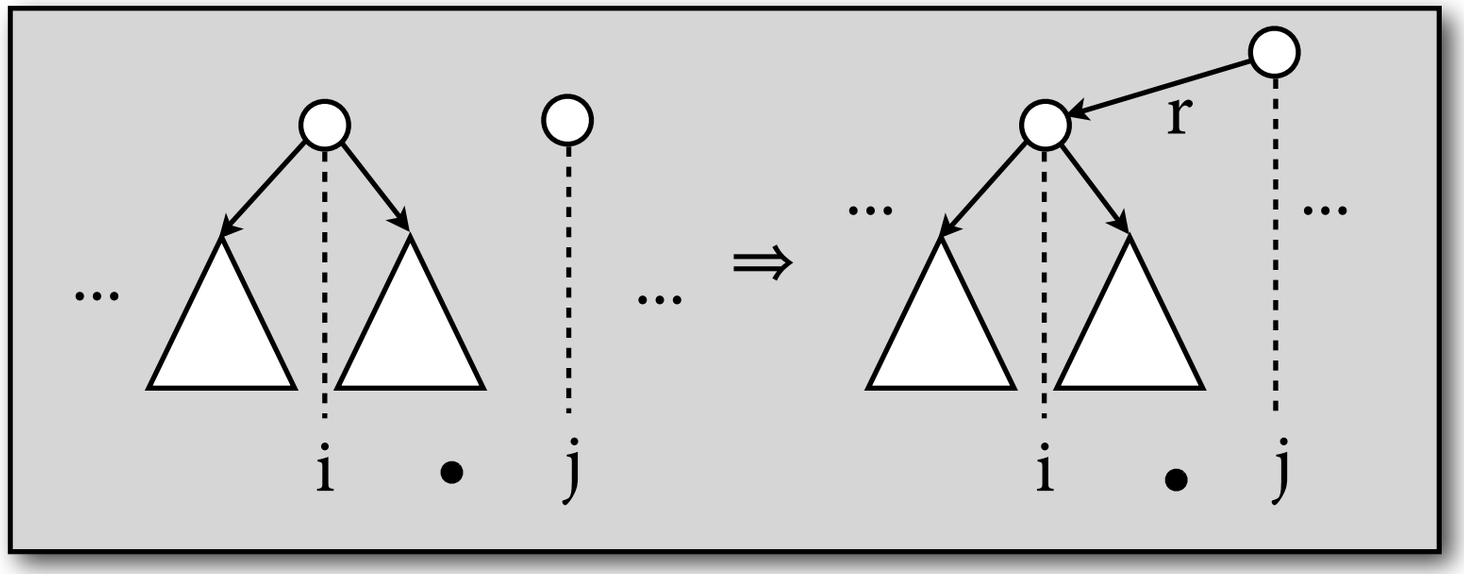
- Anfangs-Item: $(\varepsilon, (1, ..., n), h_0, d_0)$ mit
  - $h_0(i) = 0$ für alle i: kein Wort hat bisher Vater
  - $d_0(i) = r_0$ für alle i: keine Kantenlabels bekannt

- Ziel-Items: $(\sigma, \varepsilon, h, d)$ für beliebige σ, h, d
  - h, d beschreiben dann zusammen einen Dependenzbaum für den Eingabestring
  - σ kann noch mehrere Tokens enthalten; das kann bedeuten, dass noch für mehrere Wörter $h(i) = 0$ hat, die Wörter also keinen Vater haben.

Parser-Operationen

- Left-Arc(r): Oberstes Stack-Token wird (linkes) r-Kind des nächsten Eingabe-Tokens.

$$\frac{(\sigma|i, j|\tau, h, d) \quad h(i) = 0}{(\sigma, j|\tau, h[i \mapsto j], d[i \mapsto r])}$$

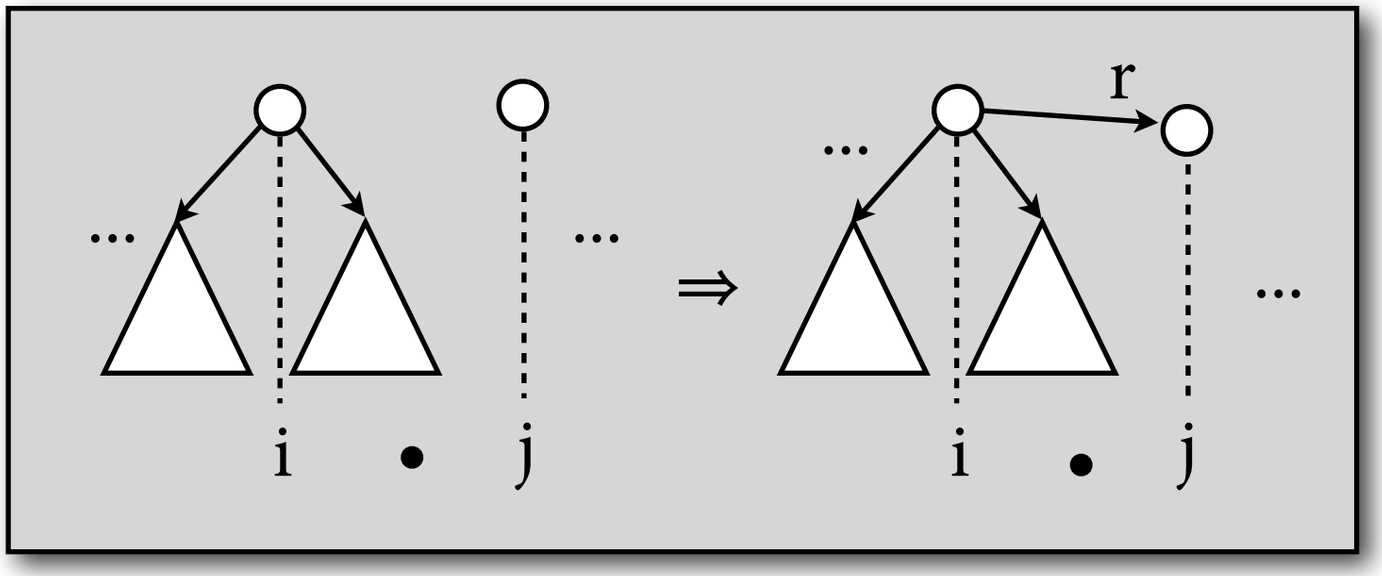


- i verschwindet vom Stack, denn in projektiven Bäumen kann i keine Kinder weiter rechts haben.

Parser-Operationen

- Right-Arc(r): Eingabetoken wird (rechtes) r-Kind des obersten Stack-Tokens.

$$\frac{(\sigma|i, j|\tau, h, d) \quad h(j) = 0}{(\sigma|i|j, \tau, h[j \mapsto i], d[j \mapsto r])}$$

