

Automaten und Formale Sprachen

Einführung

Ralf Möller

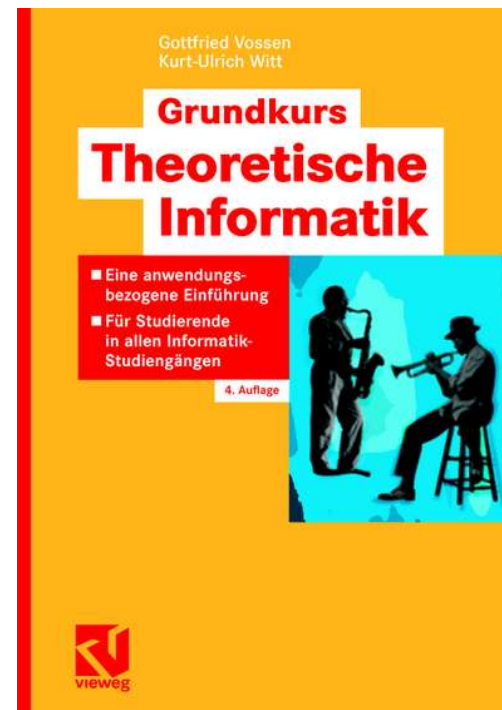
Hamburg Univ. of Technology

Übung Fr. 14:30-15:15

S. Wandelt SBS95,E4.042

Literatur

- **Gottfried Vossen, Kurt-Ulrich Witt:**
Grundkurs Theoretische Informatik,
Vieweg Verlag



Weitere Literatur

U. Schöning: Theoretische Informatik
kurz gefasst, Spektrum Akademischer
Verlag

**John E. Hopcroft, Rajeev Motwani,
Jeffrey D. Ullman:** Introduction to
Automata Theory, Languages, and
Computation, Addison Wesley
Publishing Company

Danksagung

- Kurs basiert auf Präsentationsmaterial von
 - ◆ G. Vossen (Uni Münster),
K.-U. Witt (Hochschule Bonn–Rhein–Sieg)
 - ◆ Christian Sohler (TU Dortmund)
 - ◆ Thomas Ottmann (Uni Freiburg)
 - ◆ Lenore Blum (CMU)

Zentrale Fragestellungen

Effiziente Algorithmen:

- Welche Probleme können **effizient** gelöst werden?
- Wie misst man Effizienz?
- Welche algorithmischen Methoden gibt es, Probleme zu lösen?
- Wie kann man Probleme mit geringstmöglichem Aufwand lösen?
- Wie gehen wir mit schweren Problemen um?

Wozu benötigen wir effiziente Algorithmen?

Beispiele:

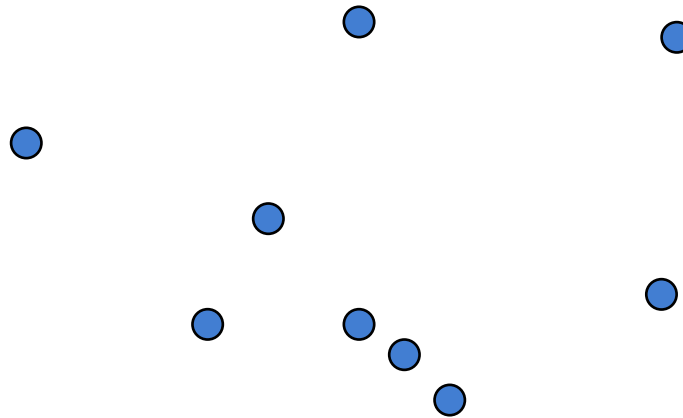
- Internetsuchmaschinen
- Berechnung von Bahnverbindungen
- Optimierung von Unternehmensabläufen
- Datenkompression
- Computer Spiele
- Datenanalyse

Alle diese Bereiche sind (immer noch) Stoff **aktueller Forschung** im Bereich der Algorithmik!

Algorithmische Problemstellungen

Typische Aufgabenstellung:

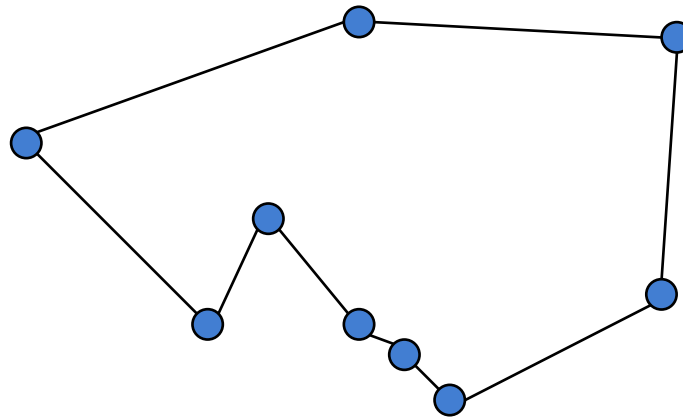
- Berechne die kürzeste Rundreise durch n Städte



Algorithmische Problemstellungen

Typische Aufgabenstellung:

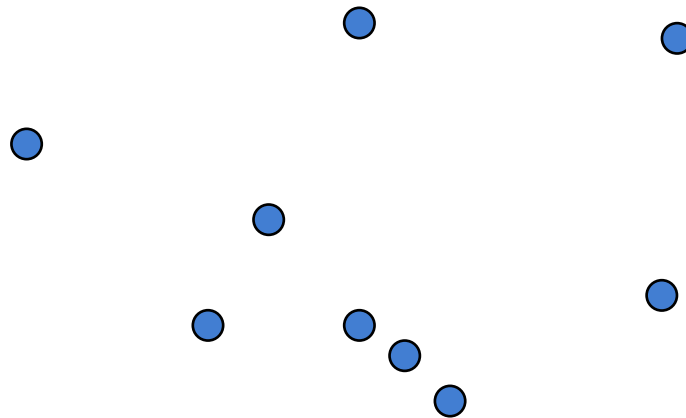
- Berechne die kürzeste Rundreise durch n Städte



