Grundlagen der Theoretischen Informatik

4. Kellerautomaten und kontextfreie Sprachen (VI)

13.06.2018

Viorica Sofronie-Stokkermans

e-mail: sofronie@uni-koblenz.de

Übersicht

1. Motivation
2. Terminologie
3. Endliche Automaten und reguläre Sprachen
4. Kellerautomaten und kontextfreie Sprachen
5. Turingmaschinen und rekursiv aufzählbare Sprachen
6. Berechenbarkeit, (Un-)Entscheidbarkeit
7. Komplexitätsklassen P und NP

Bis jetzt

- Push-Down-Automaten • Von PDA akzeptierte Sprache:
  - über **finale Zustände**
  - über **leeren Keller**
- Gleichmächtigkeit: PDAs und cf-Grammatiken

Abschlusseigenschaften

Abschlusseigenschaften

Theorem (Abschlusseigenschaften von $\mathcal{L}_2$)

$\mathcal{L}_2$ ist abgeschlossen gegen:

- Vereinigung $\cup$
- Konkatenation $\circ$
- Kleene-Stern *

Abschlusseigenschaften

Theorem (Abschlusseigenschaften von $\mathcal{L}_2$)

$\mathcal{L}_2$ ist abgeschlossen gegen:

- Vereinigung $\cup$
- Konkatenation $\circ$
- Kleene-Stern *

Beweis

Seien

$$G_i = (V_i, T_i, R_i, S_i) \qquad (i \in \{1, 2\})$$

zwei cf-Grammatiken mit $V_1 \cap V_2 = \emptyset$.

Sei

$$L_i = L(G_i)$$

Abschlusseigenschaften

Beweis (Forts.)

zu $\cup$:

$$(V_1 \cup V_2 \cup \{S_{neu}\},\ T_1 \cup T_2,\ R_1 \cup R_2 \cup \{S_{neu} \to S_1 \mid S_2\},\ S_{neu})$$

erzeugt gerade $L_1 \cup L_2$

zu $\circ$:

$$(V_1 \cup V_2 \cup \{S_{neu}\},\ T_1 \cup T_2,\ R_1 \cup R_2 \cup \{S_{neu} \to S_1 S_2\},\ S_{neu})$$

erzeugt gerade $L_1 \circ L_2$

zu *:

$$(V_1 \cup \{S_{neu}\},\ T_1,\ R_1 \cup \{S_{neu} \to S_1 S_{neu} \mid \varepsilon\},\ S_{neu})$$

erzeugt gerade L_1^*. $\square$

Abschlusseigenschaften

Theorem (Abschlusseigenschaften von $\mathcal{L}_2$)

$\mathcal{L}_2$ ist **nicht** abgeschlossen gegen:

- Durchschnitt $\cap$
- Komplement $\neg$

Abschlusseigenschaften

Beweis Zu „$\cap$“:

$$\begin{aligned} L_1 &= \{a^n b^n c^m \mid n, m \in \mathbb{N}_+\} \\ L_2 &= \{a^m b^n c^n \mid n, m \in \mathbb{N}_+\} \end{aligned}$$

wird erzeugt von $G_i = (\{S, S', T\}, \{a, b, c\}, R_i, S)$ mit

$$\begin{aligned} R_1 = \{ \quad & S \to S'T \\ & S' \to aS'b \mid ab \\ & T \to cT \mid c\} \end{aligned}$$

$$\begin{aligned} R_2 = \{ \quad & S \to TS' \\ & S' \to bS'c \mid bc \\ & T \to aT \mid a\} \end{aligned}$$

Sowohl L_1 als auch L_2 sind cf, **nicht** aber $L_1 \cap L_2 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$.

Abschlusseigenschaften

Beweis Zu „$\neg$“:

Angenommen, $\mathcal{L}_2$ wäre abgeschlossen gegen $\neg$.

Wegen

$$L_1 \cap L_2 = \neg(\neg L_1 \cup \neg L_2)$$

wäre $\mathcal{L}_2$ dann auch abgeschlossen gegen $\cap$ – **Widerspruch** $\square$

Wortprobleme

Wortprobleme

Problem

Gegeben: eine cf-Grammatik G, so dass $L(G)$ eine Sprache ist über Σ, und ein Wort $w \in \Sigma^*$

Frage: Ist $w \in L(G)$?

