

Parsing mit Backtracking

Vorlesung “Computerlinguistische Techniken”
Alexander Koller

20. Oktober 2015

Kontextfreie Grammatiken

$T = \{\text{Hans, isst, Käsebro\u00df, ein}\}$

$N = \{S, NP, VP, V, N, Det\}$; Startsymbol: S

Produktionsregeln:

$S \rightarrow NP VP$

$NP \rightarrow Det N$

$VP \rightarrow V NP$

$V \rightarrow \text{isst}$

$NP \rightarrow \text{Hans}$

$Det \rightarrow \text{ein}$

$N \rightarrow \text{Käsebro\u00df}$

Ableitung

$S \Rightarrow NP VP \Rightarrow \text{Hans } VP$

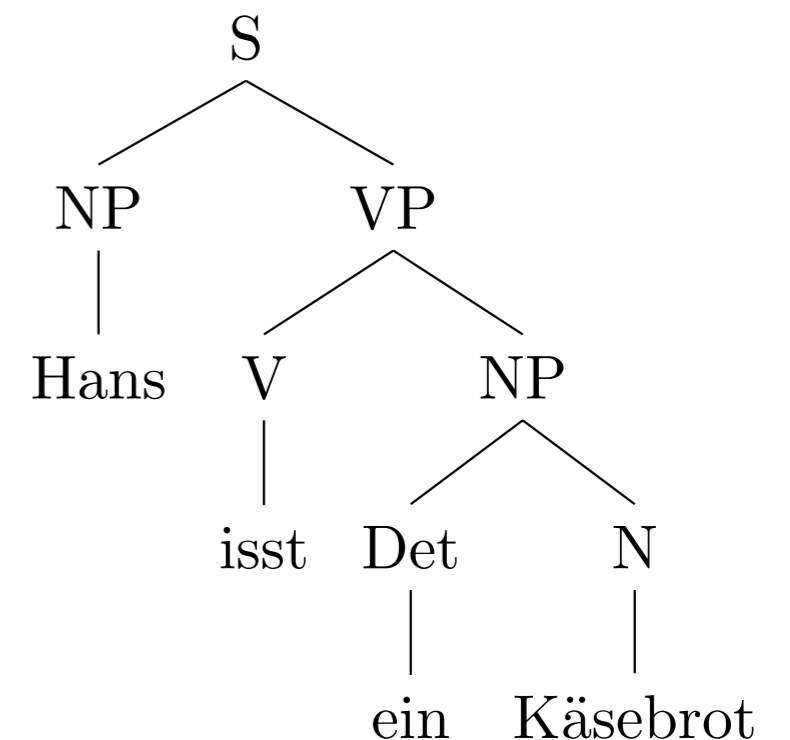
$\Rightarrow \text{Hans } V NP \Rightarrow \text{Hans isst } NP$

$\Rightarrow \text{Hans isst } Det N$

$\Rightarrow \text{Hans isst ein } N$

$\Rightarrow \text{Hans isst ein Käsebro\u00df}$

Parsebaum



Parsing

- Gegeben seien eine kfG G und ein Wort w .
- Das *Wortproblem* ist die Frage, ob $w \in L(G)$ ist.
Wortproblem wird von *Erkennung* gelöst.
- Das *Parsingproblem* ist das Problem, alle Parsebäume von w bzgl. G zu bestimmen.
Parsingproblem wird von *Parser* gelöst.

Parsing: Beispiele

$S \rightarrow a S b$

$S \rightarrow a b$

G

Ist $aabb \in L(G)$?

Wenn ja, was sind die Parsebäume?

Parsing: Beispiele

$S \rightarrow a S b$

$S \rightarrow a b$

G

Ist $abab \in L(G)$?

Wenn ja, was sind die Parsebäume?

Parsing: Beispiele

$$\begin{array}{l} S \rightarrow S S \\ S \rightarrow a \end{array} \quad G$$

Ist $aaa \in L(G)$?

Wenn ja, was sind die Parsebäume?

Parsing: Beispiele

$S \rightarrow S S$ $S \rightarrow a$	G
--	---

Ist $aaa \in L(G)$?

Wenn ja, was sind die Parsebäume?

Ist $aaaa \in L(G)$?

Wenn ja, was sind die Parsebäume?

Recursive-Descent-Parsing

- Wie löst man das Wort- und das Parsingproblem so systematisch, dass man es implementieren kann?
- Ein erster Ansatz für Parsing: Recursive Descent.
(Wir lösen hier nur das Wortproblem.)
- Wir nehmen an, dass kfGs in Chomsky-Normalform (CNF) sind: Jede Regel ist von der Form $A \rightarrow B C$ oder $A \rightarrow a$.