Algorithmische Problemstellungen

Optimierungsprobleme (informal):

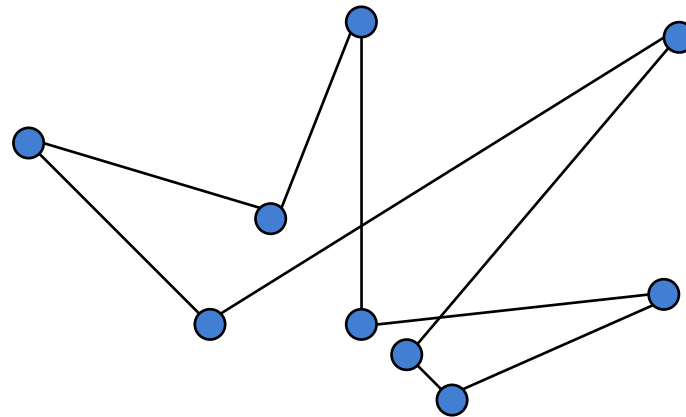
- Zulässigkeitsbedingung (Lösung ist eine Rundreise)
- Zielfunktion (Länge der Tour)
- Aufgabe: Find beste zulässige Lösung



Algorithmische Problemstellungen

Optimierungsprobleme (informal):

- Zulässigkeitsbedingung (Lösung ist eine Rundreise)
- Zielfunktion (Länge der Tour)
- Aufgabe: Find beste zulässige Lösung

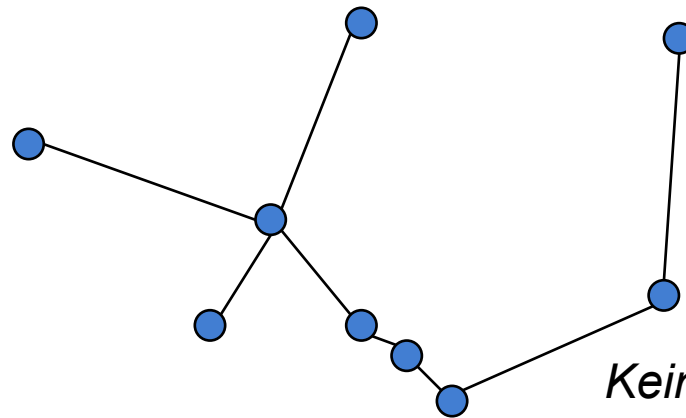


*Lösung ist zulässig, aber
nicht optimal*

Algorithmische Problemstellungen

Optimierungsprobleme (informal):

- Zulässigkeitsbedingung (Lösung ist eine Rundreise)
- Zielfunktion (Länge der Tour)
- Aufgabe: Find beste zulässige Lösung



*Keine zulässige Lösung
Kosten (Summe der Kantenlängen)
Kleiner als bei der besten Rundreise*

Algorithmische Problemstellungen

Typische Aufgabenstellung:

- Entscheide, ob eine Zahl prim ist

Algorithmische Problemstellungen

Typische Aufgabenstellung:

- Entscheide, ob eine Zahl prim ist
- 2 -> prim
- 17 -> prim
- 99 -> nicht prim

Algorithmische Problemstellungen

Typische Aufgabenstellung:

- Entscheide, ob eine Zahl prim ist
[Entscheidungsproblem]
- 2 -> prim
- 17 -> prim
- 99 -> nicht prim

Algorithmische Problemstellungen

Entscheidungsprobleme:

- Eigenschaft (Primzahl)
- Aufgabe:
 - Akzeptieren, wenn Eingabe die Eigenschaft besitzt
 - Zurückweisen, sonst
- Ausgabe: 1 (akzeptieren) oder 0 (zurückweisen)

Algorithmische Problemstellungen

Typische Aufgabenstellung:

- Sortiere Folge von n Zahlen

Algorithmische Problemstellungen

Typische Aufgabenstellung:

- Sortiere Folge von n Zahlen

Eingabe:

- 9, 3, 2, 15, 17, 8

Algorithmische Problemstellungen

Typische Aufgabenstellung:

- Sortiere Folge von n Zahlen

Eingabe:

- 9, 3, 2, 15, 17, 8

Ausgabe:

- 2, 3, 8, 9, 15, 17

Algorithmische Problemstellungen

Typische Aufgabenstellung:

- Sortiere Folge von n Zahlen
[neue Art von Problem?]