Wortproblem

Lösung des Wortproblems für $\mathcal{L}_3$

Gegeben eine rechtslineare Grammatik G, so dass $L(G)$ eine Sprache ist über Σ, und ein Wort $w \in \Sigma^*$.

- Konstruiere aus G einen ε-NDEA A_1.
- Konstruiere aus A_1 einen NDEA A_2.
- Konstruiere aus A_2 einen DEA A_3.
- Probiere aus, ob A_3 das Wort w akzeptiert.

  Dazu braucht der Automat A_3 genau $|w|$ Schritte.

Wortproblem

Das Wortproblem für $\mathcal{L}_2$

- Zu jeder cf-Grammatik G kann man einen PDA konstruieren
- Aber ein Pushdown-Automat kann ε-Übergänge machen, in denen er das Wort nicht weiter liest.
- **Wie kann man dann garantieren, dass der Automat in endlich vielen Schritten das Wort w zu Ende gelesen hat?**
- Deshalb: verwende anderes Verfahren:
  Cocke-Younger-Kasami-Algorithmus (CYK-Algorithmus)
  Auch: Chart-Parsing

Chart-Parsing

Gegeben: Ein Wort

$$w = a_1 \dots a_n$$

Idee

- Prinzip der dynamischen Programmierung
- 1.: Ermittle woraus sich die einstelligen Teilworte ableiten lassen
  2.: Ermittle woraus sich die zweistelligen Teilworte ableiten lassen
  ...
  n.: Ermittle woraus sich die n-stelligen Teilworte (w selbst) ableiten lassen

Chart-Parsing

Beispiel: Die Grammatik $G = (\{S\}, \{a, b\}, R, S)$ mit

$$R = \{\ S \to aSa \mid bSb \mid aa \mid bb\ \}$$

erzeugt die Sprache $\{vv^R \mid v \in \{a, b\}^+\}$

Betrachten wir das Wort $w = abbaabba$.

Was sind mögliche letzte Schritte von Ableitungen, die zu w geführt haben können?

Wir merken uns alle möglichen einzelnen Ableitungsschritte in einem Chart, um Mehrfacharbeit zu vermeiden.

Wenn das Wort w in der Sprache $L(G)$ ist,
enthält der Chart am Ende eine mit S markierte Kante, die vom ersten bis zum letzten Knoten reicht.

Chart-Parsing

Beispiel (Forts.)