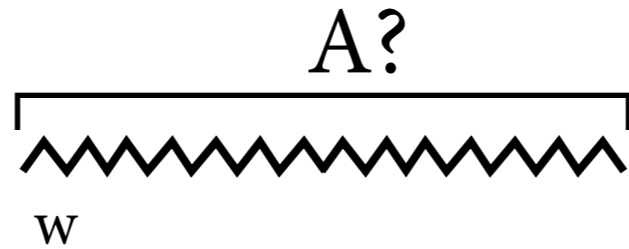
Recursive-Descent-Parsing

- Rekursiver Algorithmus, der für A und w die Frage “ $A \Rightarrow^* w$?” entscheidet.

~~~~~  
w

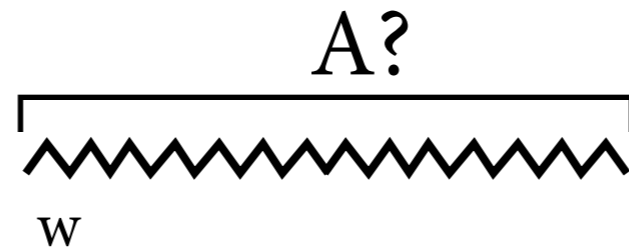
# Recursive-Descent-Parsing

- Rekursiver Algorithmus, der für  $A$  und  $w$  die Frage “ $A \Rightarrow^* w$ ?” entscheidet.



# Recursive-Descent-Parsing

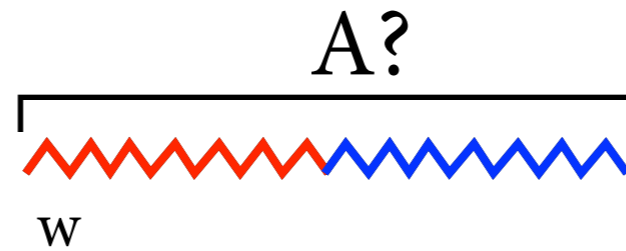
- Rekursiver Algorithmus, der für  $A$  und  $w$  die Frage “ $A \Rightarrow^* w$ ?” entscheidet.



$A \rightarrow B C$

# Recursive-Descent-Parsing

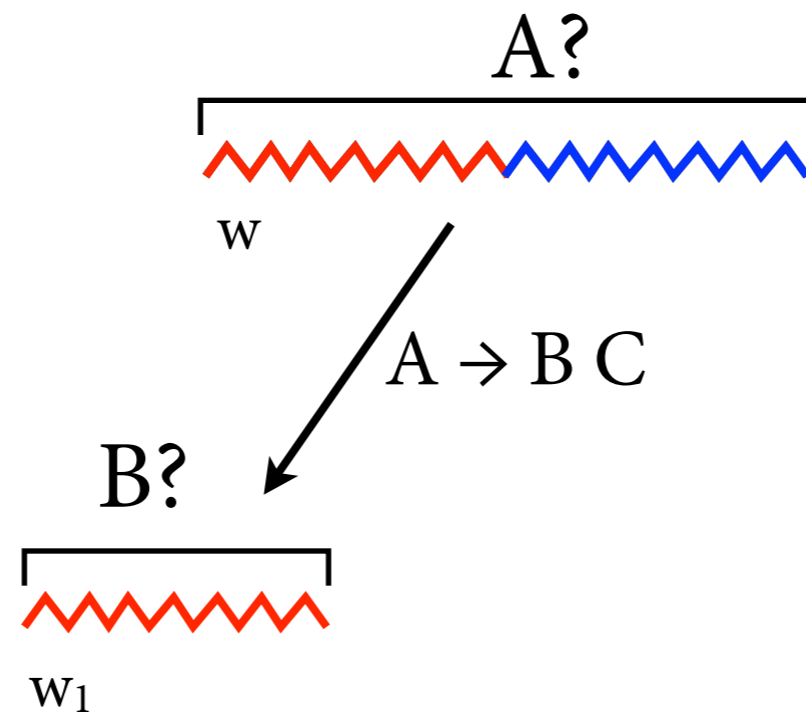
- Rekursiver Algorithmus, der für  $A$  und  $w$  die Frage “ $A \Rightarrow^* w$ ?” entscheidet.



$A \rightarrow B C$

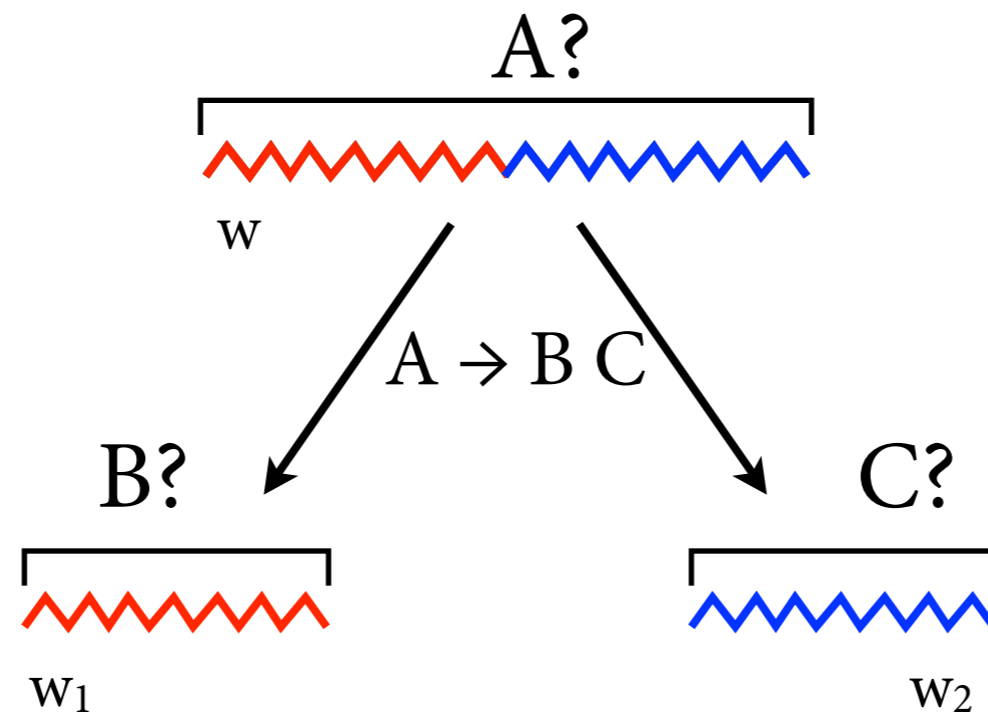
# Recursive-Descent-Parsing

- Rekursiver Algorithmus, der für  $A$  und  $w$  die Frage “ $A \Rightarrow^* w$ ?” entscheidet.



# Recursive-Descent-Parsing

- Rekursiver Algorithmus, der für  $A$  und  $w$  die Frage “ $A \Rightarrow^* w?$ ” entscheidet.



# Recursive-Descent-Parsing

- Der Algorithmus sieht so aus:
  - ▶ Wir wollen wissen, ob  $A \Rightarrow^* w$  ist.
  - ▶ Wenn  $w \in T$  und es eine Regel  $A \rightarrow w$  gibt, gib “ja” zurück, sonst “nein”.
  - ▶ Rate eine Regel  $A \rightarrow B C$ , die wir anwenden wollen.
  - ▶ Rate eine Zerlegung  $w = w_1 w_2$ .
  - ▶ Wenn sowohl  $B \Rightarrow^* w_1$  als auch  $C \Rightarrow^* w_2$  “ja” gesagt haben, gib “ja” zurück, sonst “nein”.

# RD-Parsing: Ein Beispiel

$S \rightarrow A B$

$A \rightarrow a$

$B \rightarrow b$

$S \rightarrow A C$

$C \rightarrow B C$

$C \rightarrow c$



# RD-Parsing: Ein Beispiel

 $S \rightarrow A B$  $A \rightarrow a$  $B \rightarrow b$  $S \rightarrow A C$  $C \rightarrow B C$  $C \rightarrow c$  $S \Rightarrow^* a b c ?$

# RD-Parsing: Ein Beispiel

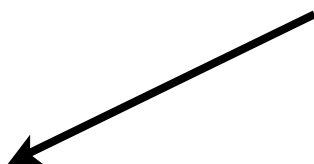
 $S \rightarrow A B$  $A \rightarrow a$  $B \rightarrow b$  $S \rightarrow A C$  $C \rightarrow B C$  $C \rightarrow c$  $S \Rightarrow^* a b c ?$ 

(versuche  $abc = a \cdot bc, S \rightarrow A B$ )

# RD-Parsing: Ein Beispiel

 $S \rightarrow A B$  $A \rightarrow a$  $B \rightarrow b$  $S \rightarrow A C$  $C \rightarrow B C$  $C \rightarrow c$  $S \Rightarrow^* a b c ?$ 

(versuche  $abc = a \cdot bc$ ,  $S \rightarrow A B$ )

 $A \Rightarrow^* a ?$

# RD-Parsing: Ein Beispiel

$S \rightarrow A B$

$A \rightarrow a$

$B \rightarrow b$

$S \rightarrow A C$

$C \rightarrow B C$

$C \rightarrow c$

$S \Rightarrow^* a b c ?$

(versuche  $abc = a \cdot bc$ ,  $S \rightarrow A B$ )

$A \Rightarrow^* a ?$

$B \Rightarrow^* b c ?$

# RD-Parsing: Ein Beispiel

$S \rightarrow A B$

$A \rightarrow a$

$B \rightarrow b$

$S \rightarrow A C$

$C \rightarrow B C$

$C \rightarrow c$

$S \Rightarrow^* a b c ?$

(versuche  $abc = a \cdot bc$ ,  $S \rightarrow A B$ )

$A \Rightarrow^* a ?$

(ja, da  $A \rightarrow a$ )

$B \Rightarrow^* b c ?$

# RD-Parsing: Ein Beispiel

$S \rightarrow A B$

$A \rightarrow a$

$B \rightarrow b$

$S \rightarrow A C$

$C \rightarrow B C$

$C \rightarrow c$

$S \Rightarrow^* a b c ?$

(versuche  $abc = a \cdot bc$ ,  $S \rightarrow A B$ )

$A \Rightarrow^* a ?$  ✓  
(ja, da  $A \rightarrow a$ )

$B \Rightarrow^* b c ?$

# RD-Parsing: Ein Beispiel

$S \rightarrow A B$

$A \rightarrow a$

$B \rightarrow b$

$S \rightarrow A C$

$C \rightarrow B C$

$C \rightarrow c$

$S \Rightarrow^* a b c ?$

(versuche  $abc = a \cdot bc$ ,  $S \rightarrow A B$ )

$A \Rightarrow^* a ?$  ✓  
(ja, da  $A \rightarrow a$ )

$B \Rightarrow^* b c ?$   
(nein: es gibt keine  
binären Regeln für B)

# RD-Parsing: Ein Beispiel

$S \rightarrow A B$

$A \rightarrow a$

$B \rightarrow b$

$S \rightarrow A C$

$C \rightarrow B C$

$C \rightarrow c$

$S \Rightarrow^* a b c ?$

(versuche  $abc = a \cdot bc$ ,  $S \rightarrow A B$ )

$A \Rightarrow^* a ?$  ✓  
(ja, da  $A \rightarrow a$ )

$B \Rightarrow^* b c ?$  ✗  
(nein: es gibt keine  
binären Regeln für B)



# RD-Parsing: Ein Beispiel

$S \rightarrow A B$

$A \rightarrow a$

$B \rightarrow b$

$S \rightarrow A C$

$C \rightarrow B C$

$C \rightarrow c$

$S \Rightarrow^* a b c ?$  **X**

(versuche  $abc = a \cdot bc$ ,  $S \rightarrow A B$ )

$A \Rightarrow^* a ?$  **✓**

(ja, da  $A \rightarrow a$ )