Algorithmische Problemstellungen

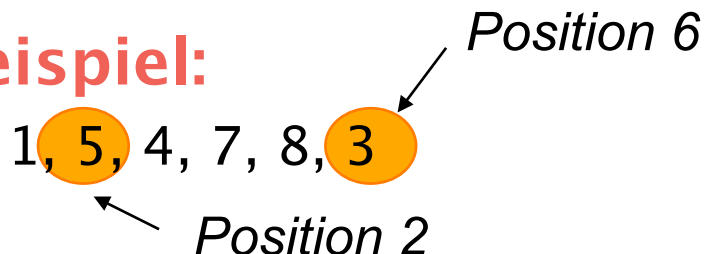
Neue Problemformulierung:

- Finde die Reihenfolge der Zahlen mit der kleinsten Anzahl Inversionen

Inversion:

- Bezeichne x^i die Zahl an Stelle i unserer Reihenfolge
- Das Paar (i,j) ist eine Inversion, wenn gilt $i < j$, aber $x^i > x^j$

Beispiel:

- 1, 5, 4, 7, 8, 3 (2,6) ist eine Inversion
- 

Algorithmische Problemstellungen

Neue Problemformulierung:

- Finde die Reihenfolge der Zahlen mit der kleinsten Anzahl Inversionen

Lemma:

Eine Reihenfolge ohne Inversionen ist aufsteigend sortiert.

Algorithmische Problemstellungen

Neue Problemformulierung:

- Finde die Reihenfolge der Zahlen mit der kleinsten Anzahl Inversionen

Lemma:

Eine Reihenfolge ohne Inversionen ist aufsteigend sortiert.

Lemma:

Eine Reihenfolge mit Inversionen ist nicht aufsteigend sortiert.

Algorithmische Problemstellungen

Erkenntnis:

- Durch geschickte Umformulierung kann man die meisten algorithmischen Probleme als Entscheidungs- oder Optimierungsprobleme formulieren

Vorgehensweise:

- Wir werden zunächst Entscheidungsprobleme untersuchen und uns danach mit Optimierungsproblemen beschäftigen

Erste grundlegende Fragestellung

Welche Entscheidungsprobleme können durch einen Rechner gelöst werden?

Formale Sprachen

Wie kann man Entscheidungsprobleme allgemein formulieren?

- Annahme: Jede Eingabe kann als endliche Zeichenkette (Bitstring) beschrieben werden
- Bei Entscheidungsproblemen müssen wir bestimmen, ob eine Eingabe eine vorgegebene Eigenschaft hat
- Äquivalent: Bestimme die Menge der Bitstrings, die eine Eingabe mit der vorgegebenen Eigenschaft kodieren

Formale Sprachen

Beispiel: Primzahlerkennung

- Eingabe ist eine Zahl
- Kann Zahl durch Binärkodierung darstellen
- Muss alle Zahlen akzeptieren, deren Binärkodierung eine Primzahl ist
- $L = \{\text{Bitstrings } b : b \text{ ist die Binärdarstellung einer Primzahl}\}$
- Entscheide, ob Bitstring b in L ist

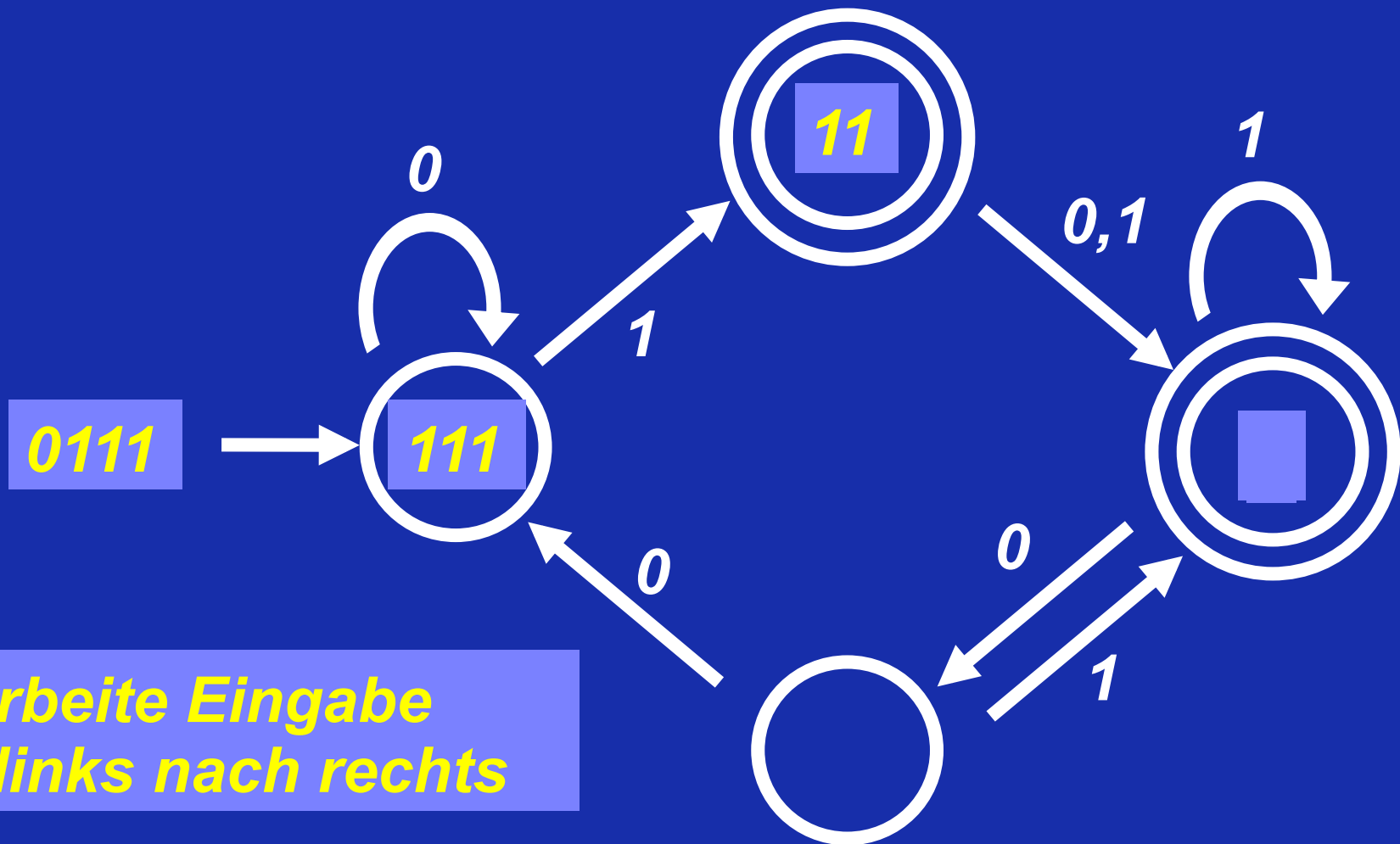
Formale Sprachen

Wichtige Erkenntnis:

- Man kann auf dieselbe Weise jedes Problem als Problem über Bitstrings formulieren

Generalisiertes Problem:

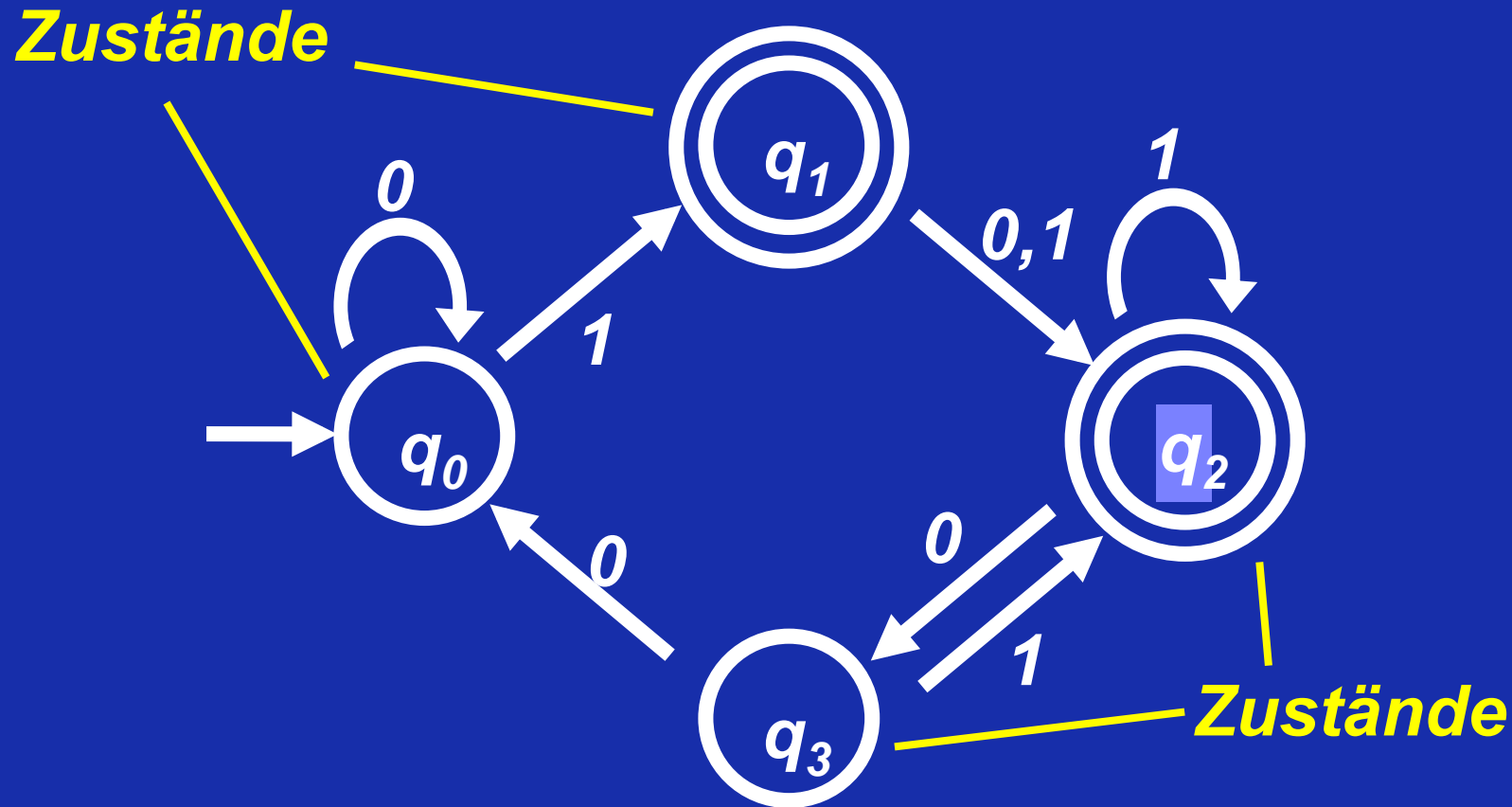
- Sei L eine Menge von Bitstrings
- Entscheide, ob Eingabebitstring b in L liegt