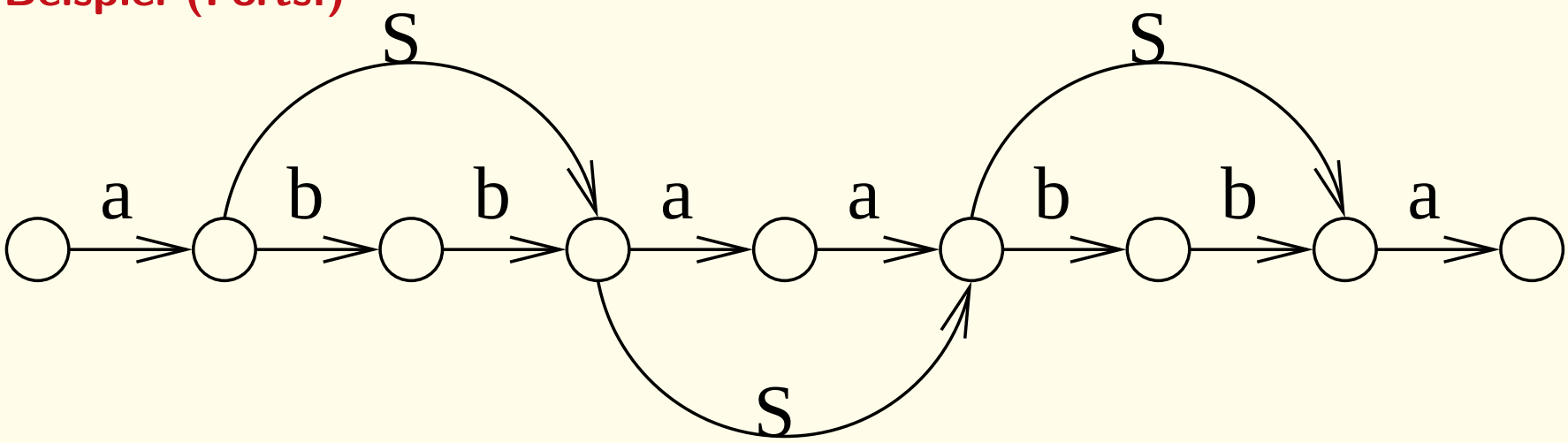

Chart-Parsing

Beispiel (Forts.)

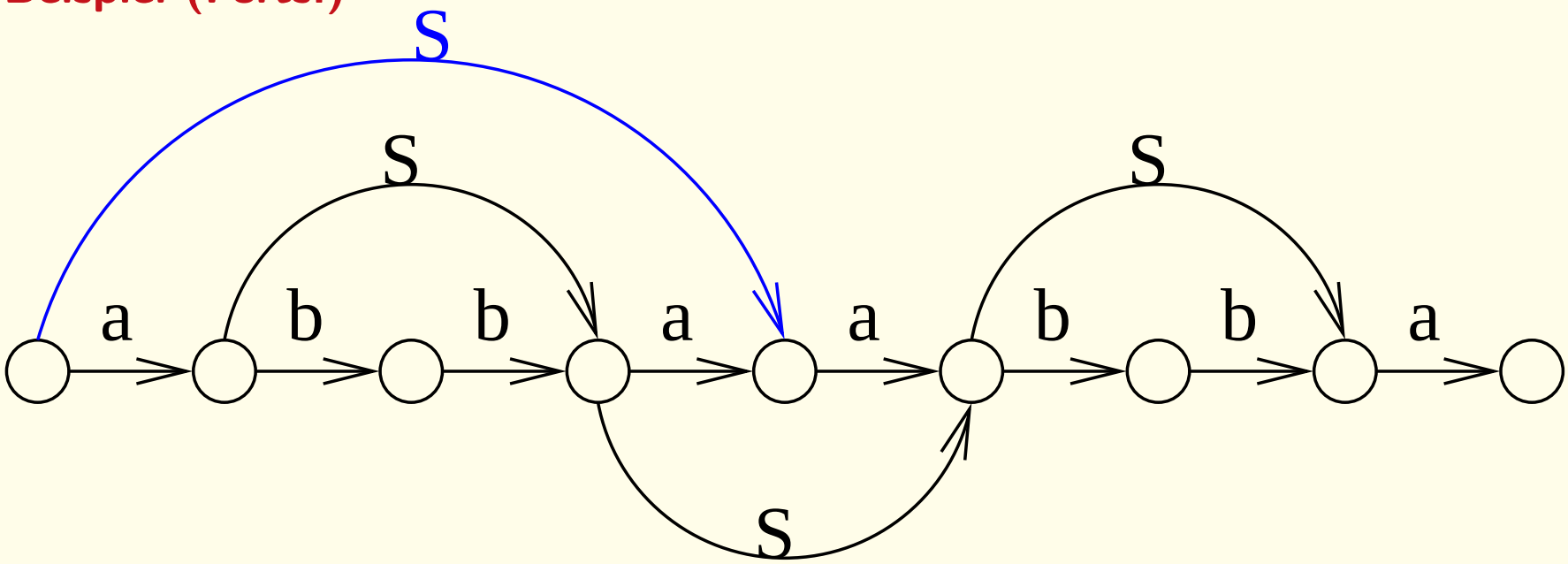

Chart-Parsing

Beispiel (Forts.)