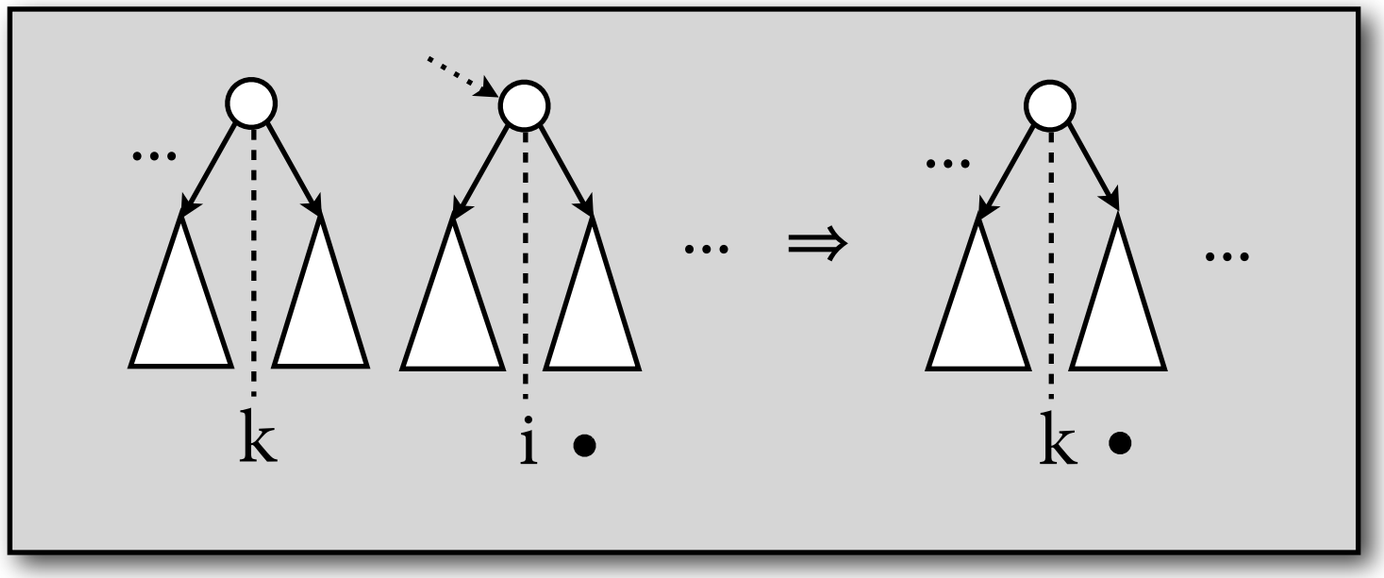


- i, j bleiben beide auf dem Stack, weil beide noch weitere rechte Kinder bekommen können.

Parser-Operationen

- Reduce: Entferne oberstes Stack-Token.

$$\frac{(\sigma|i, \tau, h, d) \quad h(i) \neq 0}{(\sigma, \tau, h, d)}$$



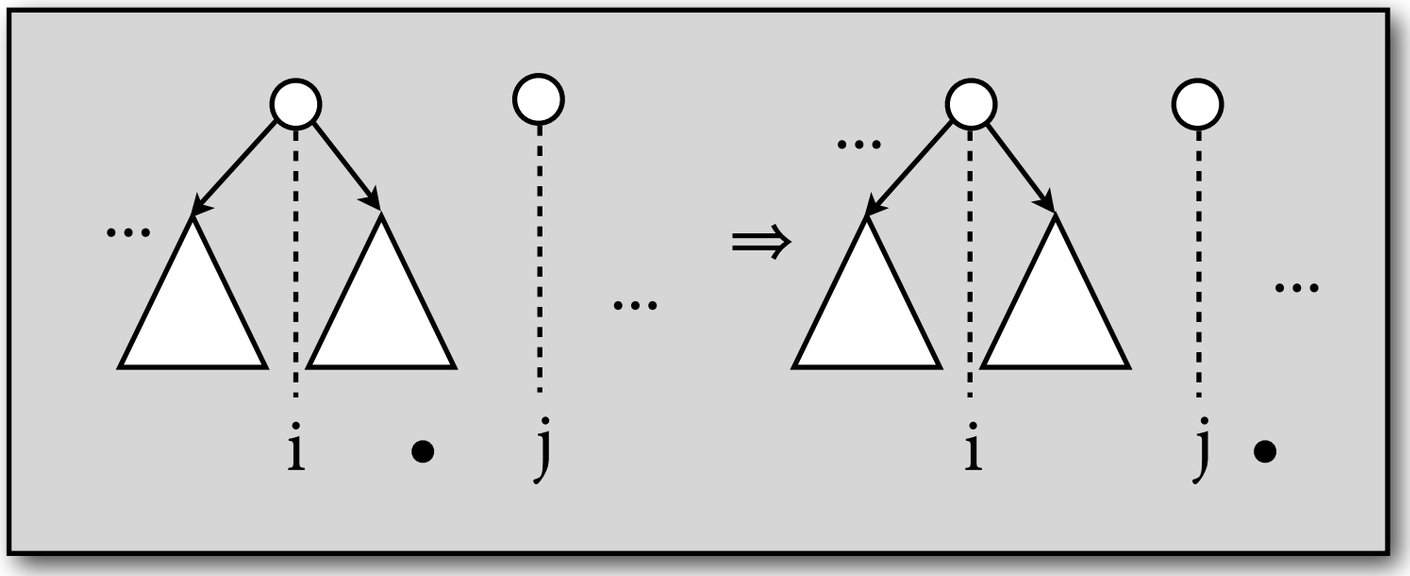
- Danach kann weiter links stehender Knoten rechte Kinder bekommen kann. Entscheidet, dass i keine weiteren Kinder bekommt.

- i muss schon Vater haben.

Parser-Operationen

- Shift: Verschiebt Token von Eingabe auf Stack.

$$\frac{(\sigma, i|\tau, h, d)}{(\sigma|i, \tau, h, d)}$$



- Entscheidet, dass i und j disjunkt im Baum liegen: alle (weiteren) Kinder von i stehen weiter rechts.

Ablauf des Parsers

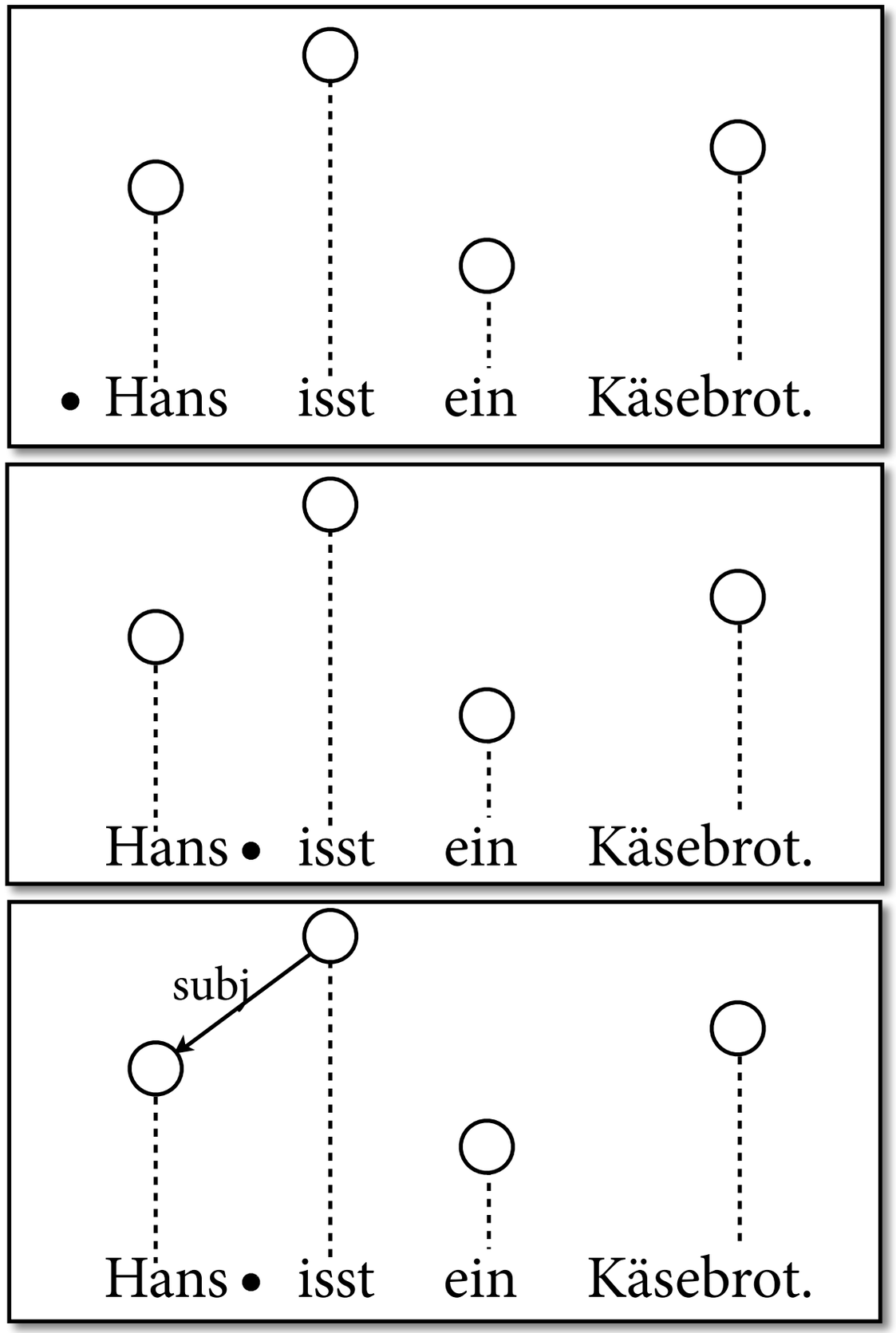
(ε, H isst ein K)