*Verarbeitete Eingabe
von links nach rechts*

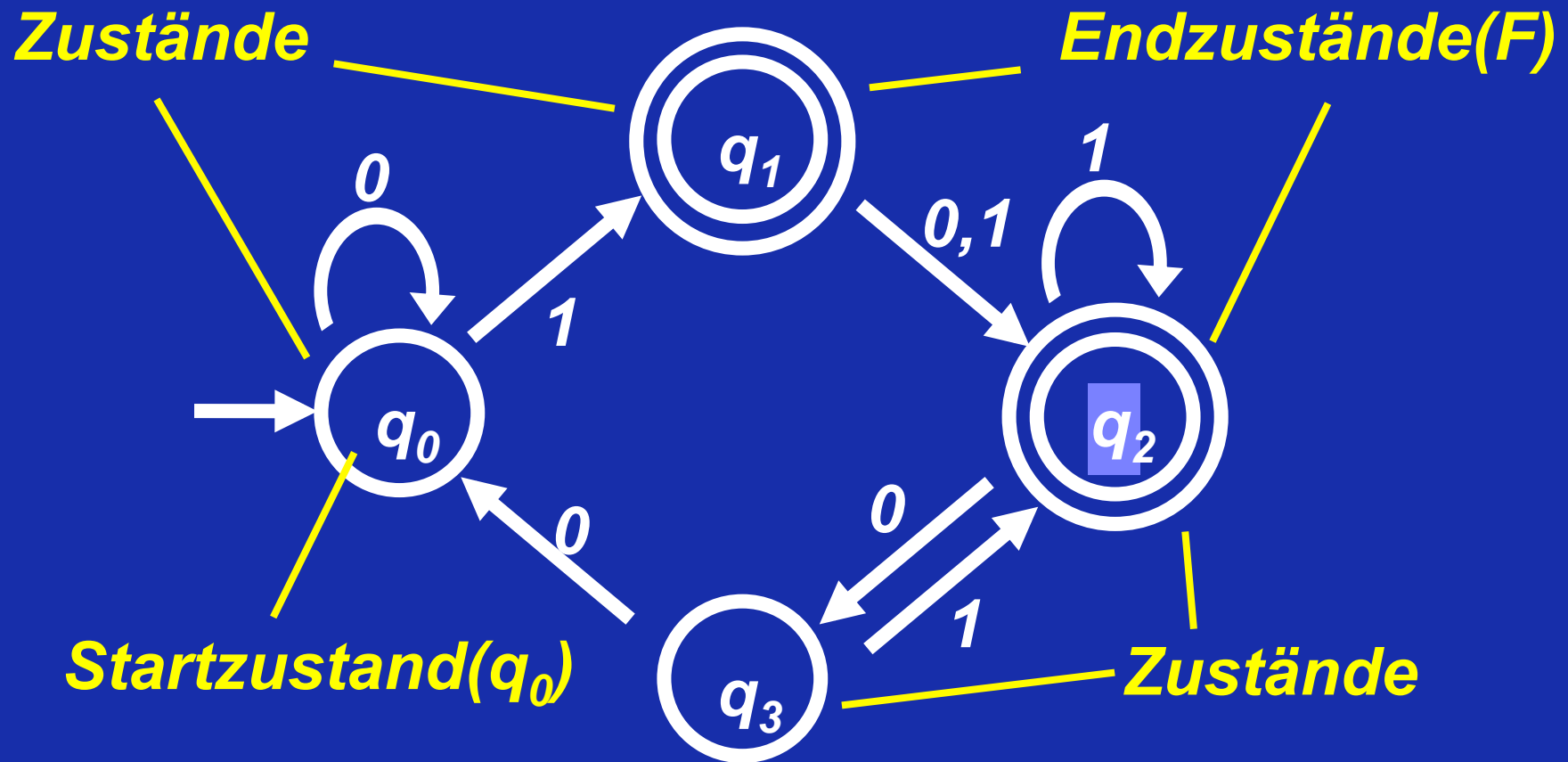
*Die Maschine **akzeptiert** eine
Eingabezeichenkette wenn der Prozess in
einem Zustand mit Doppelkreis endet*

Ein Deterministischer Endlicher Automat (DEA)



Die Maschine **akzeptiert** eine Eingabezeichenkette wenn der Prozess in einem Zustand mit Doppelkreis endet

Ein Deterministischer Endlicher Automat (DEA)

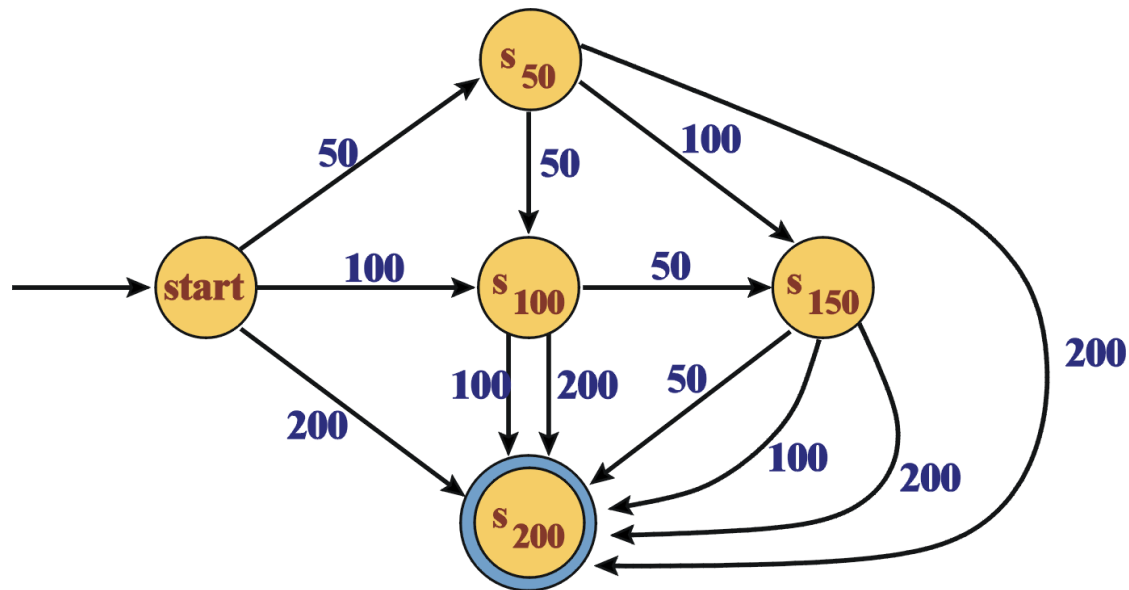


Die Maschine **akzeptiert** eine Eingabezeichenkette wenn der Prozess in einem Zustand mit Doppelkreis endet