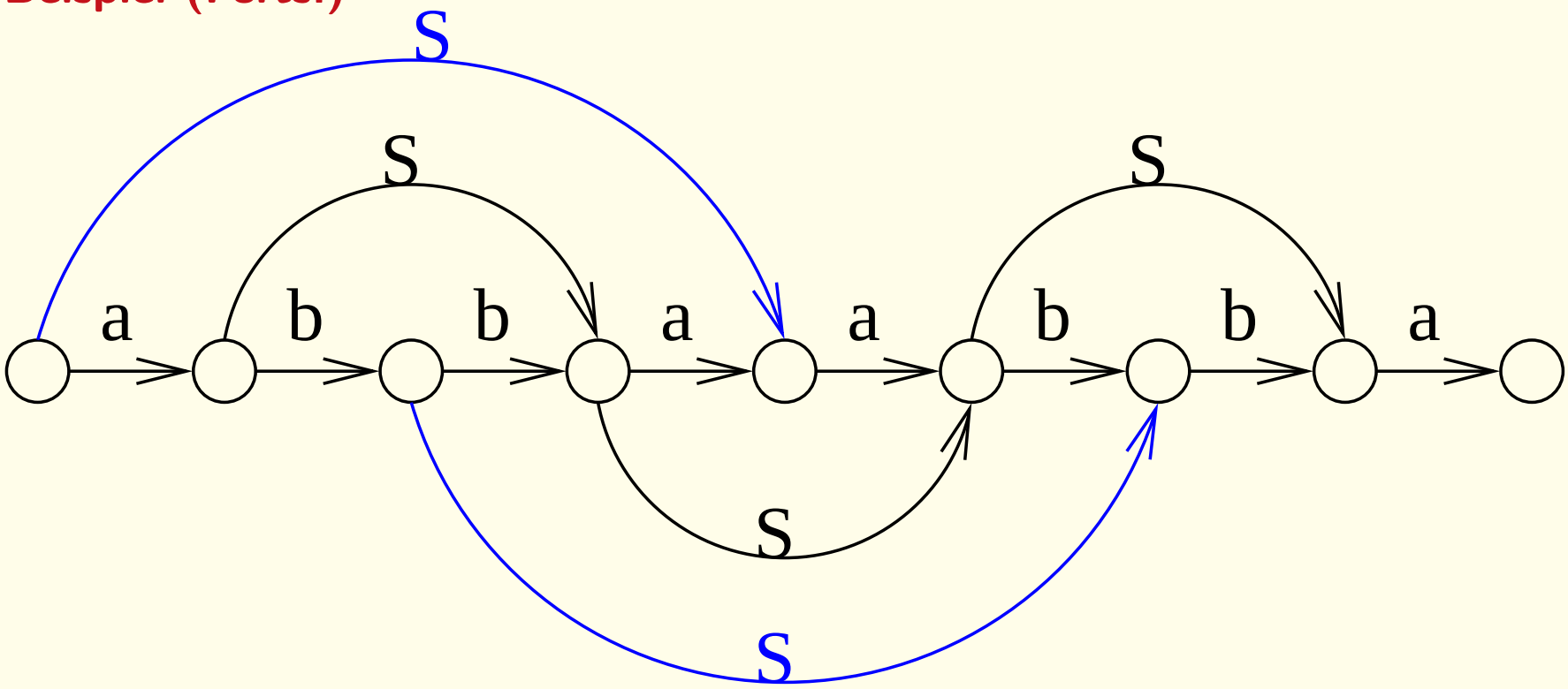

Chart-Parsing

Beispiel (Forts.)