⇓ Shift

(H, isst ein K)

⇓ Left-Arc(subj)

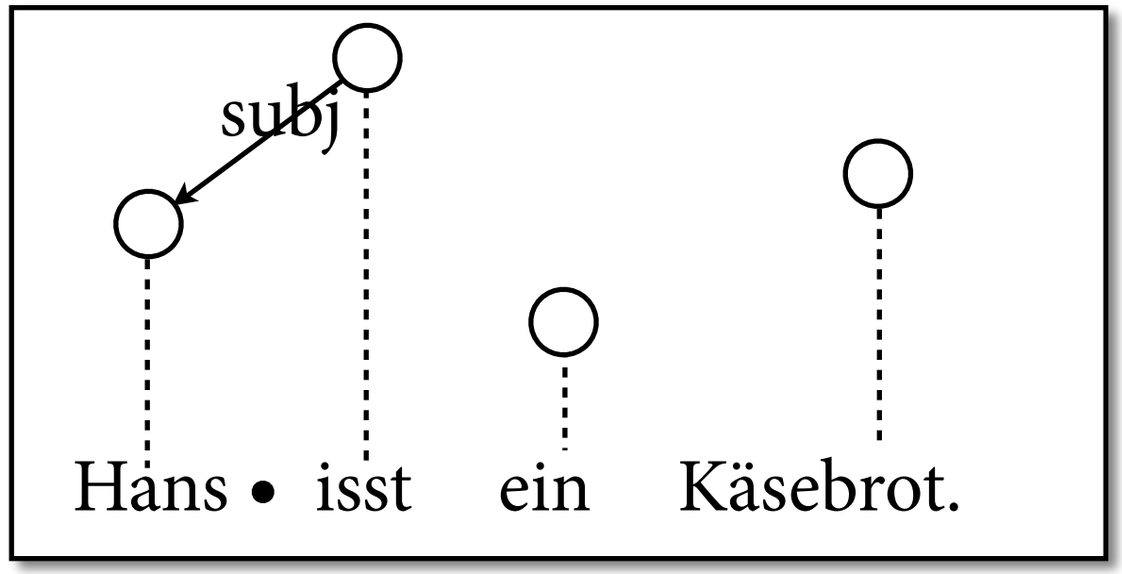
(ε, isst ein K)



Ablauf des Parsers

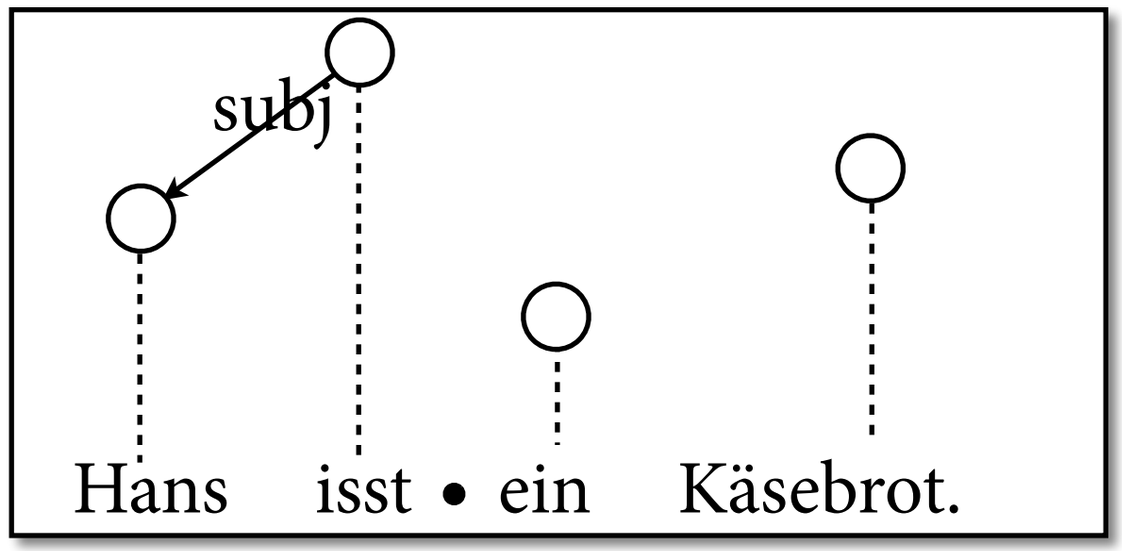
(ε, isst ein K)

⇓ Shift

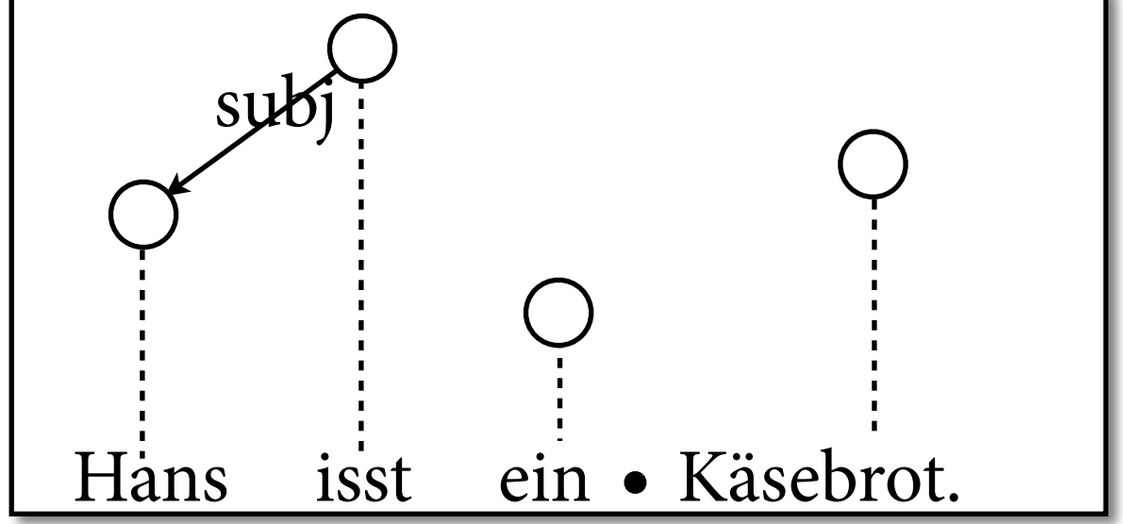


(isst, ein K)