$B \Rightarrow^* b c ?$  **X**

(nein: es gibt keine  
binären Regeln für B)

# RD-Parsing: Ein Beispiel

$S \rightarrow A B$

$A \rightarrow a$

$B \rightarrow b$

$S \rightarrow A C$

$C \rightarrow B C$

$C \rightarrow c$

# RD-Parsing: Ein Beispiel

 $S \rightarrow A B$  $A \rightarrow a$  $B \rightarrow b$  $S \rightarrow A C$  $C \rightarrow B C$  $C \rightarrow c$  $S \Rightarrow^* a b c ?$

# RD-Parsing: Ein Beispiel

 $S \rightarrow A B$  $A \rightarrow a$  $B \rightarrow b$  $S \rightarrow A C$  $C \rightarrow B C$  $C \rightarrow c$  $S \Rightarrow^* a b c ?$ 

(versuche  $abc = a \cdot bc, S \rightarrow A C$ )

# RD-Parsing: Ein Beispiel

$S \rightarrow A B$

$A \rightarrow a$

$B \rightarrow b$

$S \rightarrow A C$

$C \rightarrow B C$

$C \rightarrow c$

$S \Rightarrow^* a b c ?$

(versuche  $abc = a \cdot bc$ ,  $S \rightarrow A C$ )

$A \Rightarrow^* a ?$  ✓  
(ja, da  $A \rightarrow a$ )

$C \Rightarrow^* b c ?$

# RD-Parsing: Ein Beispiel

$S \rightarrow A B$

$A \rightarrow a$

$B \rightarrow b$

$S \rightarrow A C$

$C \rightarrow B C$

$C \rightarrow c$

$S \Rightarrow^* a b c ?$

(versuche  $abc = a \cdot bc$ ,  $S \rightarrow A C$ )

$A \Rightarrow^* a ?$  ✓  
(ja, da  $A \rightarrow a$ )

$C \Rightarrow^* b c ?$   
(versuche  $bc = b \cdot c$ ,  $C \rightarrow B C$ )

# RD-Parsing: Ein Beispiel

$S \rightarrow A B$

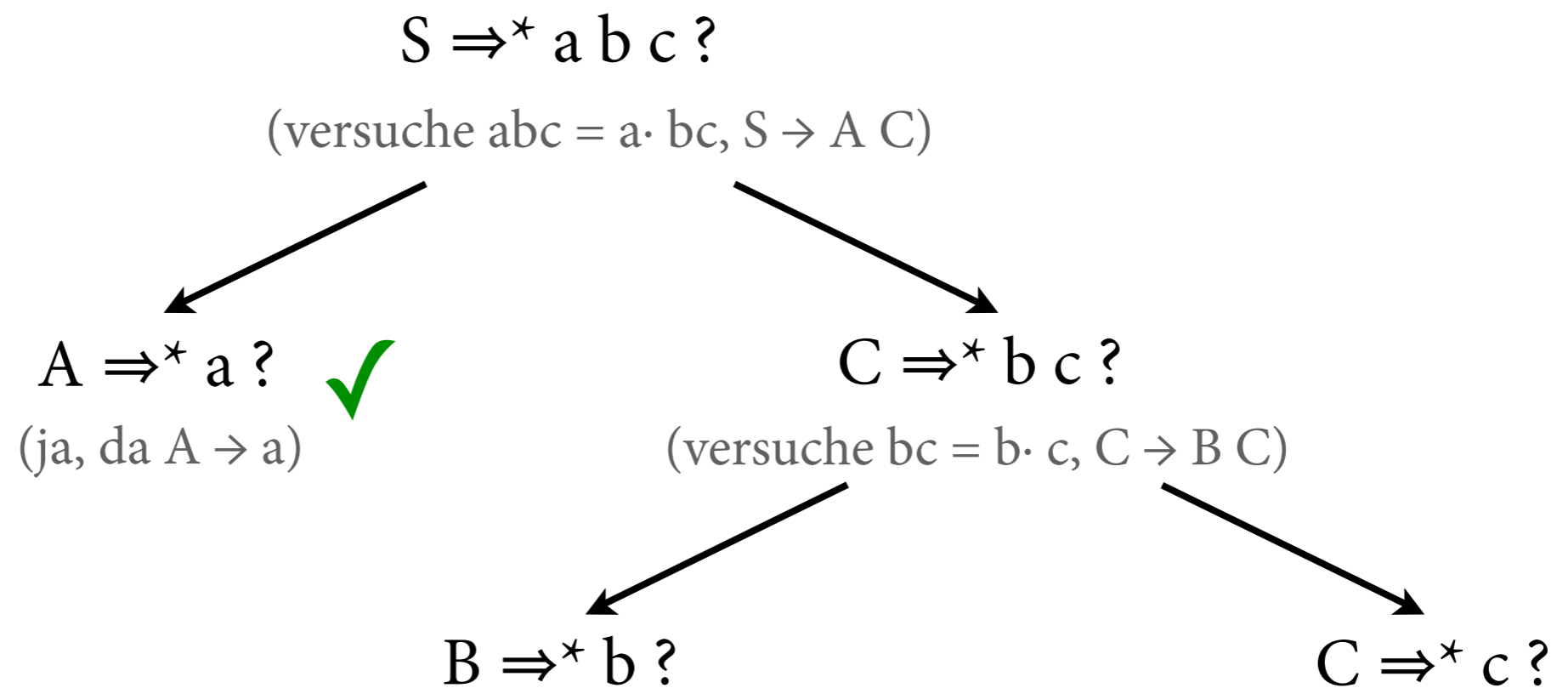
$A \rightarrow a$

$B \rightarrow b$

$S \rightarrow A C$

$C \rightarrow B C$

$C \rightarrow c$



# RD-Parsing: Ein Beispiel

$S \rightarrow A B$

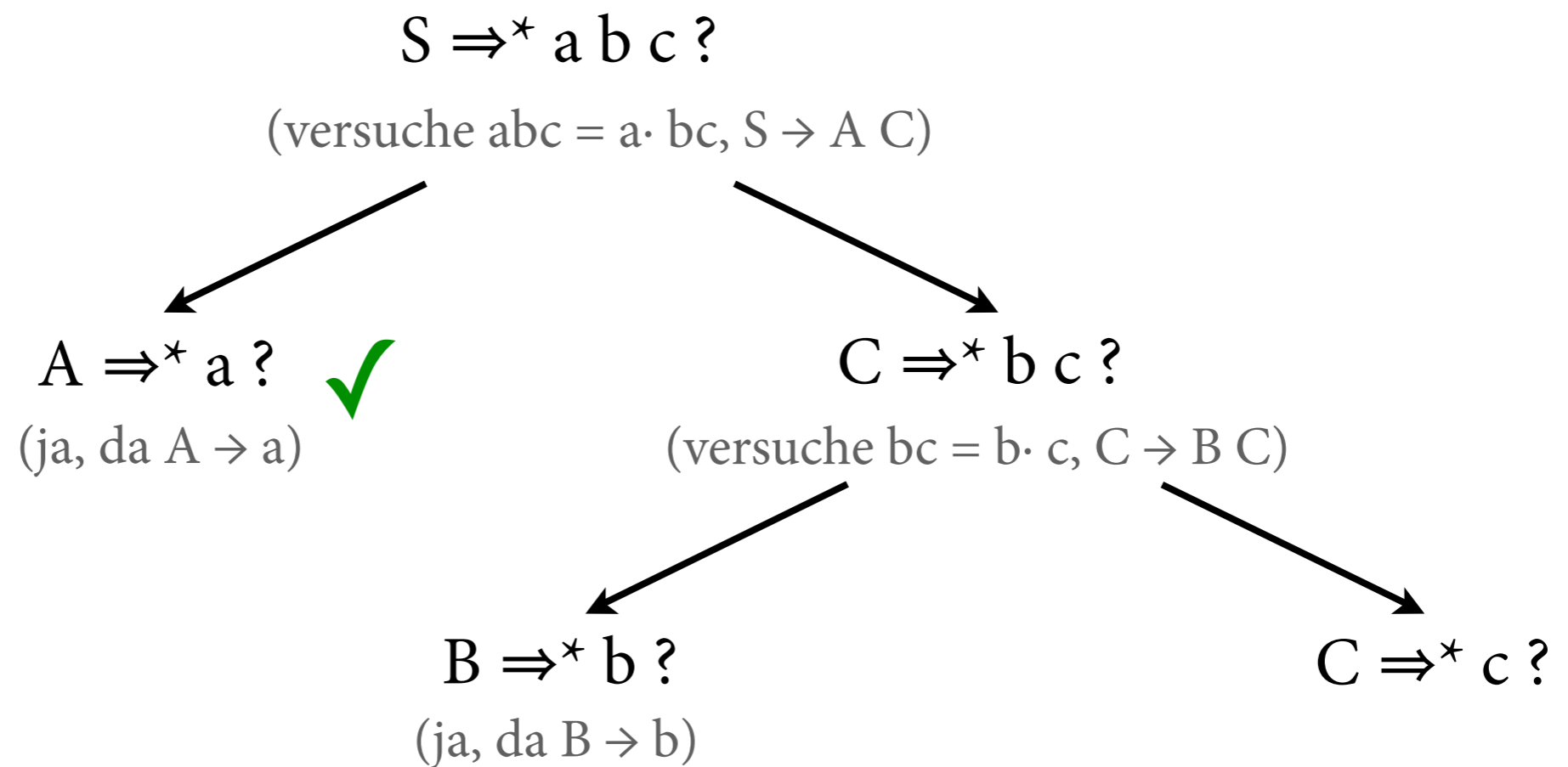
$A \rightarrow a$

$B \rightarrow b$

$S \rightarrow A C$

$C \rightarrow B C$

$C \rightarrow c$





# RD-Parsing: Ein Beispiel

$S \rightarrow A B$

$A \rightarrow a$

$B \rightarrow b$

$S \rightarrow A C$

$C \rightarrow B C$

$C \rightarrow c$

$S \Rightarrow^* a b c ?$

(versuche  $abc = a \cdot bc$ ,  $S \rightarrow A C$ )

$A \Rightarrow^* a ?$  ✓  
(ja, da  $A \rightarrow a$ )

$C \Rightarrow^* b c ?$

(versuche  $bc = b \cdot c$ ,  $C \rightarrow B C$ )

$B \Rightarrow^* b ?$  ✓  
(ja, da  $B \rightarrow b$ )

$C \Rightarrow^* c ?$

# RD-Parsing: Ein Beispiel

$S \rightarrow A B$

$A \rightarrow a$

$B \rightarrow b$

$S \rightarrow A C$

$C \rightarrow B C$

$C \rightarrow c$

$S \Rightarrow^* a b c ?$

(versuche  $abc = a \cdot bc$ ,  $S \rightarrow A C$ )

$A \Rightarrow^* a ?$  ✓

(ja, da  $A \rightarrow a$ )

$C \Rightarrow^* b c ?$

(versuche  $bc = b \cdot c$ ,  $C \rightarrow B C$ )

$B \Rightarrow^* b ?$  ✓

(ja, da  $B \rightarrow b$ )

$C \Rightarrow^* c ?$

(ja, da  $C \rightarrow c$ )

# RD-Parsing: Ein Beispiel

$S \rightarrow A B$

$A \rightarrow a$

$B \rightarrow b$

$S \rightarrow A C$

$C \rightarrow B C$

$C \rightarrow c$

$S \Rightarrow^* a b c ?$

(versuche  $abc = a \cdot bc$ ,  $S \rightarrow A C$ )

$A \Rightarrow^* a ?$  ✓

(ja, da  $A \rightarrow a$ )

$C \Rightarrow^* b c ?$

(versuche  $bc = b \cdot c$ ,  $C \rightarrow B C$ )

$B \Rightarrow^* b ?$  ✓

(ja, da  $B \rightarrow b$ )

$C \Rightarrow^* c ?$  ✓

(ja, da  $C \rightarrow c$ )

# RD-Parsing: Ein Beispiel

$S \rightarrow A B$

$A \rightarrow a$

$B \rightarrow b$

$S \rightarrow A C$

$C \rightarrow B C$

$C \rightarrow c$

$S \Rightarrow^* a b c ?$

(versuche  $abc = a \cdot bc$ ,  $S \rightarrow A C$ )

$A \Rightarrow^* a ?$  ✓  
(ja, da  $A \rightarrow a$ )

$C \Rightarrow^* b c ?$  ✓

(versuche  $bc = b \cdot c$ ,  $C \rightarrow B C$ )

$B \Rightarrow^* b ?$  ✓  
(ja, da  $B \rightarrow b$ )

$C \Rightarrow^* c ?$  ✓  
(ja, da  $C \rightarrow c$ )

# RD-Parsing: Ein Beispiel

$S \rightarrow A B$

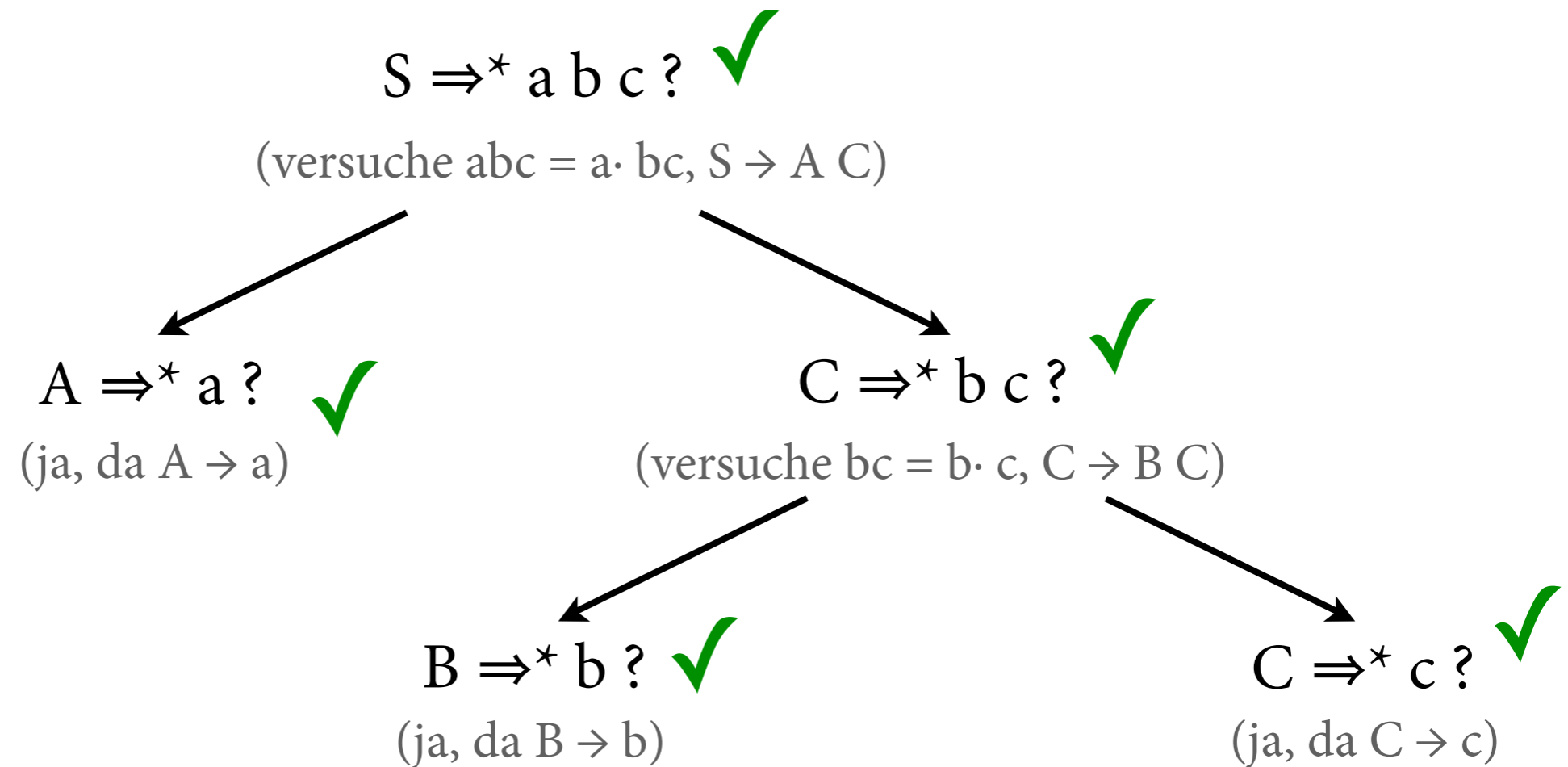
$A \rightarrow a$

$B \rightarrow b$

$S \rightarrow A C$

$C \rightarrow B C$

$C \rightarrow c$



# Zwei Beobachtungen

- Der RD-Algorithmus ist rekursiv:  
Um  $A \Rightarrow^* w$  zu testen, überprüfen wir  
 $B \Rightarrow^* w_1$  und  $C \Rightarrow^* w_2$ .
- Der RD-Algorithmus ist nichtdeterministisch:  
Er “rät” Regeln und Stringzerlegungen.
  - ▶ Implementierung auf echtem Computer muss deshalb alle Möglichkeiten der Reihe nach durchprobieren.
  - ▶ Revidiert dabei Entscheidungen aus früheren rekursiven Aufrufen -> *Backtracking*.

# Pseudocode