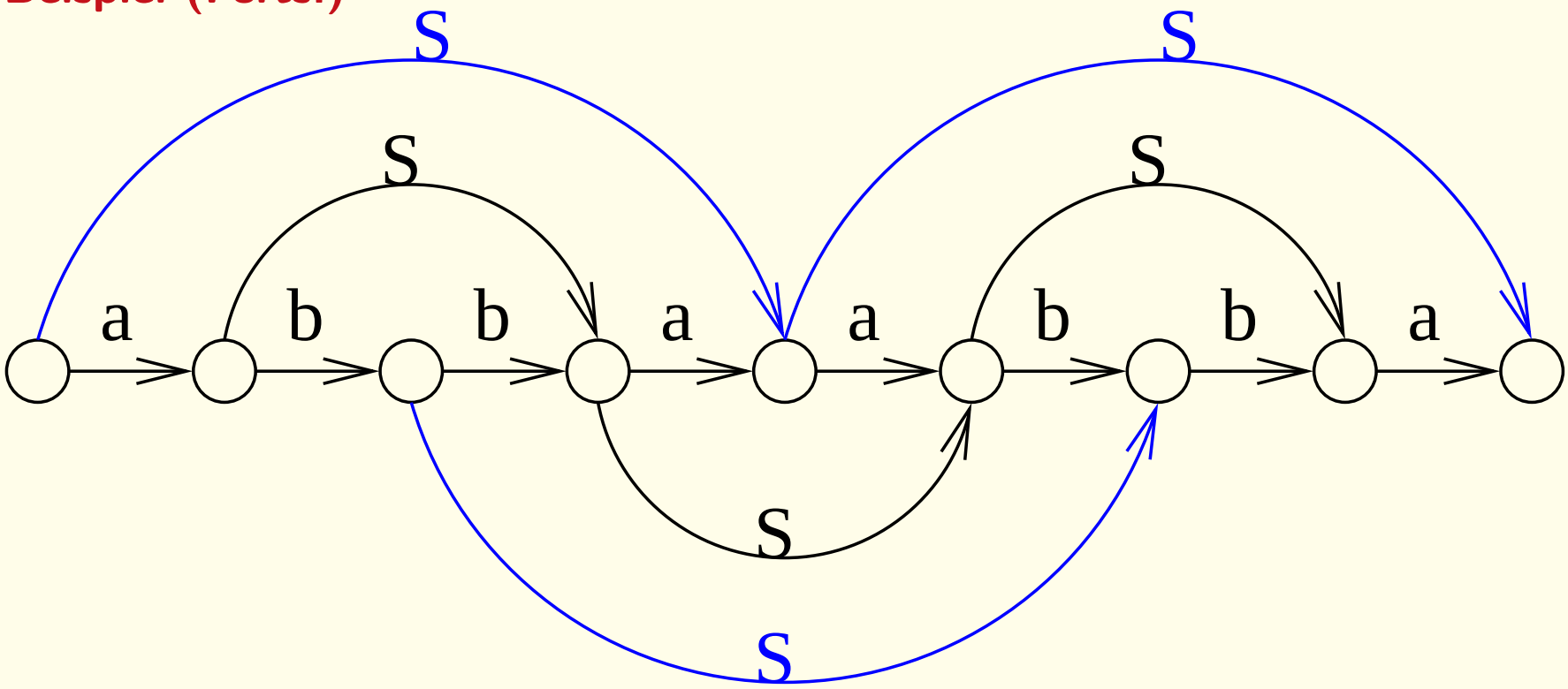

Chart-Parsing

Beispiel (Forts.)