⇓ Shift



(isst ein, K)



Ablauf des Parsers

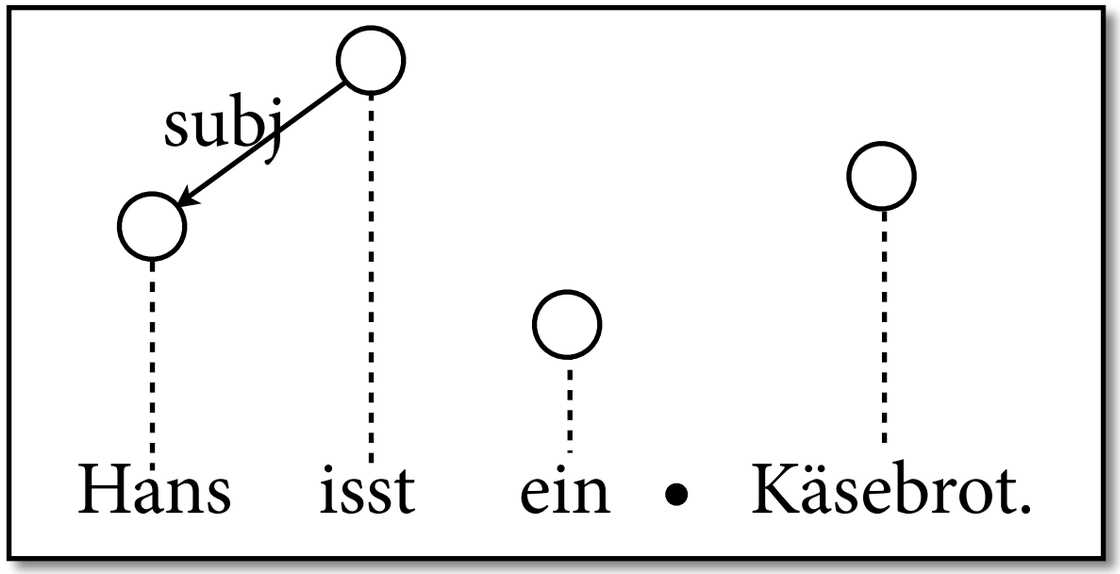
(isst ein, K)

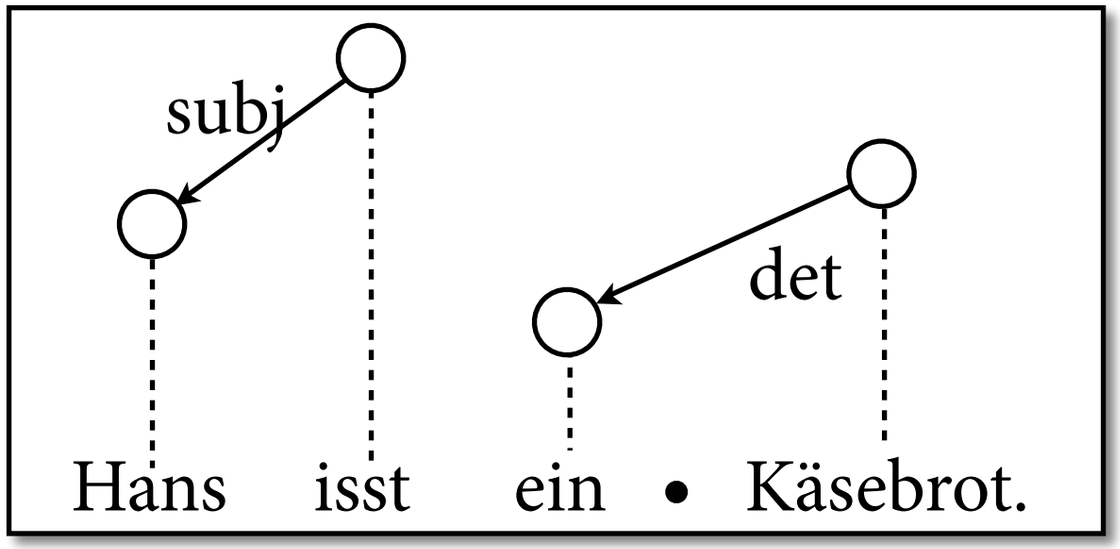
⇓ Left-Arc(det)

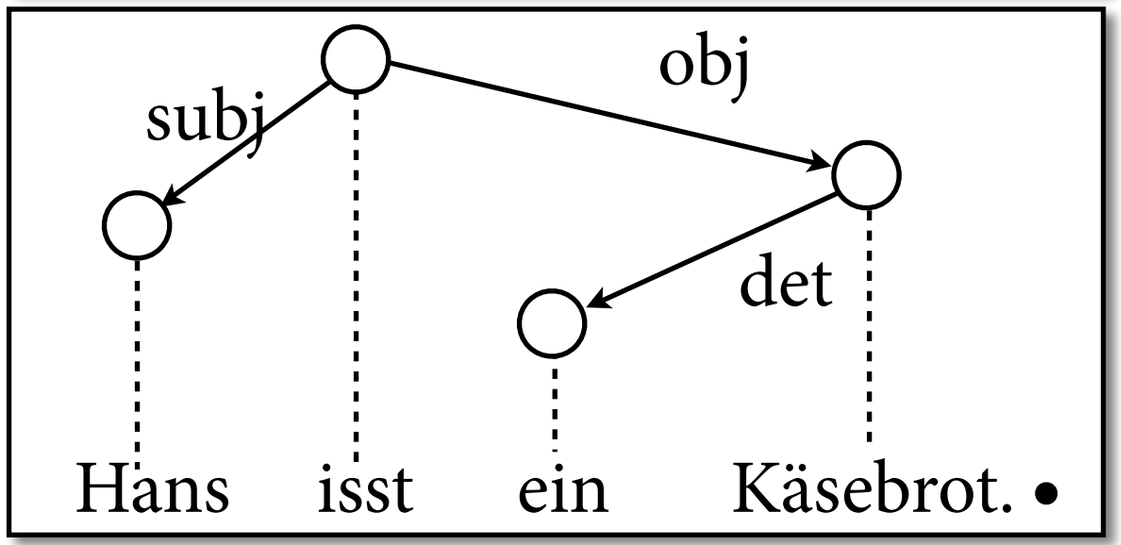
(isst, K)

⇓ Right-Arc(obj)

(isst K, ε)







Nichtdeterminismus

- Parser-Operationen sind massiv nichtdeterministisch: In jedem Schritt viele verschiedene Operationen anwendbar.

- Wir nehmen ein *Orakel* an. Für jedes Item sagt Orakel, welche Operation anzuwenden ist. Kein Backtracking!

- Implementieren mit Klassifikator.