Zustandsdiagramm des Automaten A_{swim}

Eigenschaften von A_{swim} :

- Münzeingaben mit Werten 50, 100, 200 in beliebiger Reihenfolge
- Nach Einwurf von insgesamt ≥ 200 akzeptiert A_{swim} : Eintritt freigegeben!
- Der Gesamtwert der bisherigen Eingabe ist im aktuellen Zustand vermerkt.



Startkonfiguration von A_{swim}

Eingabeband enthält Eingaben als Folgen von Zeichen
Zustandsspeicher enthält jeweils aktuellen Zustand
Programm, Kontrolle: Zustandsübergangsfunktion δ .

Eingabeband

50	100	50		...
----	-----	----	--	-----



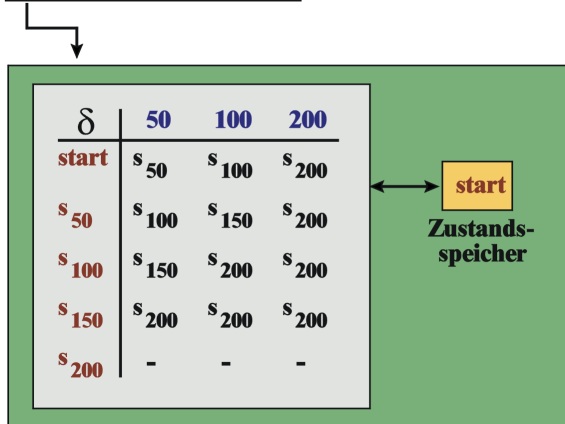
δ	50	100	200
start	s ₅₀	s ₁₀₀	s ₂₀₀
s ₅₀	s ₁₀₀	s ₁₅₀	s ₂₀₀
s ₁₀₀	s ₁₅₀	s ₂₀₀	s ₂₀₀
s ₁₅₀	s ₂₀₀	s ₂₀₀	s ₂₀₀
s ₂₀₀	-	-	-