```
function S(w, i, k):  
  for j = i+1, ..., k-1 do  
    if NP(w, i, j) and VP(w, j, k) then  
      return true  
    else  
      return false  
    end if  
  end for
```

```
function NP(w, i, k):  
  if k = i + 1 and w(i) = "Hans" then  
    return true  
  else  
    return false  
  end if
```

# Pseudocode

$S \Rightarrow^* w_i \dots w_{k-1} ?$

```
function S(w, i, k):  
  for j = i+1, ..., k-1 do  
    if NP(w, i, j) and VP(w, j, k) then  
      return true  
    else  
      return false  
    end if  
  end for
```

```
function NP(w, i, k):  
  if k = i + 1 and w(i) = "Hans" then  
    return true  
  else  
    return false  
  end if
```



# Pseudocode

$S \Rightarrow^* w_i \dots w_{k-1} ?$

```
function S(w, i, k):  
  for j = i+1, ..., k-1 do  
    if NP(w, i, j) and VP(w, j, k) then  
      return true  
    else  
      return false  
    end if  
  end for
```

$NP \Rightarrow^* w_i \dots w_{j-1} ?$

```
function NP(w, i, k):  
  if k = i + 1 and w(i) = "Hans" then  
    return true  
  else  
    return false  
  end if
```

# Pseudocode

$S \Rightarrow^* w_i \dots w_{k-1} ?$

```
function S(w, i, k):  
  for j = i+1, ..., k-1 do  
    if NP(w, i, j) and VP(w, j, k) then  
      return true  
    else  
      return false  
    end if  
  end for
```

$VP \Rightarrow^* w_j \dots w_{k-1} ?$

$NP \Rightarrow^* w_i \dots w_{j-1} ?$

```
function NP(w, i, k):  
  if k = i + 1 and w(i) = "Hans" then  
    return true  
  else  
    return false  
  end if
```

# Pseudocode

$S \Rightarrow^* w_i \dots w_{k-1} ?$

```
function S(w, i, k):  
  for j = i+1, ..., k-1 do  
    if NP(w, i, j) and VP(w, j, k) then  
      return true  
    else  
      return false  
    end if  
  end for
```

$VP \Rightarrow^* w_j \dots w_{k-1} ?$

$NP \Rightarrow^* w_i \dots w_{j-1} ?$

$NP \Rightarrow^* w_i \dots w_{k-1} ?$

```
function NP(w, i, k):  
  if k = i + 1 and w(i) = "Hans" then  
    return true  
  else  
    return false  
  end if
```

# Pseudocode

$S \Rightarrow^* w_i \dots w_{k-1} ?$

```
function S(w, i, k):  
  for j = i+1, ..., k-1 do  
    if NP(w, i, j) and VP(w, j, k) then  
      return true  
    else  
      return false  
    end if  
  end for
```

$VP \Rightarrow^* w_j \dots w_{k-1} ?$

$NP \Rightarrow^* w_i \dots w_{j-1} ?$

$NP \Rightarrow^* w_i \dots w_{k-1} ?$

```
function NP(w, i, k):  
  if k = i + 1 and w(i) = "Hans" then  
    return true  
  else  
    return false  
  end if
```

ja falls  $w_i \dots w_{k-1} = \text{Hans}$

# Korrektheit und Vollständigkeit

- Warum sollen wir glauben, dass RD-Erkennen für alle Eingaben das richtige Ergebnis liefert?
- Wir müssen beweisen:
  - ▶ *Korrektheit*: RD-Erkennen behauptet  $w \in L(G)$  nur dann, wenn es auch wahr ist.
  - ▶ *Vollständigkeit*: Wenn  $w \in L(G)$  ist, dann behauptet das der RD-Erkennen auch.

# Probleme und Algorithmen

- *Problem*: Funktion von irgendwelchen Eingaben zu  $\{0,1\}$ .
  - ▶ Wortproblem von kfGs: gegeben Eingabe  $(G,w)$ , gib 1 aus gdw  $w \in L(G)$ , sonst 0.
- *Algorithmus*: abstrakte Berechnungsvorschrift, die das Problem löst.
  - ▶ RD-Erkennen ist Algorithmus für das Wortproblem.
- Algorithmen kann man in Programmiersprachen *implementieren*.

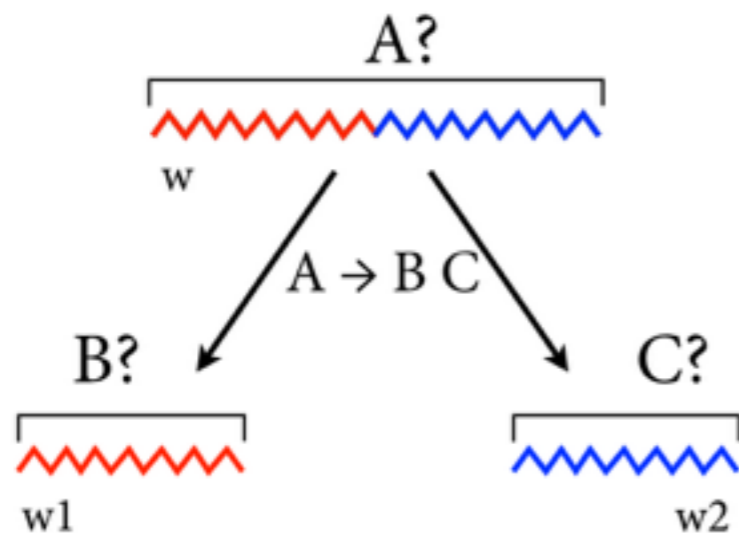
# Probleme und Algorithmen

## Problem

Wortproblem( $G, w$ ) = 1  
gdw  $w \in L(G)$