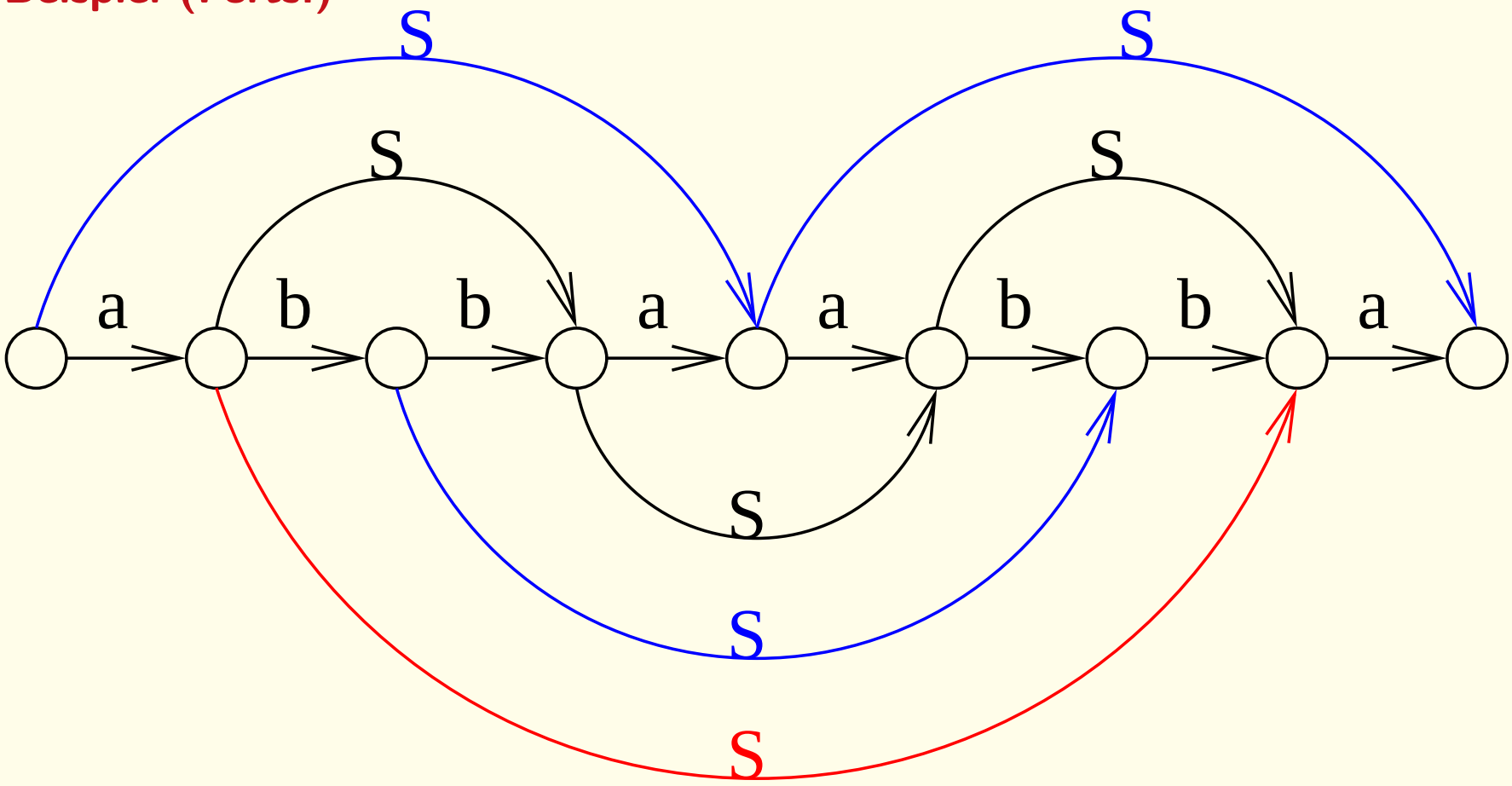

Chart-Parsing

Beispiel (Forts.)