start
Zustands-
speicher

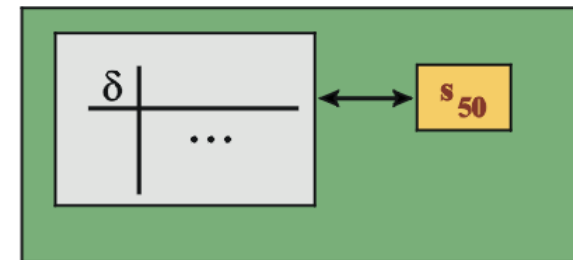
Rechnung des Automaten A_{swim}

Eingabeband

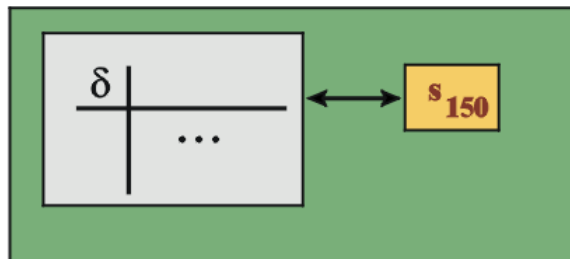
50	100	50		...
----	-----	----	--	-----



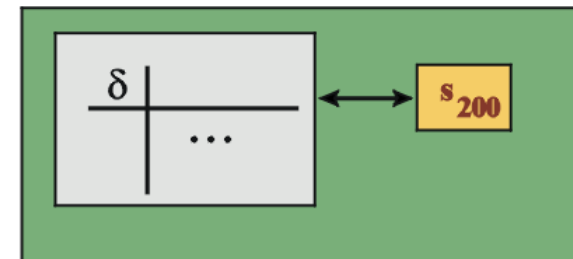
50	100	50		...
----	-----	----	--	-----



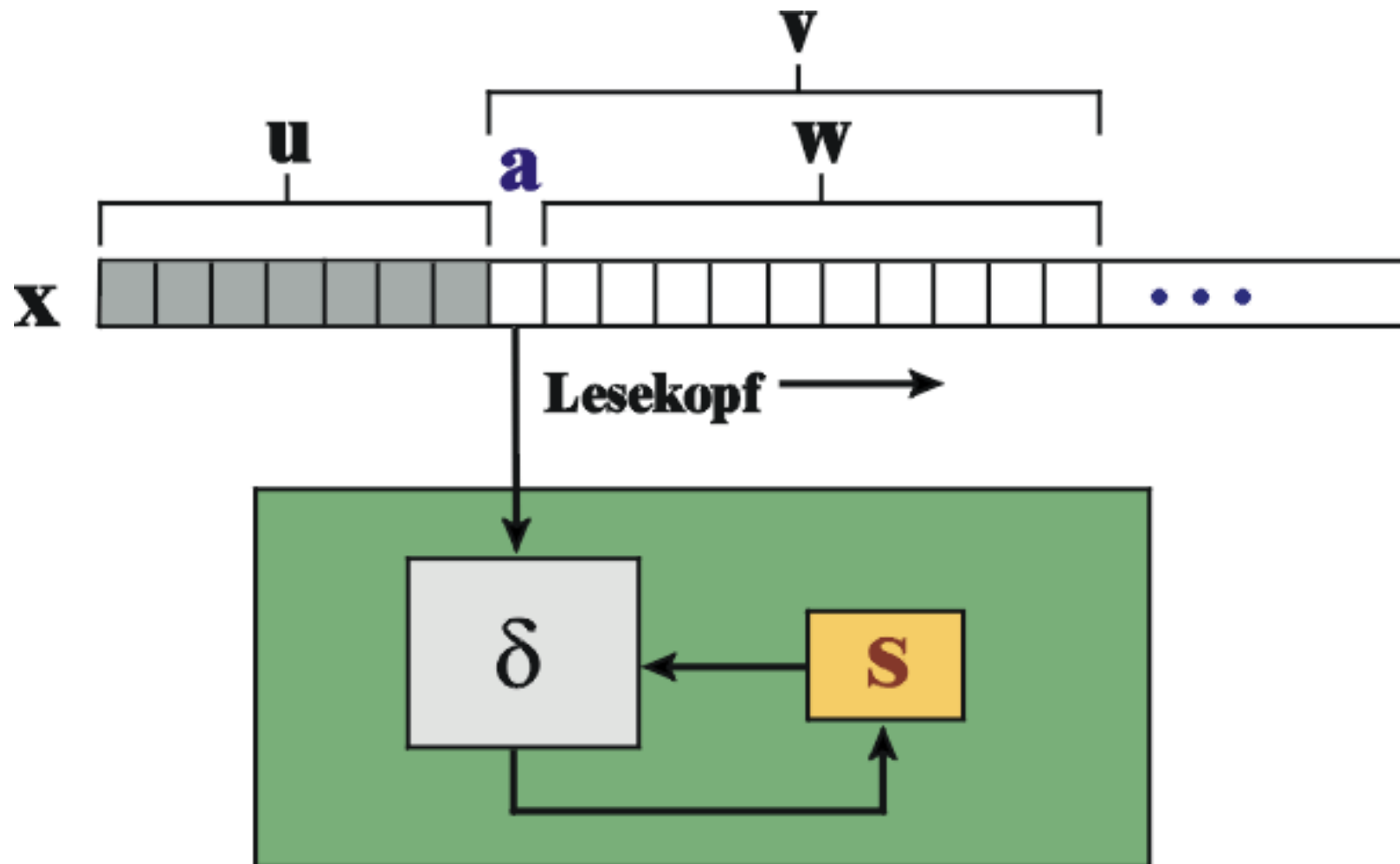
50	100	50		...
----	-----	----	--	-----



50	100	50		...
----	-----	----	--	-----



Konfiguration eines endlichen Automaten



Alphabete

- Automaten verarbeiten **Zeichenfolgen**, die aus atomaren **Symbolen** bestehen.
- Menge der zugelassenen Zeichen: **Endliches Alphabet Σ** .

Beispiele:

- ♦ $\Sigma = \{\underline{50}, \underline{100}, \underline{200}\}$ $|\Sigma| = 3$
- ♦ $\Sigma = \{a_1, a_2, a_3, \dots, a_n\}$ $|\Sigma| = n$
- ♦ $\Sigma = \{a, b, \dots, z\}$ $|\Sigma| = 26$
- ♦ $\Sigma = \emptyset$ $|\Sigma| = 0$

Deterministische endliche Automaten

Ein **deterministischer endlicher Automat (DFA)** ist gegeben durch

- eine endliche Menge S von **Zuständen**
- eine endliche Menge Σ von **Eingabezeichen**
- einen **Anfangszustand** $s_0 \in S$
- eine **Endzustandsmenge** $F \subseteq S$
- eine **Übergangsfunktion** $\delta : S \times \Sigma \rightarrow S$

Kurz: $A = (\Sigma, S, \delta, s_0, F)$

δ kann auch durch einen **Zustandsübergangs Graphen** oder als Menge von Tripeln (s, a, t) mit $\delta(s, a) = t$ gegeben sein

δ ist manchmal nicht total (überall definiert)

Erweiterte Übergangsfunktion

Die Zustandsübergangsfunktion δ kann von Zeichen auf Wörter erweitert werden:

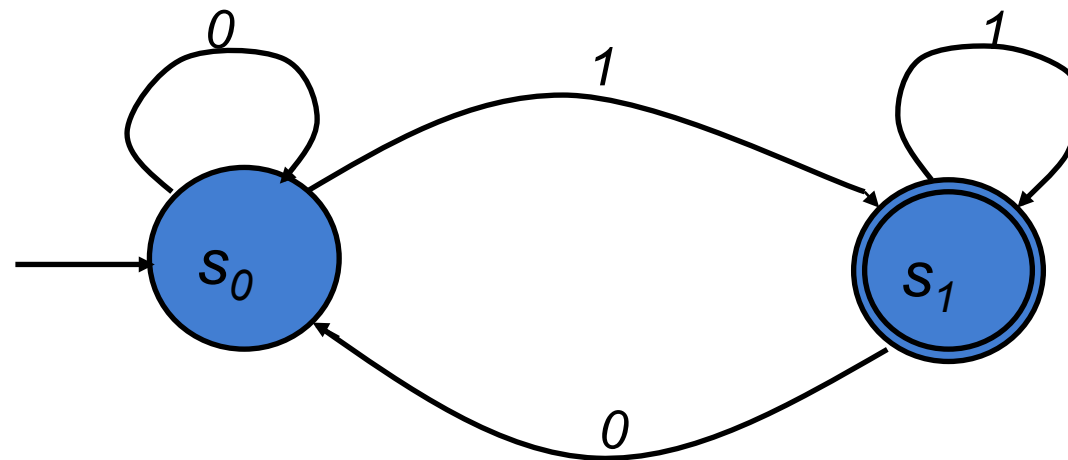
$\delta^* : S \times \Sigma^* \rightarrow S$ definiert durch