```
function S(w, i, k):  
  for j = i+1, ..., k-1 do  
    if NP(w, i, j) and VP(w, j, k) then  
      return true  
    else  
      return false  
    end if  
  end for
```

## Algorithmus



*Programm*

# Beweistechnik

- Korrektheit und Vollständigkeit von Parsingalgo.  
beweist man typischerweise mit *vollständiger Induktion*:
  - ▶ Korrektheit: ... über Länge der Berechnung des Algorithmus
  - ▶ Vollständigkeit: ... über Länge der Ableitung.
- Vollständige Induktion:
  - ▶ Zeige, dass Aussage für  $n = 0$  (oder  $n = 1$ ) wahr ist.
  - ▶ Zeige: Wenn Aussage für beliebiges  $n$  wahr ist, dann ist sie auch für  $n+1$  wahr.  
(alternativ: sei  $n$  beliebig, und nimm an, dass Aussage für alle  $k \leq n$  wahr ist),
  - ▶ Daraus folgt dann: Aussage für alle  $n \geq 0$  (bzw.  $n \geq 1$ ) wahr.



# Korrektheit des RD-Erkenner

- Korrektheit: Wenn RD-Erkenner  $w \in L(G)$  behauptet, dann ist es auch wahr.
- Zeige stärkere Aussage: Wenn RD-Erkenner  $A \Rightarrow^* w$  behauptet, ist das wahr. (Daraus folgt Korrektheit.)
- Induktion über die Länge  $n$  der Berechnung (= Anzahl der rekursiven Aufrufe):
  - ▶ Fall  $n = 1$ : Algo. hat ohne rekursiven Aufruf direkt “ja” gesagt. Dann war das, weil es Regel  $A \rightarrow w$  gibt, also ist  $A \Rightarrow^* w$ .

# Korrektheit des RD-Erkenner

- Zeige stärkere Aussage: Wenn RD-Erkennenner  $A \Rightarrow^* w$  behauptet, ist das wahr. (Daraus folgt Korrektheit.)
  - ▶ Fall  $n \rightarrow n+1$ : Algo. hat “ja” gesagt und dabei im ersten Schritt  $w = w_1 w_2$  zerlegt und Regel  $A \rightarrow B C$  angewendet, und rekursive Aufrufe  $B \Rightarrow^* w_1$  und  $C \Rightarrow^* w_2$  waren beide “ja”.
  - ▶ Rekursive Berechnungen waren beide kürzer:  $k_B, k_C \leq n$ . Wegen Induktionsannahme gilt deshalb:  
 $B \Rightarrow^* w_1$  und  $C \Rightarrow^* w_2$  sind beide wahr.
  - ▶ Deshalb ist auch  $A \Rightarrow^* w$  wahr, nämlich wie folgt:  
 $A \Rightarrow B C \Rightarrow^* w_1 C \Rightarrow^* w_1 w_2 = w$ .

# Vollständigkeit des RD-Erkenner

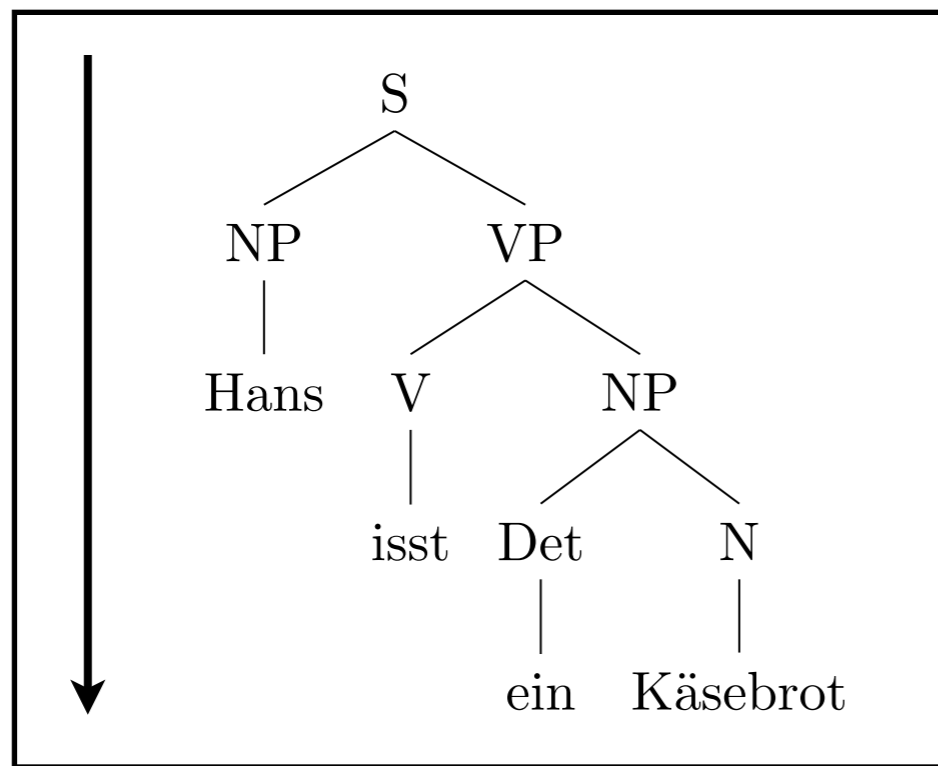
- Vollständigkeit: Wenn  $w \in L(G)$  wahr ist, dann behauptet es der RD-Erkenner auch.
- Zeige stärkere Aussage: Wenn  $A \Rightarrow^* w$  wahr ist, dann sagt RD-Erkenner “ja”. (Daraus folgt Vollständigkeit.)
- Induktion über die Länge  $n$  der Ableitung  $A \Rightarrow^* w$ :
  - ▶ Fall  $n = 1$ :  $A \Rightarrow w$ , es gibt also Regel  $A \rightarrow w$ .  
RD-Erkenner sagt ohne rekursiven Aufruf “ja”.

# Vollständigkeit des RD-Erkenner

- Vollständigkeit: Wenn  $w \in L(G)$  wahr ist, dann behauptet es der RD-Erkenner auch.
  - ▶ Fall  $n \rightarrow n + 1$ : Es gilt  $A \Rightarrow^* w$ . Diese Ableitung beginnt mit einem ersten Schritt,  $A \Rightarrow B C \Rightarrow^* w_1 C \Rightarrow^* w_1 w_2 = w$ .
  - ▶ RD-Erkenner überprüft alle Zerlegungen von  $w$ , also insbesondere auch  $A \rightarrow B C$  und  $w = w_1 w_2$ .
  - ▶ Die Ableitungen  $B \Rightarrow^* w_1$  und  $C \Rightarrow^* w_2$  haben beide weniger als  $n$  Schritte. Nach Induktionsannahme sagt RD-Erkenner für beide rekursive Aufrufe deshalb “ja”.
  - ▶ Daher sagt RD-Erkenner auch für  $A \Rightarrow^* w$  “ja”.

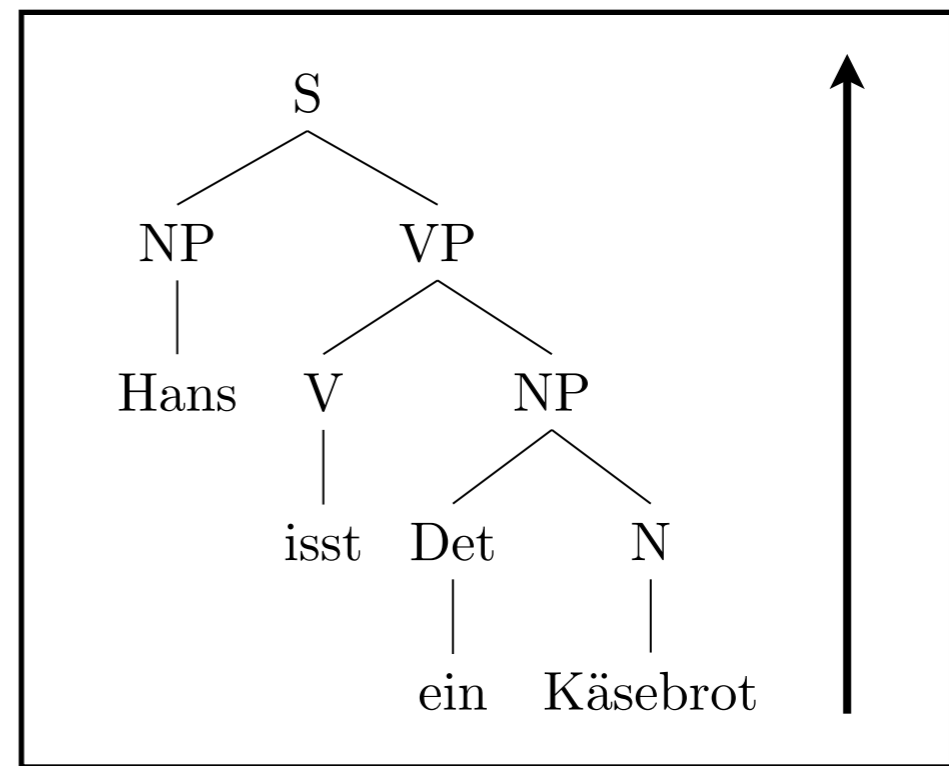
# Top-Down vs. Bottom-Up

- Parser kann den Parsebaum top-down oder bottom-up zu berechnen versuchen.



top-down

(z.B. Recursive Descent)



bottom-up

# Bottom-Up-Parsing

$S \rightarrow NP VP$

$V \rightarrow \text{isst}$

$\text{Det} \rightarrow \text{ein}$

$NP \rightarrow \text{Det } N$

$NP \rightarrow \text{Hans}$

$N \rightarrow \text{Käsebrod}$

$VP \rightarrow V NP$

Hans isst ein Käsebrod.

# Bottom-Up-Parsing

$S \rightarrow NP VP$

$V \rightarrow \text{isst}$

$\text{Det} \rightarrow \text{ein}$

$NP \rightarrow \text{Det } N$

$NP \rightarrow \text{Hans}$

$N \rightarrow \text{Käsebrod}$

$VP \rightarrow V NP$

NP

|

Hans

isst

ein

Käsebrod.

# Bottom-Up-Parsing

$S \rightarrow NP VP$

$V \rightarrow \text{isst}$

$\text{Det} \rightarrow \text{ein}$

$NP \rightarrow \text{Det } N$

$NP \rightarrow \text{Hans}$

$N \rightarrow \text{Käsebrod}$

$VP \rightarrow V NP$

NP

V

|

|

Hans

isst

ein

Käsebrod.



# Bottom-Up-Parsing

$S \rightarrow NP VP$

$V \rightarrow \text{isst}$

$\text{Det} \rightarrow \text{ein}$

$NP \rightarrow \text{Det } N$

$NP \rightarrow \text{Hans}$

$N \rightarrow \text{Käsebrot}$