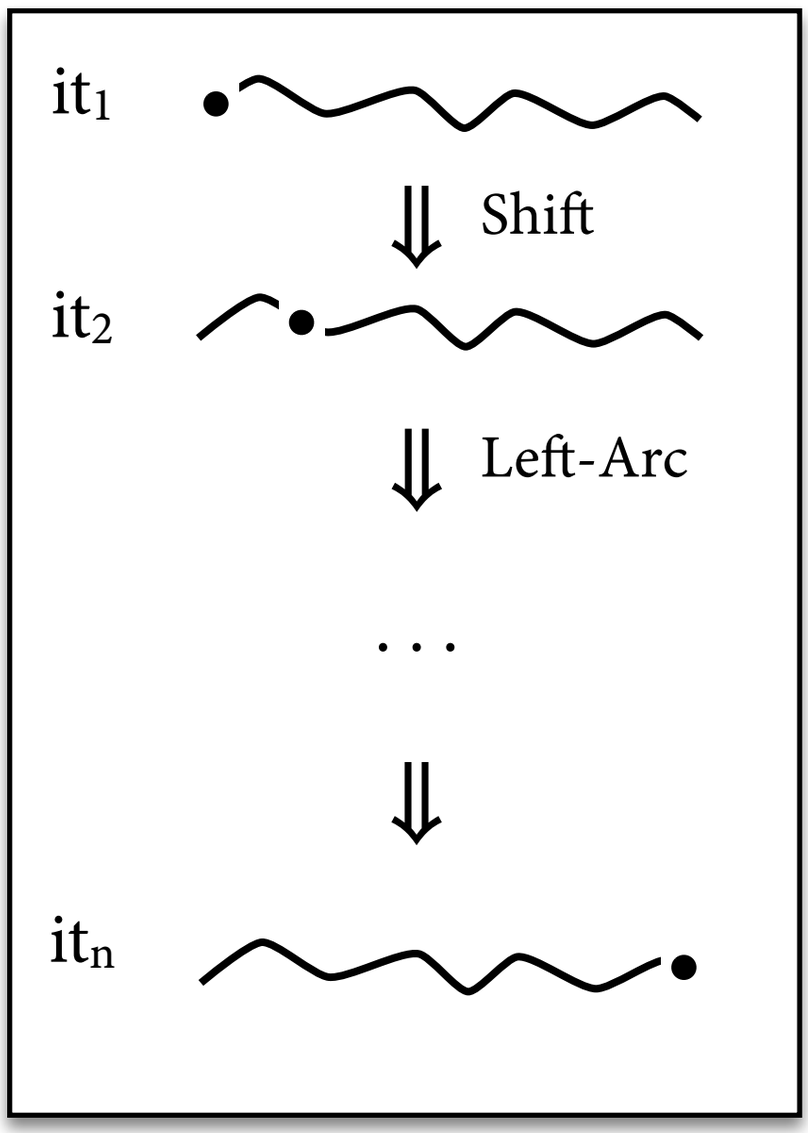
Parsing als Klassifizieren

- (Deterministisches) Parsing geht jetzt so:

```
c = start-item
while (c kein ziel-item und eine Operation
 auf c anwendbar ist):
    op = next-operation(c)
    c = perform-operation(c, op)
```

- “next-operation” klassifiziert c.
  Zentrale Frage: Woher bekommen wir den Klassifikator?

Grundansatz



Korpus von Ableitungen

$(ft_1(it_1), ..., ft_m(it_1), \text{Shift})$

$(ft_1(it_2), ..., ft_m(it_2), \text{Left-Arc})$

. . .

$(ft_1(it_{n+1}), ..., ft_m(it_{n+1}), \text{Shift})$

Feature-Vektoren

Klassifikator trainieren

Features für Dependenzparsing

- MaltParser (= Standard-Implementierung des Nivre-Algorithmus) bietet "Baukasten" für Features an:
  - σ_i: i-tes Token auf dem Stack (von oben)
  - τ_i: i-tes Token in der ungelesenen Eingabe
  - h(x): Vater von x im Baum, der von h definiert wird
  - l(x), r(x): linkestes (rechtestes) Kind von x im Baum
  - p(x): POS-Tag von x
  - d(x): Kantenlabel von h(x) zu x
  - daraus beliebige Terme, z.B. $p(l(\sigma_0))$

Standard-Featureset

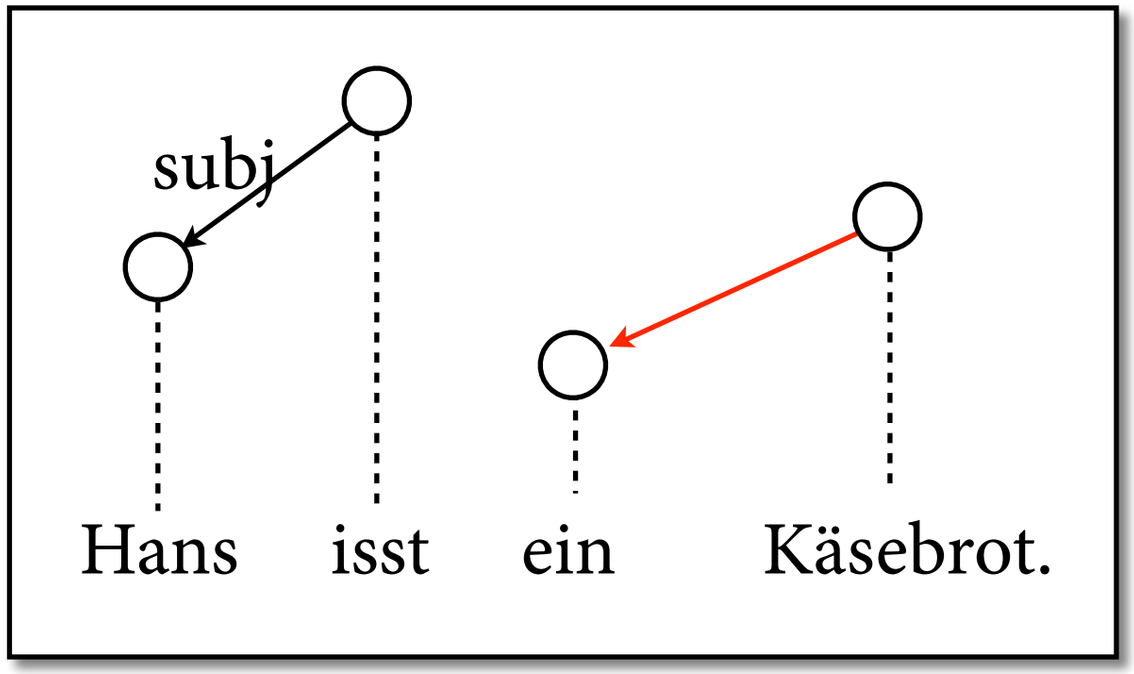
- Für jede Sprache bzw. Korpus kann Nutzer aus Baukasten eigenes Featureset bauen.

- Standard-Featureset:
  - POS-Tags $p(\sigma_0)$, $p(\sigma_1)$, $p(\tau_0)$, $p(\tau_1)$, $p(\tau_2)$, $p(\tau_3)$
  - Tokens σ_0, τ_0, τ_1, $h(\sigma_0)$
  - Grammatische Rollen $d(l(\sigma_0))$, $d(\sigma_0)$, $d(r(\sigma_0))$, $d(l(\tau_0))$

- Richtiges Featureset für Sprache finden
  = Ausprobieren und evaluieren.