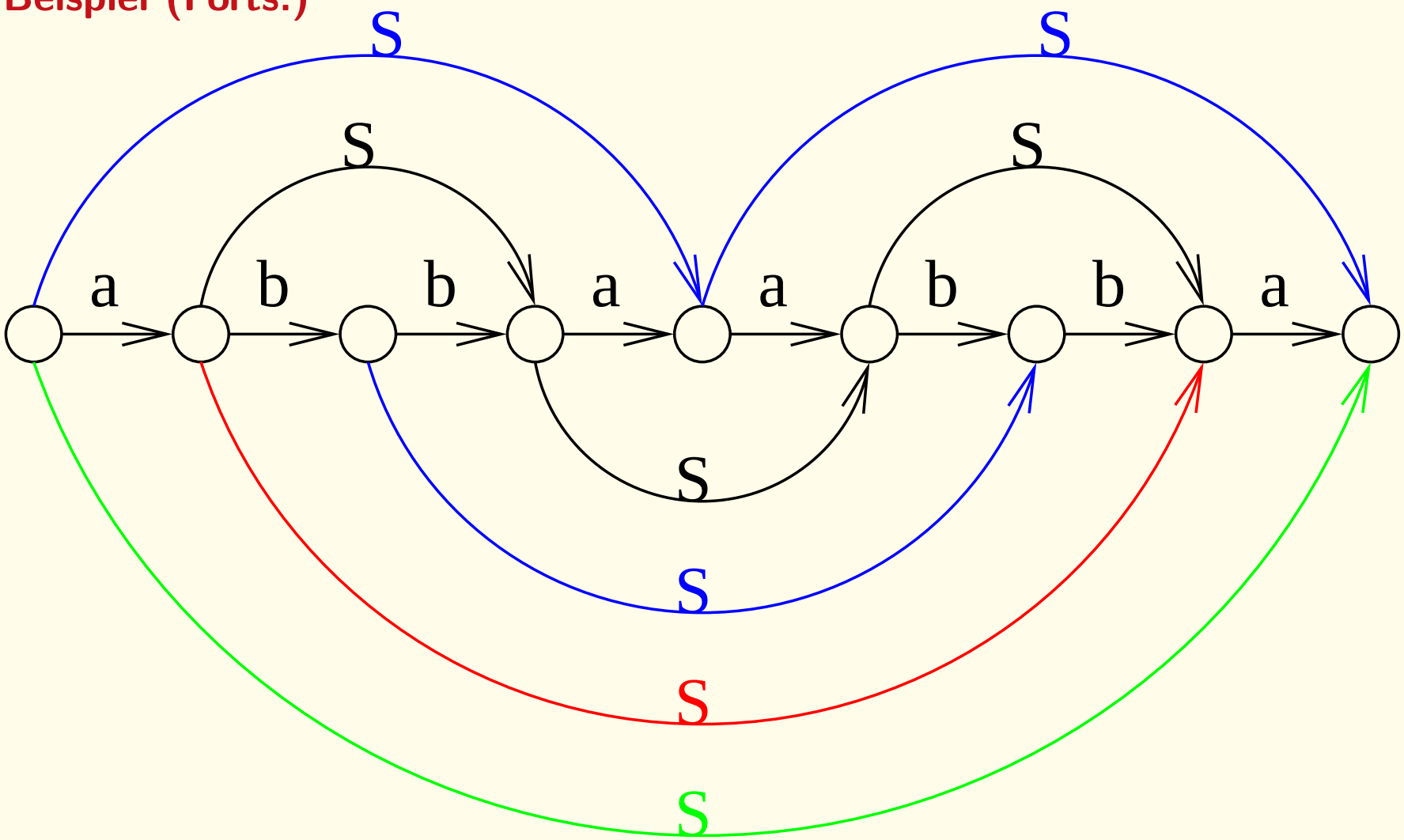

Chart-Parsing

Zur Vereinfachung

Wir fordern: Grammatik ist in **Chomsky-Normalform**

Dann:

Immer nur **zwei** benachbarte Kanten betrachten, um herauszufinden, ob darüber eine neue Kante eingefügt werden kann.

Chart-Parsing

Beispiel (Forts.) Grammatik in CNF, die dieselbe Sprache wie oben erzeugt: $G = (\{S, S_a, S_b, A, B\}, \{a, b\}, R, S\}$ mit

$$
\begin{aligned}
R = \{ \quad & S \to AS_a \mid BS_b \mid AA \mid BB \\
& S_a \to SA \\
& S_b \to SB \\
& A \to a \\
& B \to b\}
\end{aligned}
$$

Chart-Parsing

Darstellung als Array

Für eine Kante, die den i. bis j. Buchstaben überspannt und mit A markiert ist, steht im $[i, j]$-Element des Arrays die Eintragung A.

Definition (M $*$ N)

Sei $L = L(G)$ kontextfrei, und $G = (V, T, R, S)$ in Chomsky-Normalform. Mit $M, N \subseteq V$ sei

$$M * N := \{A \in V \mid \exists B \in M, \exists C \in N : \ A \to BC \in R\}$$

Chart-Parsing

Definition ($w_{i,j}$, $V_{i,j}$)

Sei $w = a_1 \dots a_n$ mit $a_i \in \Sigma$.