$VP \rightarrow V NP$

NP

V

Det

|

|

|

Hans

isst

ein

Käsebrot.

# Bottom-Up-Parsing

$S \rightarrow NP VP$

$V \rightarrow \text{isst}$

$\text{Det} \rightarrow \text{ein}$

$NP \rightarrow \text{Det } N$

$NP \rightarrow \text{Hans}$

$N \rightarrow \text{Käsebrod}$

$VP \rightarrow V NP$

NP

V

Det

N

|

|

|

|

Hans

isst

ein

Käsebrod.

# Bottom-Up-Parsing

$S \rightarrow NP VP$

$V \rightarrow \text{isst}$

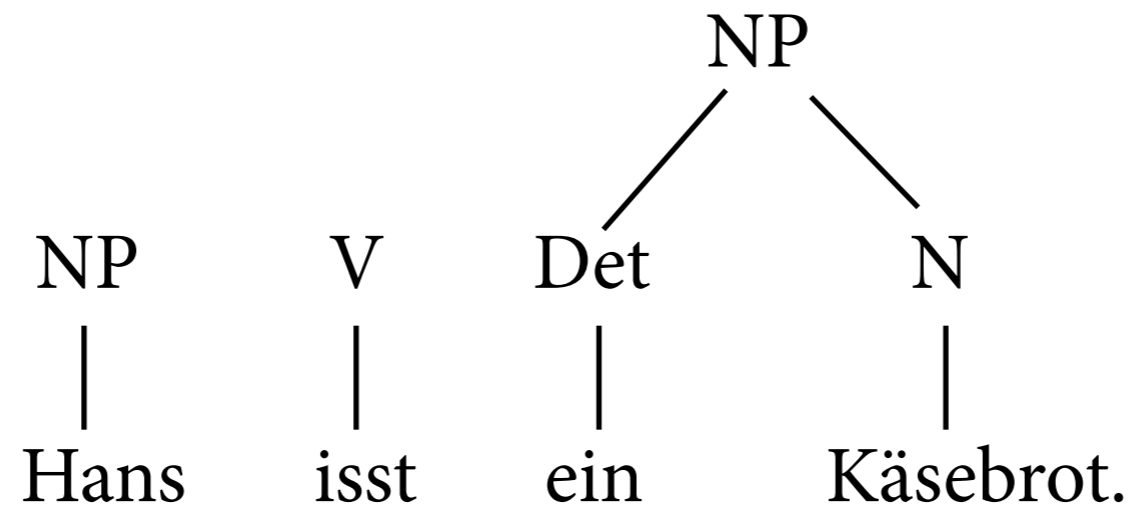
$\text{Det} \rightarrow \text{ein}$

$NP \rightarrow \text{Det } N$

$NP \rightarrow \text{Hans}$

$N \rightarrow \text{Käsebroten}$

$VP \rightarrow V NP$



# Bottom-Up-Parsing

$S \rightarrow NP VP$

$V \rightarrow \text{isst}$

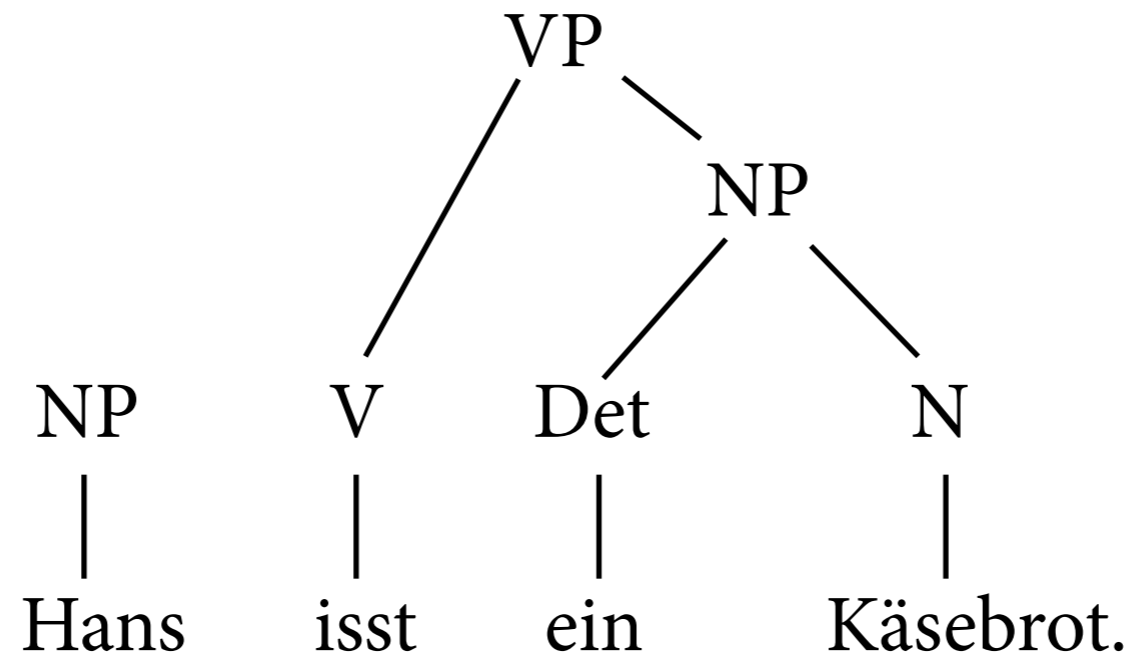
$\text{Det} \rightarrow \text{ein}$

$NP \rightarrow \text{Det } N$

$NP \rightarrow \text{Hans}$

$N \rightarrow \text{Käsebrod}$

$VP \rightarrow V NP$



# Bottom-Up-Parsing

$S \rightarrow NP VP$

$V \rightarrow \text{isst}$

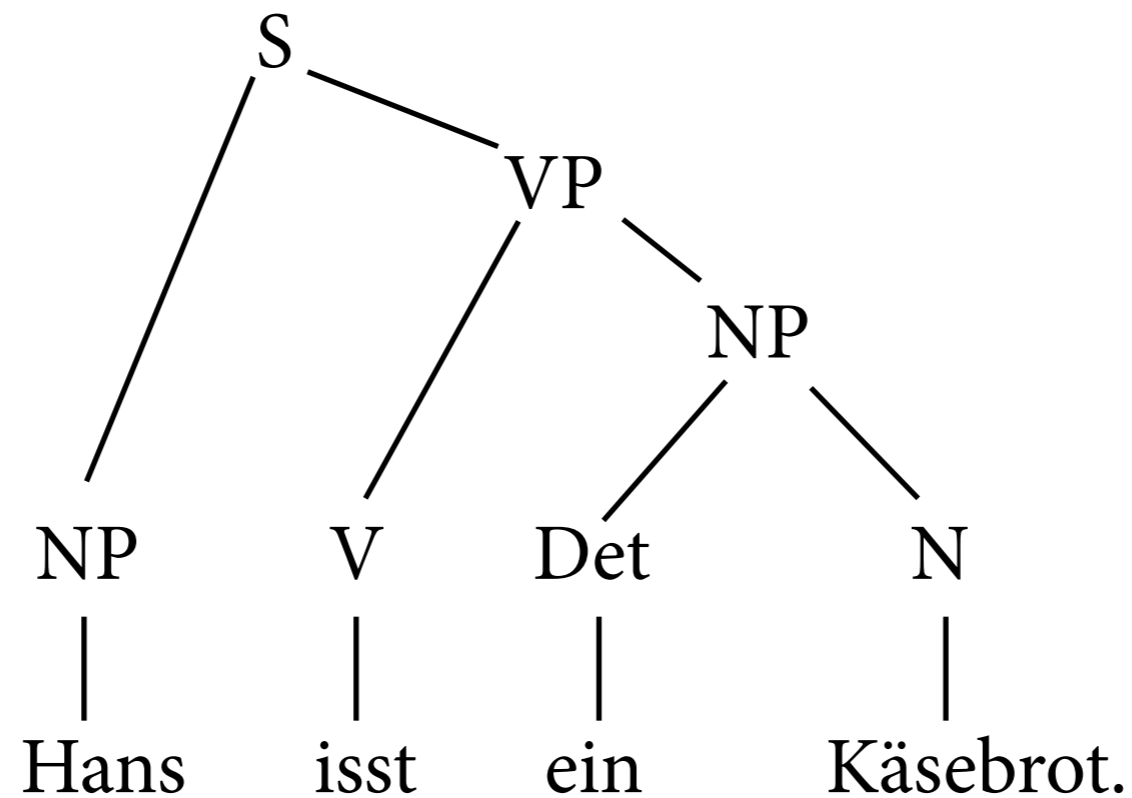
$\text{Det} \rightarrow \text{ein}$

$NP \rightarrow \text{Det } N$

$NP \rightarrow \text{Hans}$

$N \rightarrow \text{Käsebrod}$

$VP \rightarrow V NP$



# Bottom-up-Parsing

$S \rightarrow S S$

$S \rightarrow a$

a

a

a

# Bottom-up-Parsing

$S \rightarrow S S$

$S \rightarrow a$

$S$



$a$

$a$

$a$

# Bottom-up-Parsing

$S \rightarrow S S$

$S \rightarrow a$

$S$

|

$a$

$S$

|

$a$

$a$



# Bottom-up-Parsing

$S \rightarrow S S$

$S \rightarrow a$

$S$

|

$a$

$S$

|

$a$

$S$

|

$a$

# Bottom-up-Parsing

$S \rightarrow S S$

$S \rightarrow a$

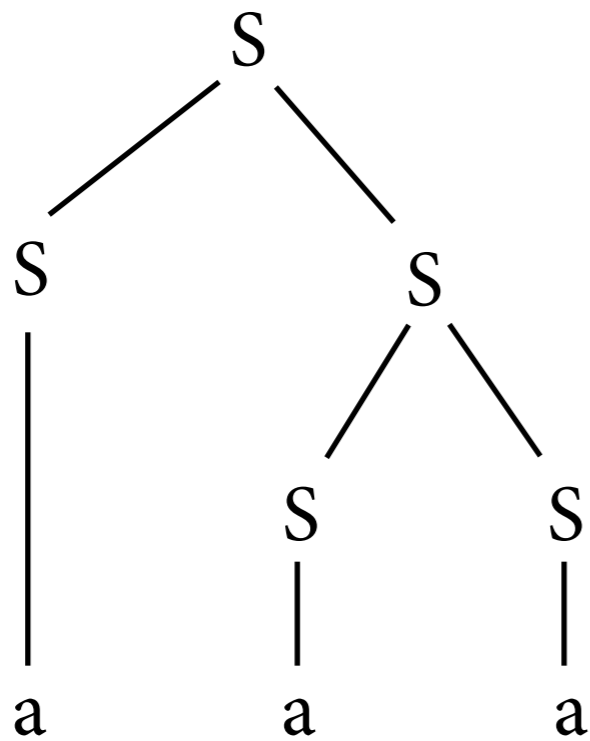
$S$   
|  
 $a$

$S$   
/ \  
 $S$   $S$   
| |  
 $a$   $a$

# Bottom-up-Parsing

$S \rightarrow S S$

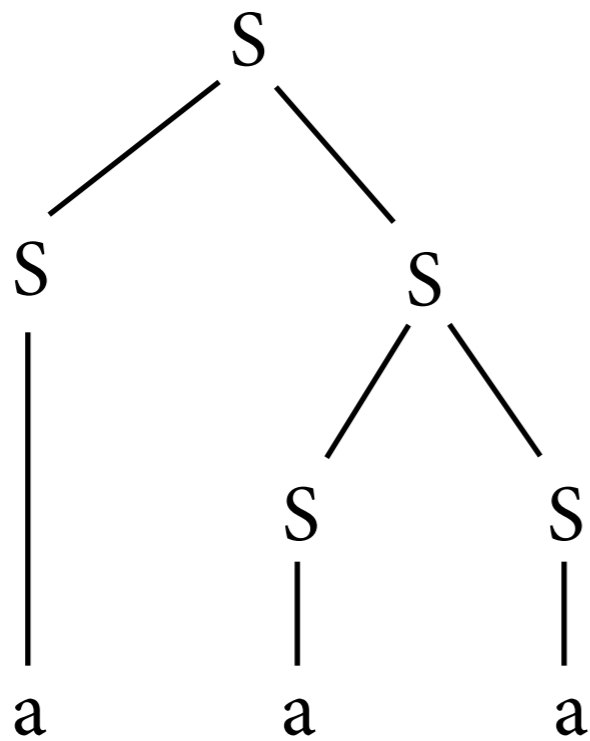
$S \rightarrow a$



# Bottom-up-Parsing

$S \rightarrow S S$

$S \rightarrow a$



$a$

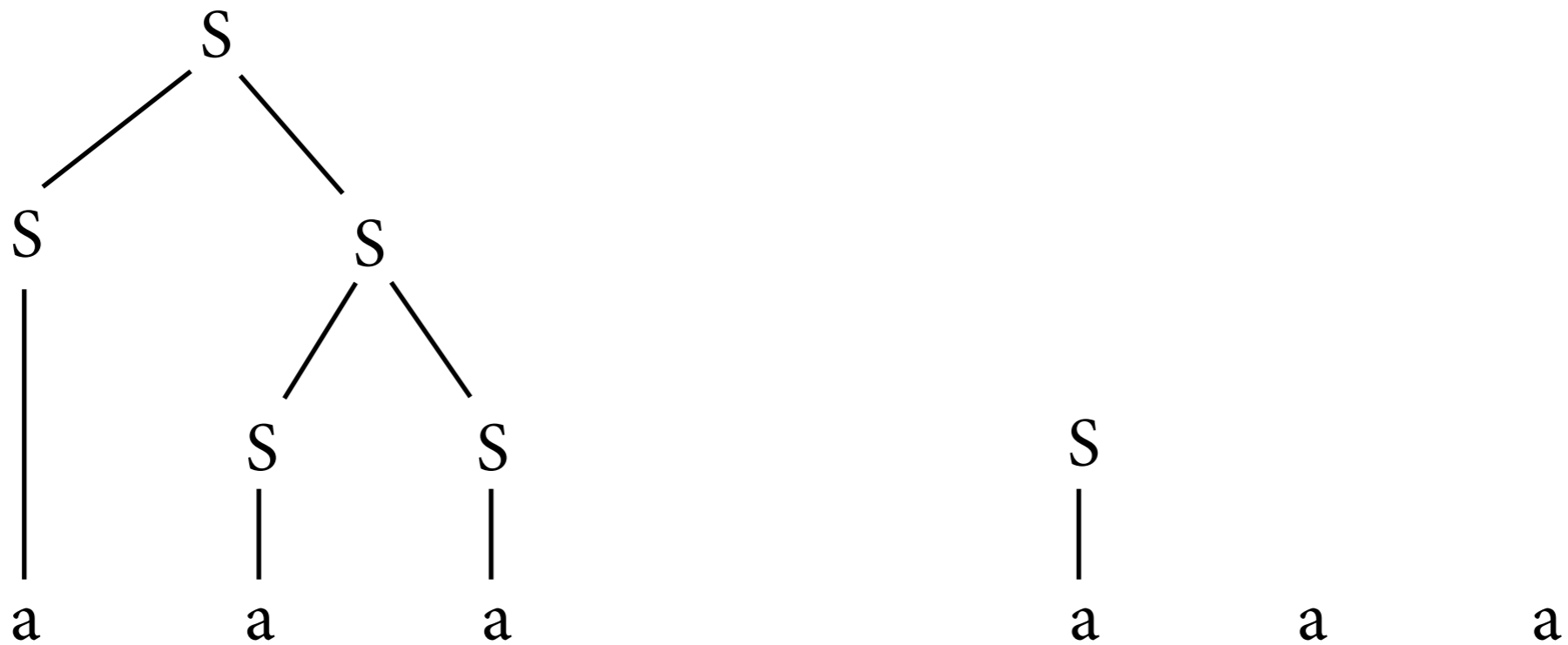
$a$

$a$

# Bottom-up-Parsing

$S \rightarrow S S$

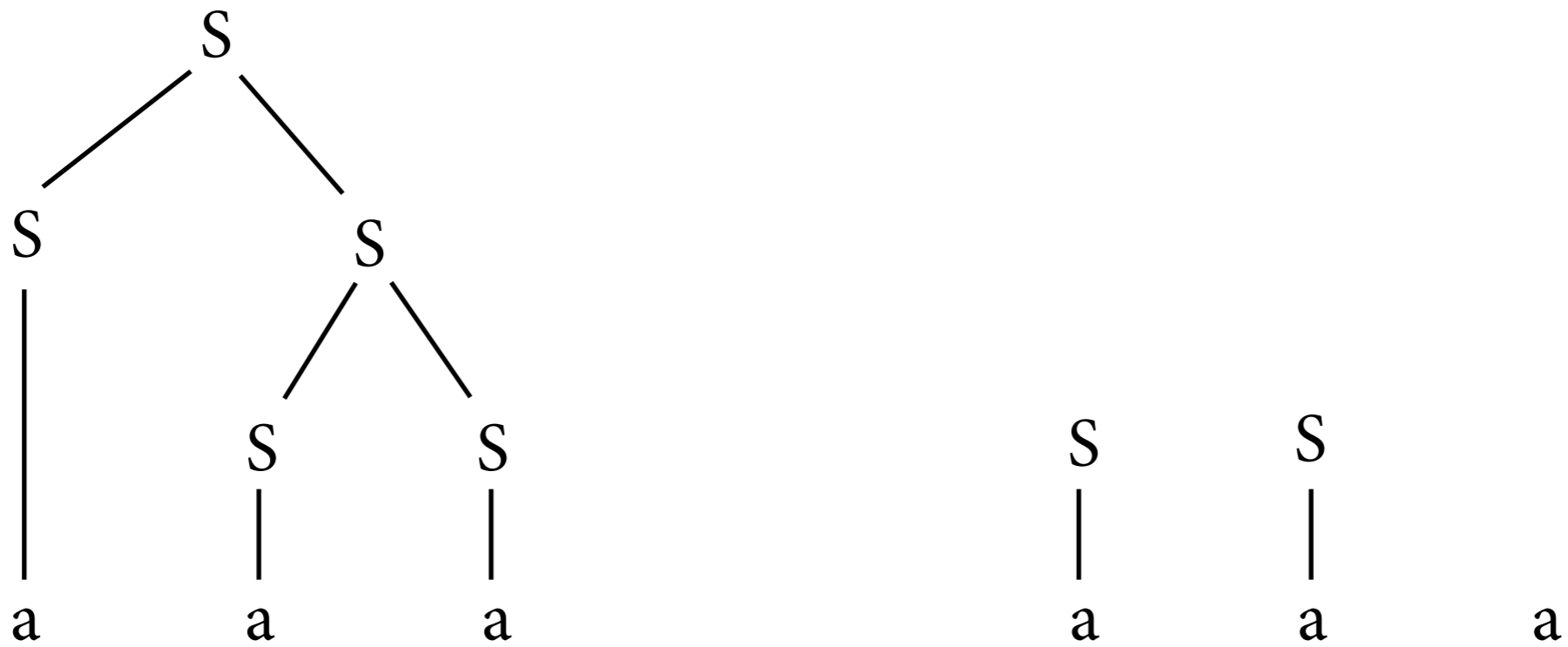
$S \rightarrow a$



# Bottom-up-Parsing

$S \rightarrow S S$

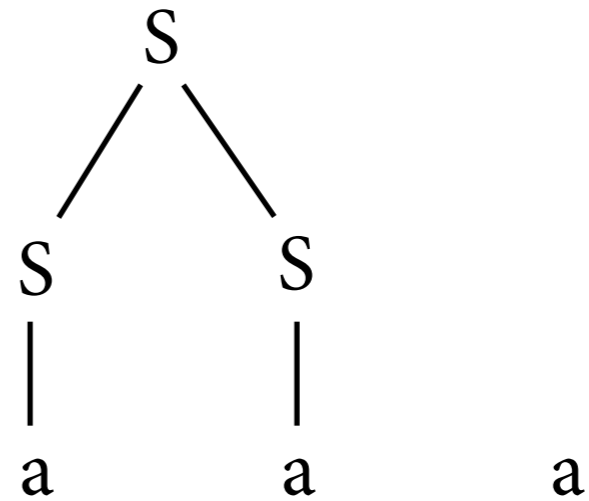
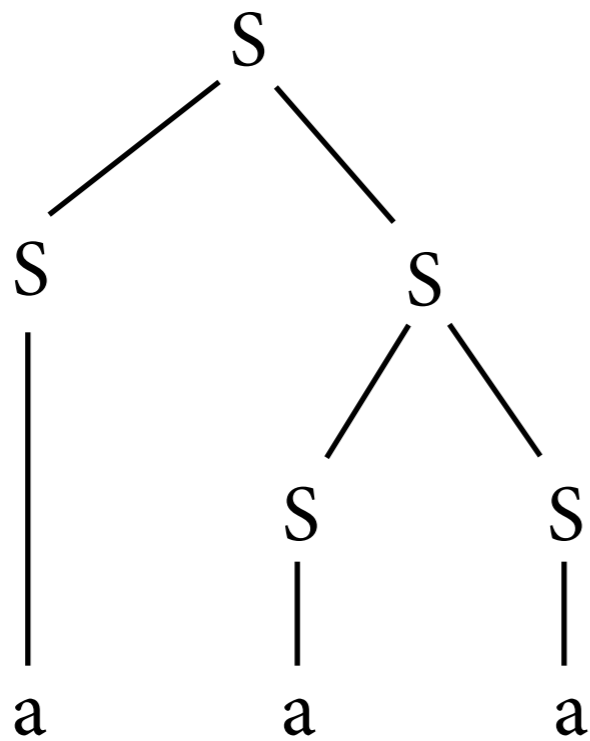
$S \rightarrow a$



# Bottom-up-Parsing

$S \rightarrow SS$

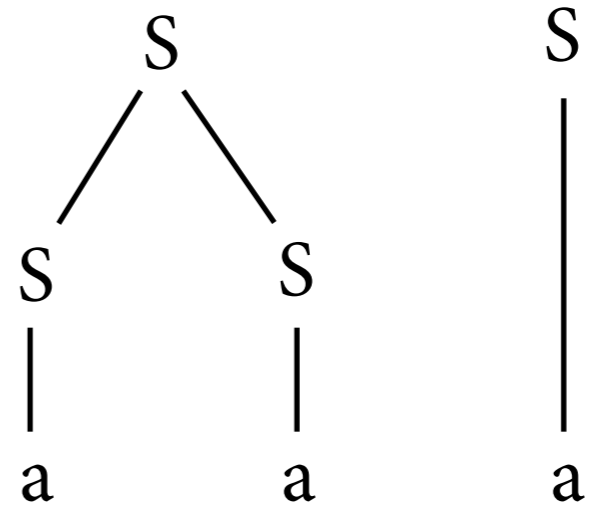
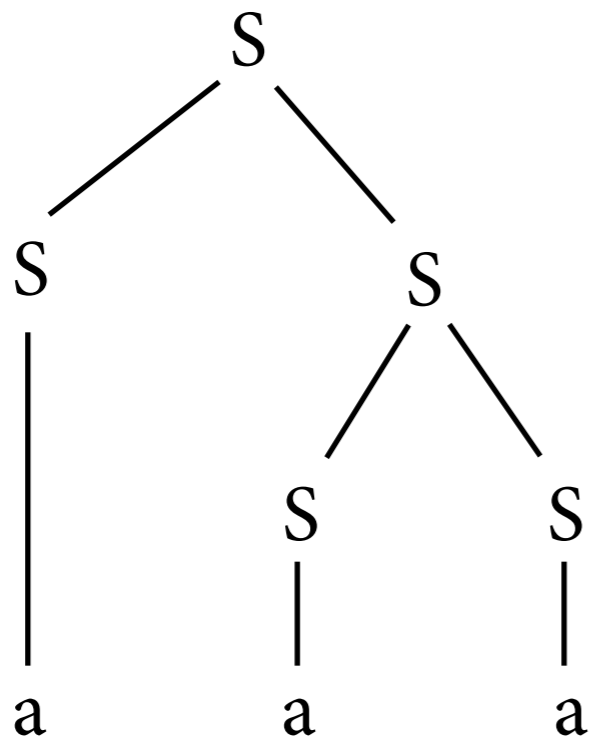
$S \rightarrow a$



# Bottom-up-Parsing

$S \rightarrow SS$

$S \rightarrow a$

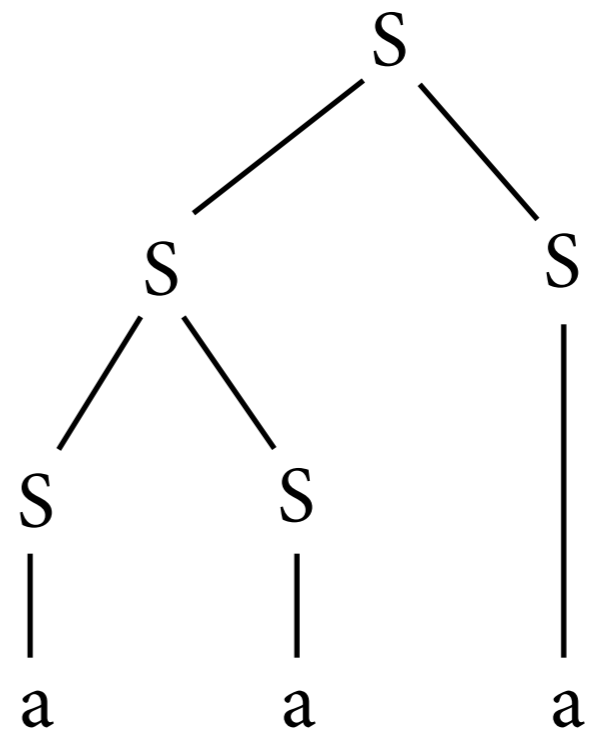
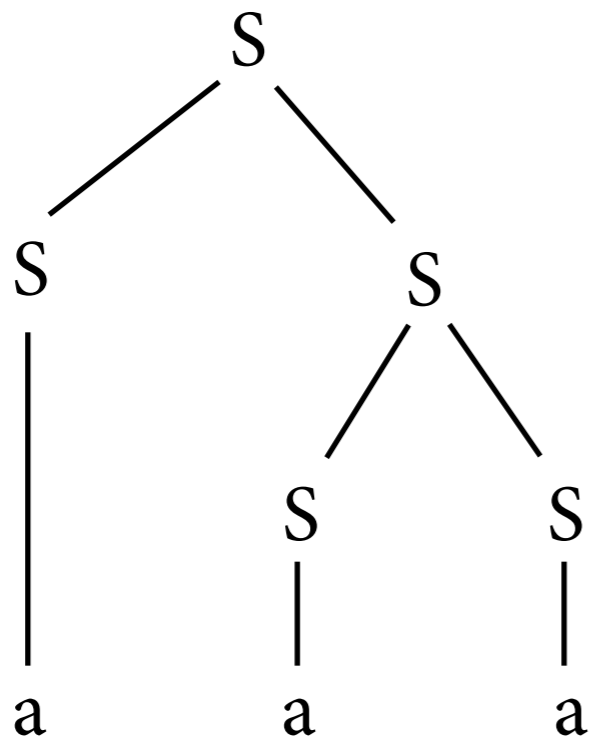




# Bottom-up-Parsing

$S \rightarrow SS$

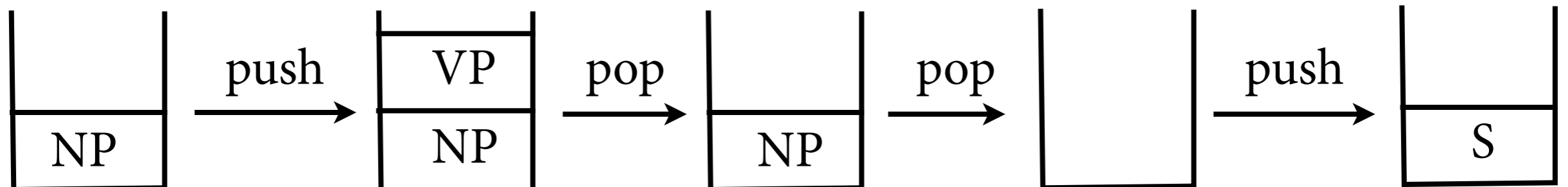
$S \rightarrow a$





# Shift-Reduce-Parsing

- Wir verwenden als Datenstruktur einen Stack von Terminal- und Nichtterminalsymbolen.
- Ein Stack ist eine Liste, in der ich nur an einem Ende (“oben”) lesen und schreiben kann.
- Unser Stack enthält die unverarbeiteten Terminal- und Nichtterminalsymbole.  
Schreibweise: in Stack  $s_1 s_2 s_3$  ist  $s_3$  “oben”.



# Shift-Reduce-Parsing

- Shift-Regel:  
 $(s, a \cdot w) \rightarrow (s \cdot a, w)$
- Reduce-Regel:  
 $(s \cdot w', w) \rightarrow (s \cdot A, w)$  falls  $A \rightarrow w'$  in  $P$
- Start:  $(\varepsilon, w)$
- Wende Regeln nichtdeterministisch an. Algorithmus sagt “ja”, wenn er Konfiguration  $(S, \varepsilon)$  erreicht (d.h. erreichen kann).

# Shift-Reduce: Beispiel

Shift:  $(s, a \cdot w) \rightarrow (s \cdot a, w)$

Reduce:  $(s \cdot w', w) \rightarrow (s \cdot A, w)$

Hans isst ein Käsebrot.

# Shift-Reduce: Beispiel

Shift:  $(s, a \cdot w) \rightarrow (s \cdot a, w)$   
Reduce:  $(s \cdot w', w) \rightarrow (s \cdot A, w)$

Hans isst ein Käsebrot.

$(\varepsilon, \text{Hans isst ein K.})$

# Shift-Reduce: Beispiel

Shift:  $(s, a \cdot w) \rightarrow (s \cdot a, w)$

Reduce:  $(s \cdot w', w) \rightarrow (s \cdot A, w)$

NP  
|  
Hans isst ein Käsebrötchen.

$(\epsilon, \text{Hans isst ein K.}) \rightarrow (\text{Hans}, \text{isst ein K.})$

# Shift-Reduce: Beispiel

Shift:  $(s, a \cdot w) \rightarrow (s \cdot a, w)$

Reduce:  $(s \cdot w', w) \rightarrow (s \cdot A, w)$

NP  
|  
Hans isst ein Käsebrot.

$(\epsilon, \text{Hans isst ein K.}) \rightarrow (\text{Hans}, \text{isst ein K.}) \rightarrow (\text{NP}, \text{isst ein K.})$



# Shift-Reduce: Beispiel

Shift:  $(s, a \cdot w) \rightarrow (s \cdot a, w)$   
Reduce:  $(s \cdot w', w) \rightarrow (s \cdot A, w)$

NP          V  
|          |  
Hans      isst    ein    Käsebröt.