Beispiel

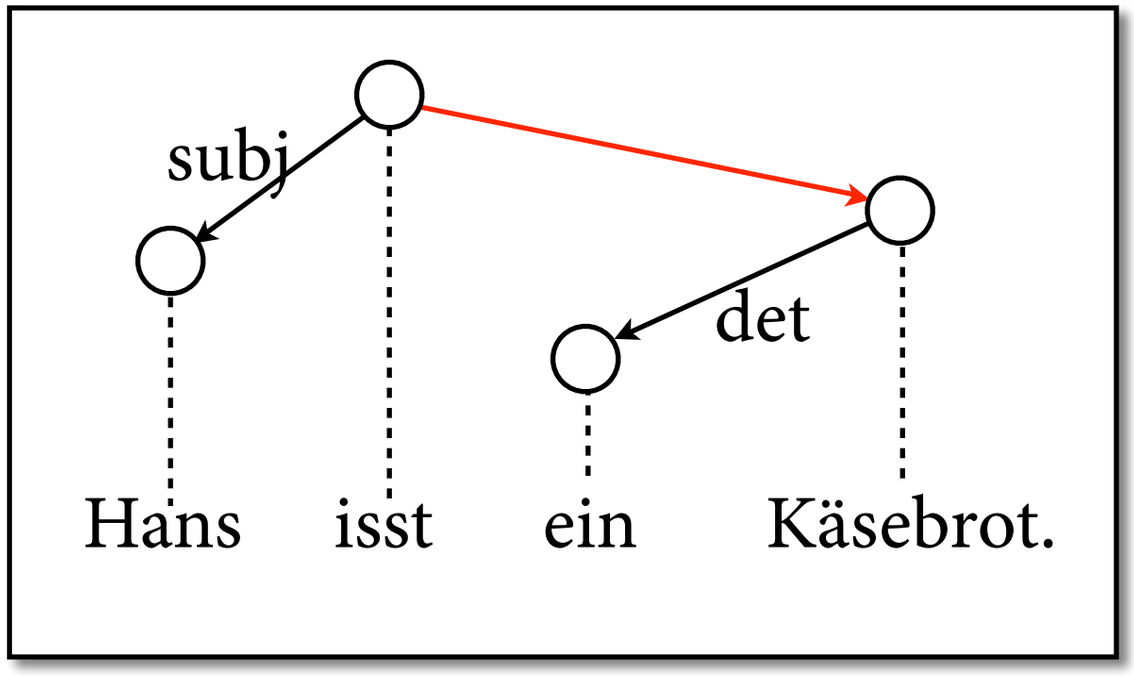
(isst ein, K)



p(	p(	p(	p(	p(	p(	σ	τ	τ	h(	d(l(	d(	d(r(	d(l(	Op
ART	VVFIN	NN	0	0	0	ein	K.	0	0	0	0	0	0	LA

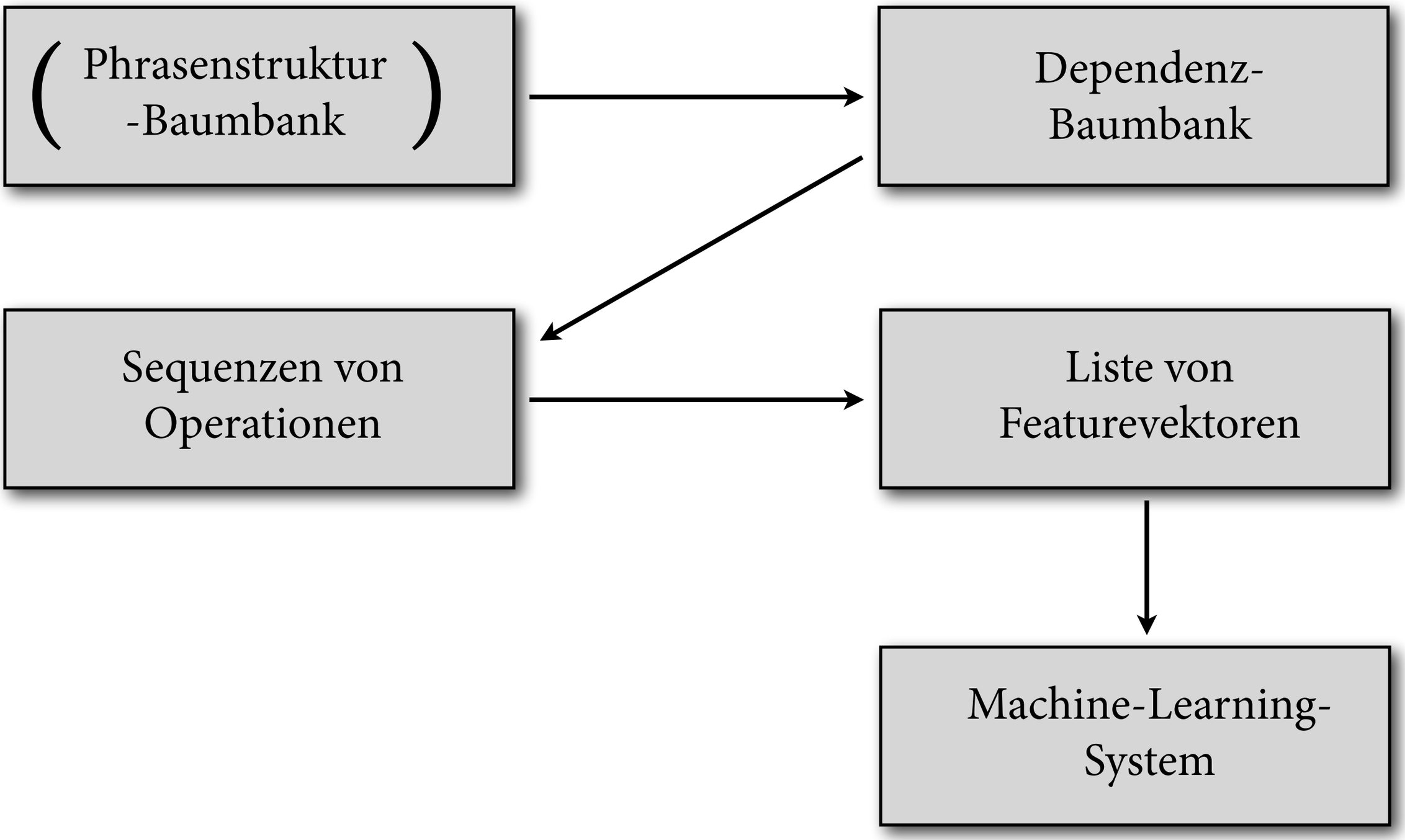
Beispiel

(isst, K)



p(	p(	p(	p(	p(	p(	σ	τ	τ	h(	d(l(	d(	d(r(	d(l(	Op
ART	VVFIN	NN	0	0	0	ein	K.	0	0	0	0	0	0	LA
VVFIN	0	NN	0	0	0	isst	K.	0	0	subj	0	subj	det	RA

Training: Komplett



Evaluation

- Welcher Anteil von Kanten korrekt vorhergesagt?
  - *label accuracy*: #(Knoten mit korrektem Label für eingehende Kante) / #Knoten
  - *unlabeled attachment score*: #(Knoten mit korrektem Vater) / #Knoten
  - *labeled attachment score*: #(Knoten mit korrektem Vater und eingehendem Kantenlabel) / #Knoten

Der CoNLL-X Shared Task

- Evaluation von Dependenzparsern auf 13 Sprachen (2006).
  - Vergleich von 12 teilnehmenden Systemen
- Für jede Sprache: Entweder bestehende Dependenz-Baumbank verwendet oder Phrasenstruktur-Baumbank in Dependenz konvertiert.
  - für Deutsch: TIGER-Baumbank (Saarbrücken/Stuttgart/Potsdam), 900.000 Tokens Zeitungstext