Dann:

- $w_{i,j} := a_i \dots a_j$ ist das Fragment von w vom i-ten bis zum j-ten Buchstaben
- $V_{i,j} := \{A \in V \mid A \Longrightarrow_G^* w_{i,j}\}$

Chart-Parsing

Sei $w = a_1 \dots a_n$, $a_i \in \Sigma$, d.h. $|w| = n$. Dann gilt:

1. $V_{i,i} = \{A \in V \mid A \to a_i \in R\}$
2. $V_{i,k} = \bigcup_{j=i}^{k-1} V_{i,j} * V_{j+1,k}$ für $1 \leq i < k \leq n$

Beachte:

Die Grammatik muss in Chomsky-Normalform sein!

Chart-Parsing

Beweis.

1. $V_{i,i} = \{A \in V \mid A \Longrightarrow_G^* a_i\} = \{A \in V \mid A \to a_i \in R\}$, da G in CNF ist.

   $A \in V_{i,k}$ mit $1 \leq i < k \leq n$

   gdw $A \Longrightarrow_G^* a_i \ldots a_k$

   gdw $\exists j,\ i \leq j < k :\ \exists B, C \in V : A \Longrightarrow BC$, und
   $B \Longrightarrow_G^* w_{i,j} \neq \varepsilon$
   und $C \Longrightarrow_G^* w_{j+1,k} \neq \varepsilon$ (da G in CNF ist)

   gdw $\exists j,\ i \leq j < k :\ \exists B, C \in V : A \Longrightarrow BC$
   und $B \in V_{i,j}$ und $C \in V_{j+1,k}$

2. gdw $\exists j,\ i \leq j < k : A \in V_{i,j} * V_{j+1,k}$

CYK-Algorithmus (Cocke-Younger-Kasami)

Algorithmus

Input sei eine Grammatik G in CNF und ein Wort $w = a_1 \dots a_n \in \Sigma^*$.

(i) **for** $i := 1$ *to* n **do** `/* Regeln A → a eintragen */`

$V_{i,i} := \{A \in V \mid A \to a_i \in R\}$

(ii) **for** $h := 1$ *to* $n-1$ **do**

for $i := 1$ *to* $n-h$ **do**

$$V_{i,i+h} = \bigcup_{j=i}^{i+h-1} V_{i,j} * V_{j+1,i+h}$$

(iii) **if** $S \in V_{1,n}$ **then return** `Ausgabe` $w \in L(G)$

else return `Ausgabe` $w \notin L(G)$

CYK-Algorithmus (Cocke-Younger-Kasami)

Beispiel:

$G = (\{S, A, B\}, \{a, b\}, R, S)$ mit

$$
\begin{aligned}
R: \quad & S \to AB \mid BC \\
& A \to BA \mid a \\
& B \to CC \mid b \\
& C \to AB \mid a
\end{aligned}
$$

An der Tafel.

CYK-Algorithmus (Cocke-Younger-Kasami)

Eigenschaften

Für Wörter der Länge $|w| = n$ entscheidet der CYK-Algorithmus in der Größenordnung von n^3 Schritten, ob $w \in L(G)$ ist.

CYK-Algorithmus (Cocke-Younger-Kasami)

Beispiel.

Grammatik in CNF $G = (\{S, S_a, S_b, A, B\}, \{a, b\}, R, S\}$

$$
\begin{aligned}
R = \{ \quad & S \to AS_a \mid BS_b \mid AA \mid BB \\
& S_a \to SA \\
& S_b \to SB \\
& A \to a \\
& B \to b\}
\end{aligned}
$$

$L(G) = \{vv^R \mid v \in \{a, b\}^+\}$

Beispiel an der Tafel.

Übersicht

1. Motivation
2. Terminologie
3. Endliche Automaten und reguläre Sprachen
4. Kellerautomaten und kontextfreie Sprachen
5. Turingmaschinen und rekursiv aufzählbare Sprachen
6. Berechenbarkeit, (Un-)Entscheidbarkeit
7. Komplexitätsklassen P und NP

Übersicht

1. Motivation
2. Terminologie
3. Endliche Automaten und reguläre Sprachen
4. Kellerautomaten und kontextfreie Sprachen
5. Turingmaschinen und rekursiv aufzählbare Sprachen
6. Berechenbarkeit, (Un-)Entscheidbarkeit
7. Komplexitätsklassen P und NP