$(\epsilon, \text{Hans isst ein K.}) \rightarrow (\text{Hans}, \text{isst ein K.}) \rightarrow (\text{NP}, \text{isst ein K.})$   
 $\rightarrow (\text{NP isst}, \text{ein K.})$

# Shift-Reduce: Beispiel

Shift:  $(s, a \cdot w) \rightarrow (s \cdot a, w)$

Reduce:  $(s \cdot w', w) \rightarrow (s \cdot A, w)$

NP          V  
|            |  
Hans        isst    ein    Käsebröt.

$(\epsilon, \text{Hans isst ein K.}) \rightarrow (\text{Hans}, \text{isst ein K.}) \rightarrow (\text{NP}, \text{isst ein K.})$   
 $\rightarrow (\text{NP isst}, \text{ein K.}) \rightarrow (\text{NP V}, \text{ein K.})$

# Shift-Reduce: Beispiel

Shift:  $(s, a \cdot w) \rightarrow (s \cdot a, w)$   
Reduce:  $(s \cdot w', w) \rightarrow (s \cdot A, w)$

|      |      |     |           |
|------|------|-----|-----------|
| NP   | V    | Det |           |
|      |      |     |           |
| Hans | isst | ein | Käsebröt. |

$(\epsilon, \text{Hans isst ein K.}) \rightarrow (\text{Hans}, \text{isst ein K.}) \rightarrow (\text{NP}, \text{isst ein K.})$   
 $\rightarrow (\text{NP isst}, \text{ein K.}) \rightarrow (\text{NP V}, \text{ein K.}) \rightarrow (\text{NP V ein}, \text{K.})$

# Shift-Reduce: Beispiel

Shift:  $(s, a \cdot w) \rightarrow (s \cdot a, w)$   
Reduce:  $(s \cdot w', w) \rightarrow (s \cdot A, w)$

|      |      |     |           |
|------|------|-----|-----------|
| NP   | V    | Det |           |
|      |      |     |           |
| Hans | isst | ein | Käsebröt. |

$(\epsilon, \text{Hans isst ein K.}) \rightarrow (\text{Hans}, \text{isst ein K.}) \rightarrow (\text{NP}, \text{isst ein K.})$   
 $\rightarrow (\text{NP isst}, \text{ein K.}) \rightarrow (\text{NP V}, \text{ein K.}) \rightarrow (\text{NP V ein}, \text{K.})$   
 $\rightarrow (\text{NP V Det}, \text{K.})$

# Shift-Reduce: Beispiel

Shift:  $(s, a \cdot w) \rightarrow (s \cdot a, w)$   
Reduce:  $(s \cdot w', w) \rightarrow (s \cdot A, w)$

|      |      |     |           |
|------|------|-----|-----------|
| NP   | V    | Det | N         |
|      |      |     |           |
| Hans | isst | ein | Käsebröt. |

$(\epsilon, \text{Hans isst ein K.}) \rightarrow (\text{Hans}, \text{isst ein K.}) \rightarrow (\text{NP}, \text{isst ein K.})$   
 $\rightarrow (\text{NP isst}, \text{ein K.}) \rightarrow (\text{NP V}, \text{ein K.}) \rightarrow (\text{NP V ein}, \text{K.})$   
 $\rightarrow (\text{NP V Det}, \text{K.}) \rightarrow (\text{NP V Det K.}, \epsilon)$

# Shift-Reduce: Beispiel

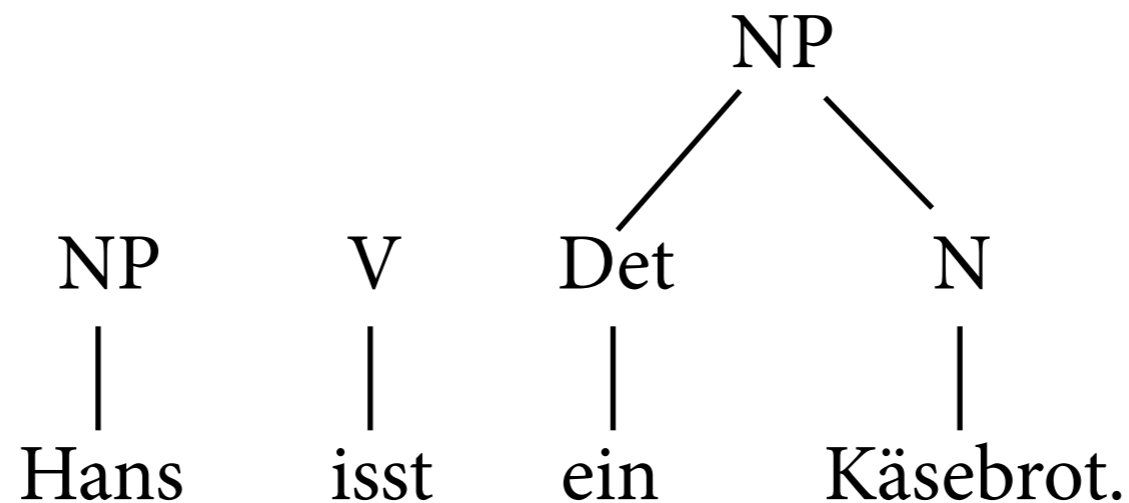
Shift:  $(s, a \cdot w) \rightarrow (s \cdot a, w)$   
Reduce:  $(s \cdot w', w) \rightarrow (s \cdot A, w)$

|      |      |     |           |
|------|------|-----|-----------|
| NP   | V    | Det | N         |
|      |      |     |           |
| Hans | isst | ein | Käsebröt. |

$(\epsilon, \text{Hans isst ein K.}) \rightarrow (\text{Hans}, \text{isst ein K.}) \rightarrow (\text{NP}, \text{isst ein K.})$   
 $\rightarrow (\text{NP isst}, \text{ein K.}) \rightarrow (\text{NP V}, \text{ein K.}) \rightarrow (\text{NP V ein}, \text{K.})$   
 $\rightarrow (\text{NP V Det}, \text{K.}) \rightarrow (\text{NP V Det K.}, \epsilon) \rightarrow (\text{NP V Det N}, \epsilon)$

# Shift-Reduce: Beispiel

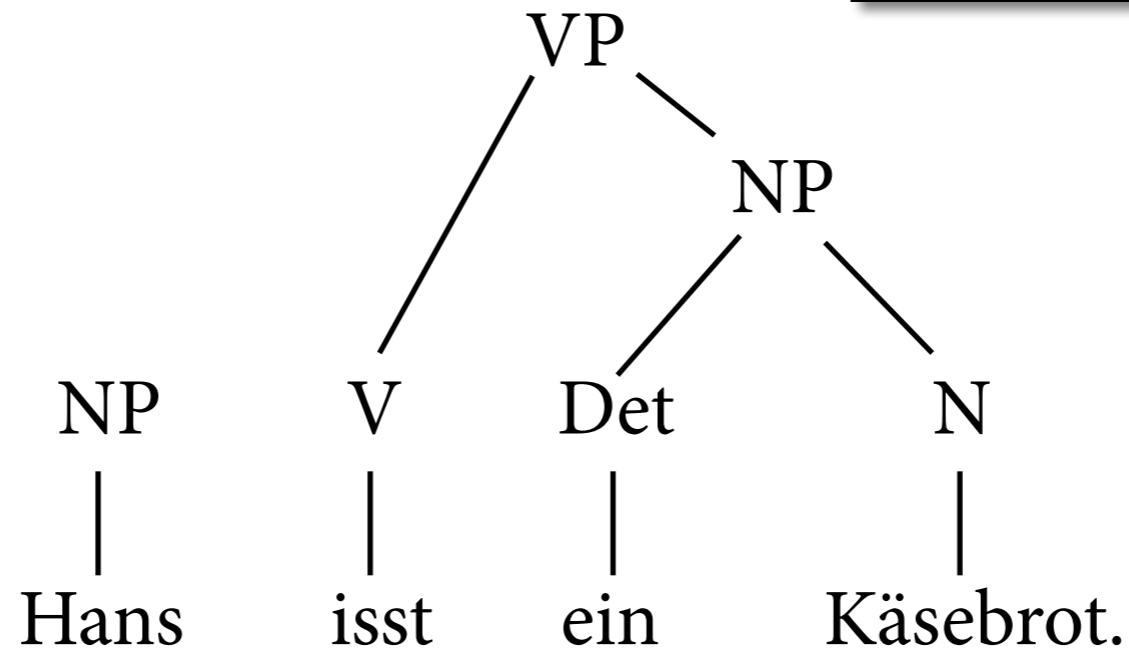
Shift:  $(s, a \cdot w) \rightarrow (s \cdot a, w)$   
Reduce:  $(s \cdot w', w) \rightarrow (s \cdot A, w)$



$(\epsilon, \text{Hans isst ein K.}) \rightarrow (\text{Hans}, \text{isst ein K.}) \rightarrow (\text{NP}, \text{isst ein K.})$   
 $\rightarrow (\text{NP isst}, \text{ein K.}) \rightarrow (\text{NP V}, \text{ein K.}) \rightarrow (\text{NP V ein}, \text{K.})$   
 $\rightarrow (\text{NP V Det}, \text{K.}) \rightarrow (\text{NP V Det K.}, \epsilon) \rightarrow (\text{NP V Det N}, \epsilon)$   
 $\rightarrow (\text{NP V NP}, \epsilon)$

# Shift-Reduce: Beispiel

Shift:  $(s, a \cdot w) \rightarrow (s \cdot a, w)$   
Reduce:  $(s \cdot w', w) \rightarrow (s \cdot A, w)$

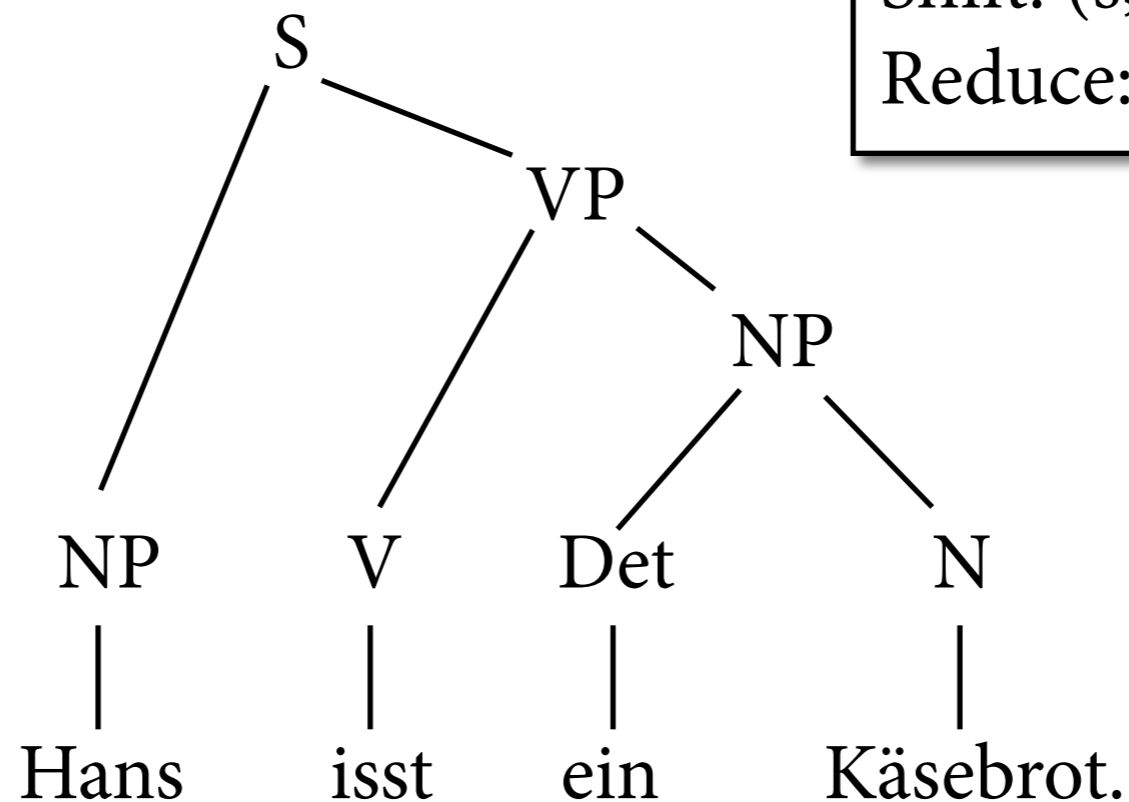


$(\epsilon, \text{Hans isst ein K.}) \rightarrow (\text{Hans}, \text{isst ein K.}) \rightarrow (\text{NP}, \text{isst ein K.})$   
 $\rightarrow (\text{NP isst}, \text{ein K.}) \rightarrow (\text{NP V}, \text{ein K.}) \rightarrow (\text{NP V ein}, \text{K.})$   
 $\rightarrow (\text{NP V Det}, \text{K.}) \rightarrow (\text{NP V Det K.}, \epsilon) \rightarrow (\text{NP V Det N}, \epsilon)$   