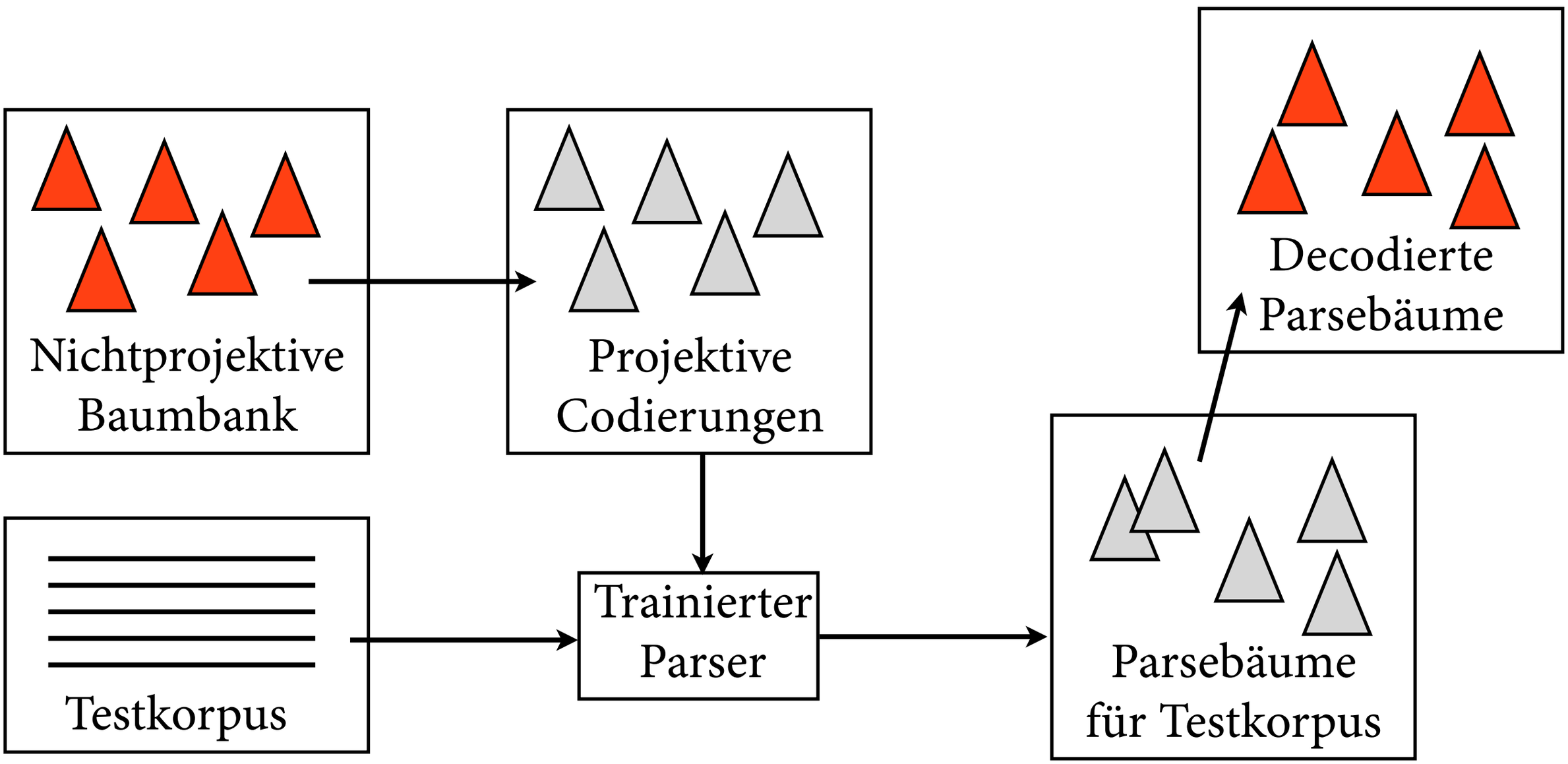
MaltParser: Ergebnisse auf CoNLL

Sprache	UAS	LAS
Arabisch	77.5	66.7
Chinesisch	90.5	86.9
Tschechisch	84.8	78.4
Dänisch	89.8	84.8
Niederländisch	81.4	78.6
Deutsch	88.8	85.8
Japanisch	93.1	91.7
Slowenisch	78.7	70.3
Türkisch	75.8	65.7

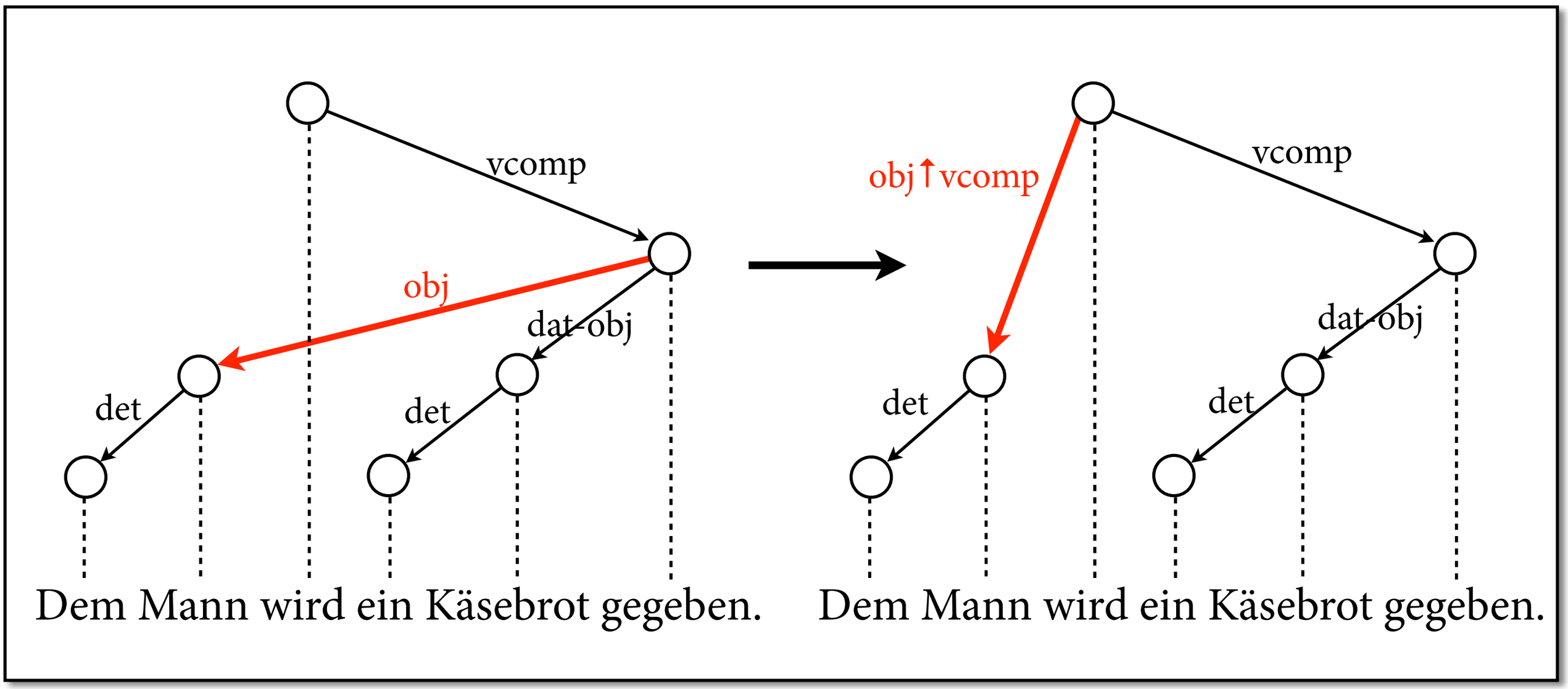
MaltParser = Nivres Implementierung des Parsers; Ergebnisse aus CoNLL-X, 2007

Pseudoprojektives Parsing

- Ein einfacher Ansatz zum Parsen von leicht nichtprojektiven Bäumen (Nivre & Nilsson 05):