 $\rightarrow (\text{NP V NP}, \epsilon) \rightarrow (\text{NP VP}, \epsilon)$



# Shift-Reduce: Beispiel

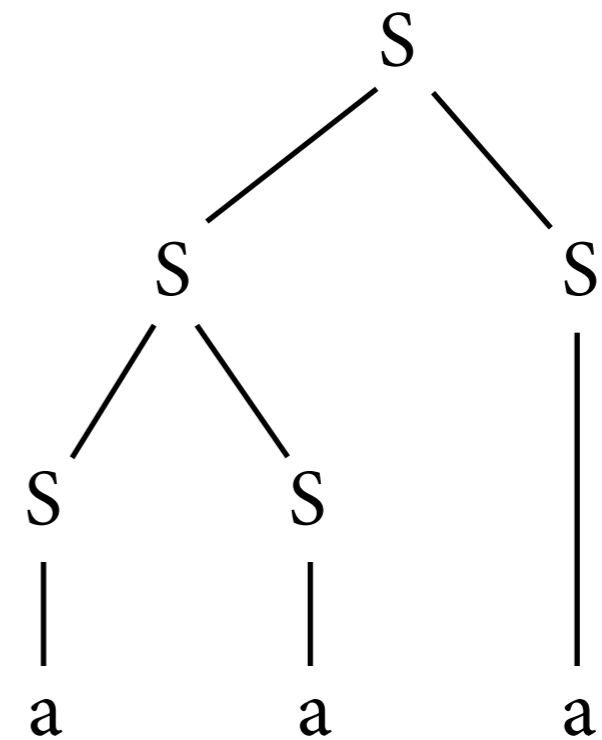
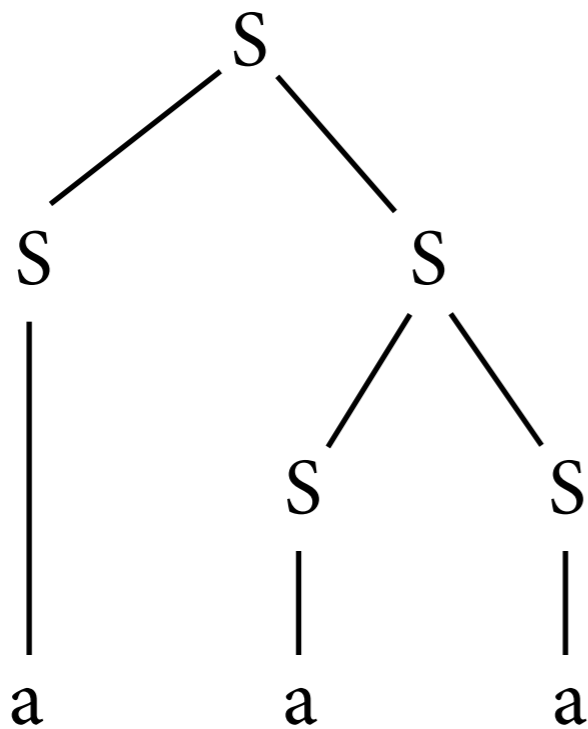
Shift:  $(s, a \cdot w) \rightarrow (s \cdot a, w)$   
Reduce:  $(s \cdot w', w) \rightarrow (s \cdot A, w)$



$(\epsilon, \text{Hans isst ein K.}) \rightarrow (\text{Hans}, \text{isst ein K.}) \rightarrow (\text{NP}, \text{isst ein K.})$   
 $\rightarrow (\text{NP isst}, \text{ein K.}) \rightarrow (\text{NP V}, \text{ein K.}) \rightarrow (\text{NP V ein}, \text{K.})$   
 $\rightarrow (\text{NP V Det}, \text{K.}) \rightarrow (\text{NP V Det K.}, \epsilon) \rightarrow (\text{NP V Det N}, \epsilon)$   
 $\rightarrow (\text{NP V NP}, \epsilon) \rightarrow (\text{NP VP}, \epsilon) \rightarrow (\text{S}, \epsilon)$

# Shift-Reduce: Beispiel

$S \rightarrow SS$        $S \rightarrow a$



$(\epsilon, aaa) \rightarrow (a, aa) \rightarrow (S, aa)$   
 $\rightarrow (Sa, a) \rightarrow (SS, a) \rightarrow (SSa, \epsilon)$   
 $\rightarrow (SSS, \epsilon) \rightarrow (SS, \epsilon) \rightarrow (S, \epsilon)$

$(\epsilon, aaa) \rightarrow (a, aa) \rightarrow (S, aa)$   
 $\rightarrow (Sa, a) \rightarrow (SS, a) \rightarrow (S, a)$   
 $\rightarrow (Sa, \epsilon) \rightarrow (SS, \epsilon) \rightarrow (S, \epsilon)$

# Shift-Reduce: Korrektheit

- Zeige:
  - ▶ wenn Shift-Reduce-Berechnung  $(s,w) \rightarrow^* (s',w')$  existiert,
  - ▶ dann gibt es kfG-Ableitung  $s'w' \Rightarrow^* sw$ .
- Daraus folgt dann Korrektheit:
  - ▶ SR-Erkennen behauptet  $w \in L(G)$
  - ▶ gdw  $(\varepsilon, w) \rightarrow^* (S, \varepsilon)$
  - ▶ daher  $S \Rightarrow^* w$  (s.o.)
  - ▶ d.h.  $w \in L(G)$  ist wahr.

$$(s, w) \rightarrow^* (s', w') \Rightarrow s'w' \Rightarrow^* sw$$

Induktion über Länge  $n$  der Berechnung  
(d.h. Anzahl der Shift- und Reduce-Schritte)

- $n = 0$ :  $sw \Rightarrow^* sw$  trivial
- $n \rightarrow n+1$ : Betrachte ersten Schritt von  $(s, w) \rightarrow^* (s', w')$ 
  - ▶ erster Schritt shift:  $(s, aw) \rightarrow (sa, w) \rightarrow^* (s', w')$   
es gilt:  $s'w' \underset{IA}{\Rightarrow^*} sa \cdot w \Rightarrow^* s \cdot aw$
  - ▶ erster Schritt reduce:  $(sw', w) \rightarrow (sA, w) \rightarrow^* (s', w')$   
es gilt:  $s'w' \underset{IA}{\Rightarrow^*} sAw \Rightarrow sw'w$

# Shift-Reduce: Vollständigkeit

- Zeige:
  - ▶ wenn kfG-Ableitung  $A \Rightarrow^* w$  existiert mit  $w \in T^*$ ,
  - ▶ dann gibt es SR-Berechnung  $(\varepsilon, w) \rightarrow^* (A, \varepsilon)$
- Daraus folgt dann Vollständigkeit:
  - ▶  $w \in L(G)$  gilt
  - ▶ gdw kfG-Ableitung  $S \Rightarrow^* w$  existiert mit  $w \in T^*$
  - ▶ dann gibt es SR-Berechnung  $(\varepsilon, w) \rightarrow^* (S, \varepsilon)$  (s.o.)
  - ▶ also behauptet SR-Erkennen, dass  $w \in L(G)$  gilt.

$$A \Rightarrow^* w \quad \Rightarrow \quad (\varepsilon, w) \rightarrow^* (A, \varepsilon)$$

Induktion über Länge  $n$  der kfG-Ableitung  
 (für CNF-Grammatiken; allgemeiner Fall nur mehr Schreibarbeit)

- $n = 1$ : d.h.  $A \rightarrow w \in P$ , also  $(\varepsilon, w) \xrightarrow{s} (w, \varepsilon) \xrightarrow{r} (A, \varepsilon)$
- $n \rightarrow n+1$ : d.h.  $A \Rightarrow B C \Rightarrow^* B w_2 \Rightarrow^* w_1 w_2 = w$ 
  - ▶ IA1:  $(\varepsilon, w_1) \rightarrow^* (B, \varepsilon)$ ; daher auch  $(\varepsilon, w_1 v) \rightarrow^* (B, v)$
  - ▶ IA2:  $(\varepsilon, w_2) \rightarrow^* (C, \varepsilon)$ ; daher auch  $(u, w_2) \rightarrow^* (u C, \varepsilon)$
  - ▶ also:  $(\varepsilon, w_1 w_2) \xrightarrow{IV1} (B, w_2) \xrightarrow{IV2} (B C, \varepsilon) \xrightarrow{r} (A, \varepsilon)$

# Probleme: Übersicht

- Parsertypen haben komplementäre Probleme:

|                 | top-down                                       | bottom-up                                  |
|-----------------|------------------------------------------------|--------------------------------------------|
| Regeln raten    | $A \rightarrow w_1$<br>vs. $A \rightarrow w_2$ | $A \rightarrow w$<br>vs. $B \rightarrow w$ |
| Zerlegung raten | String aufteilen                               | Shift vs. Reduce<br>entscheiden            |

- Allgemein nicht zu vermeiden (Ambiguität).

# Zusammenfassung

- Bottom-up vs. Top-down:
  - ▶ top-down: Recursive Descent
  - ▶ bottom-up: Shift-Reduce
- Korrektheit und Vollständigkeit
- Beide Parsertypen haben komplementäre Probleme beim Parsing verschiedener Grammatiken.