Beispiel



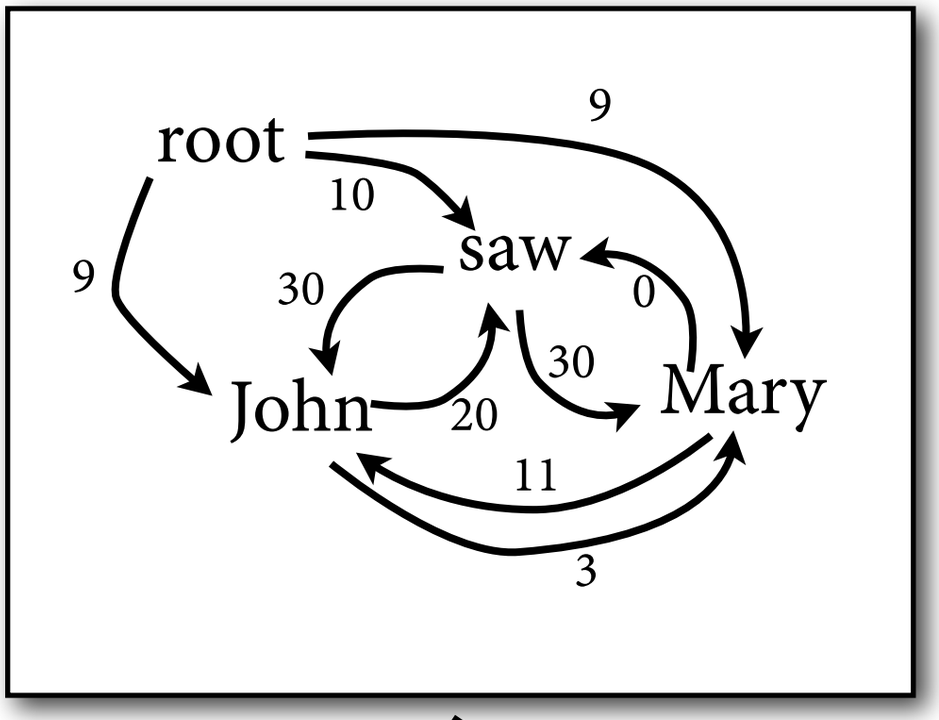
Pseudoprojektive Codierung “HEAD”:
Kantenlabel codiert d(Vater,Knoten) + d(Großvater, Vater)

Voll nichtprojektives Parsing

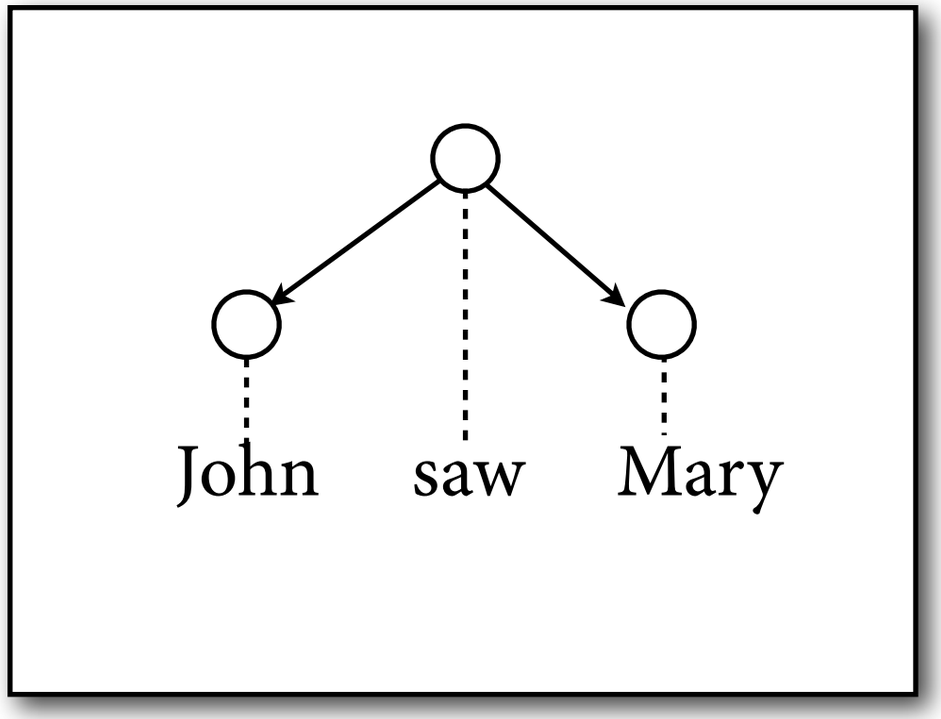
- MST-Parser (McDonald et al. 05):
  - berechne gerichteten gewichteten Graphen;
    Knoten = Wörter, Gewicht von Kante = Qualität dieser Kante gemäß der Features.
  - berechne dann Maximal Spanning Tree, d.h. den Baum, der jeden Knoten enthält und unter allen solchen Bäumen maximales Gewicht hat (geht effizient).
- Spannbäume sind beliebig nichtprojektiv.
- Ähnlich gut wie MaltParser; komplementäre Stärken (McDonald & Nivre 07).

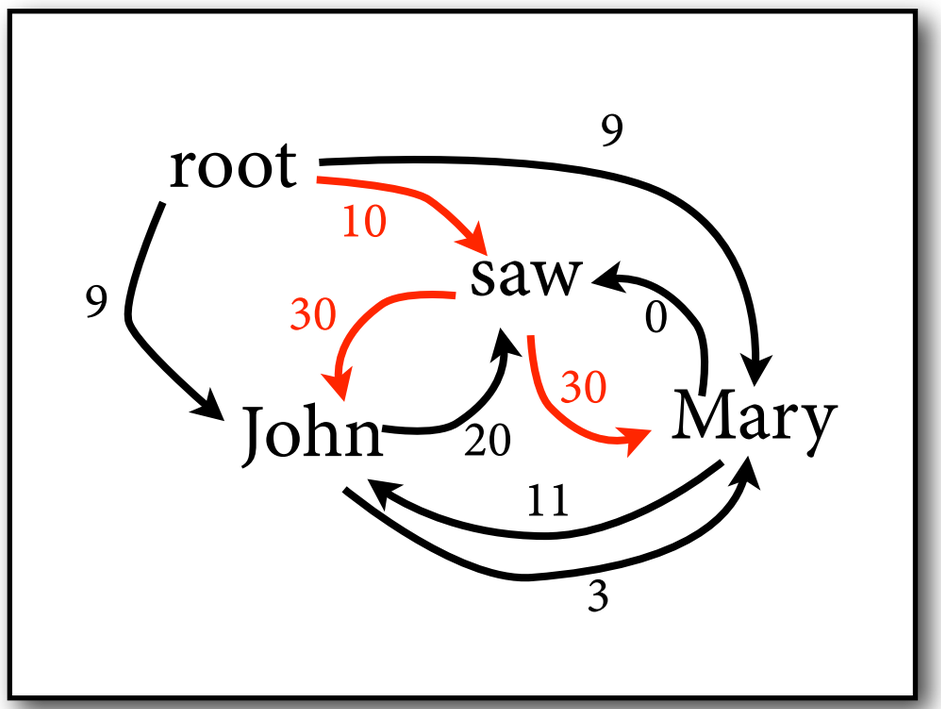
Beispiel

Gewichteter Graph



Dependenzbaum





Maximaler Spannbaum

Zusammenfassung

- Dependenzbäume: Wörter direkt mit Dependenzkanten verbinden.

- Deterministisches Dependenzparsing: Unter Parser-Operationen mit Klassifikator auswählen.

- Projektivität vs. Nichtprojektivität.