- ♦ $\delta^*(s, \varepsilon) = s$ für alle $s \in S$
- ♦ $\delta^*(s, aw) = \delta^*(\delta(s, a), w)$ für alle $a \in \Sigma, w \in \Sigma^*$

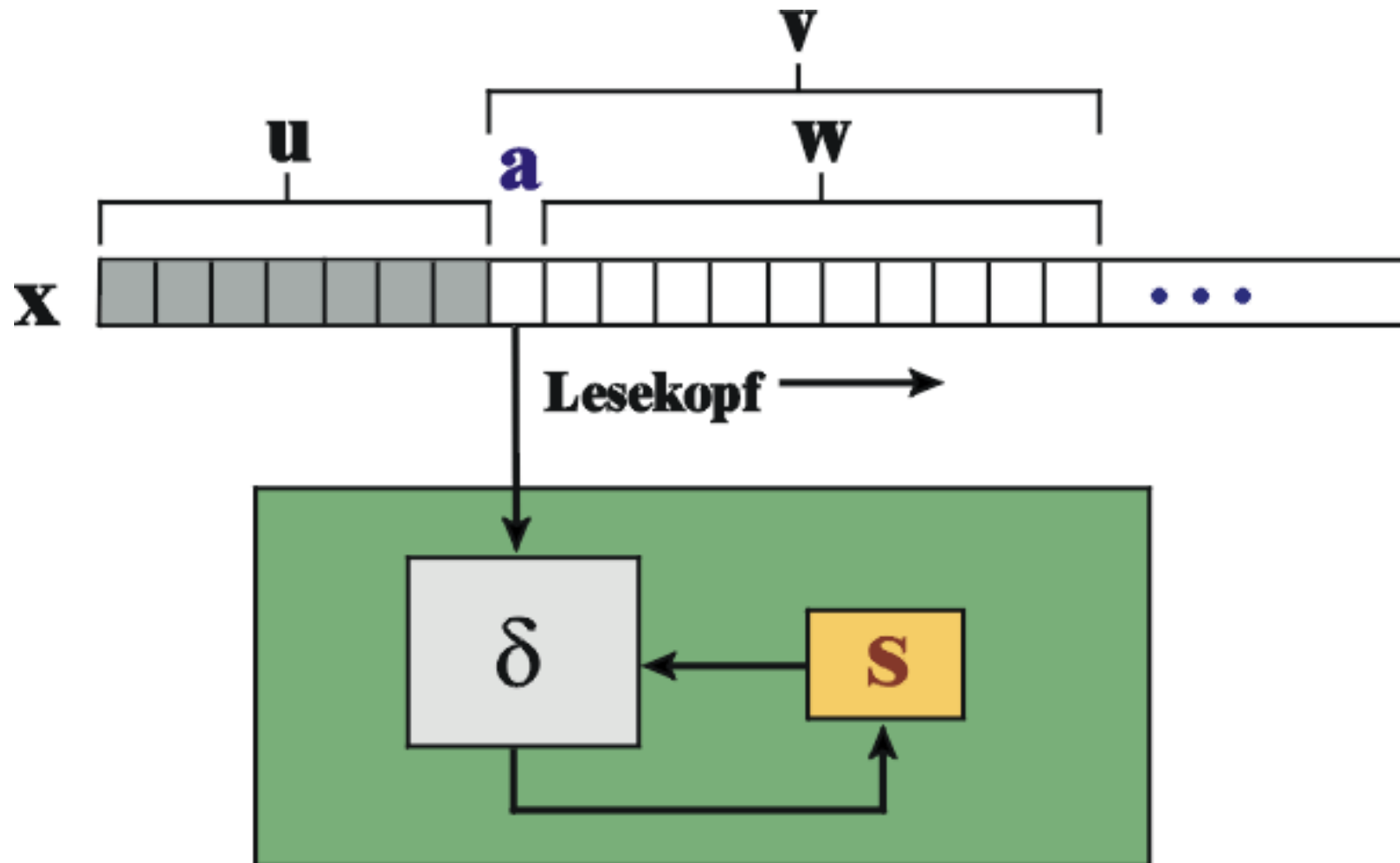
Für einen endlichen Automaten $A = (\Sigma, S, \delta, s_0, F)$ wird **die von A akzeptierte Sprache** (die Menge aller von A akzeptierten Eingabefolgen) $L(A) \subseteq \Sigma^*$ definiert durch:

$$L(A) = \{w; \delta^*(s_0, w) \in F\}$$

Beispiel



Konfiguration eines endlichen Automaten



Konfigurationsübergänge

Ein **Konfigurationsübergang** $(s, v) \vdash (t, w)$ kann stattfinden, wenn $v = aw$ und $\delta(s, a) = t$ ist.

Die **Abarbeitung** eines Wortes $x = x_1x_2 \dots x_r$ durch einen DFA kann als Folge von Konfigurationsübergängen beschrieben werden:

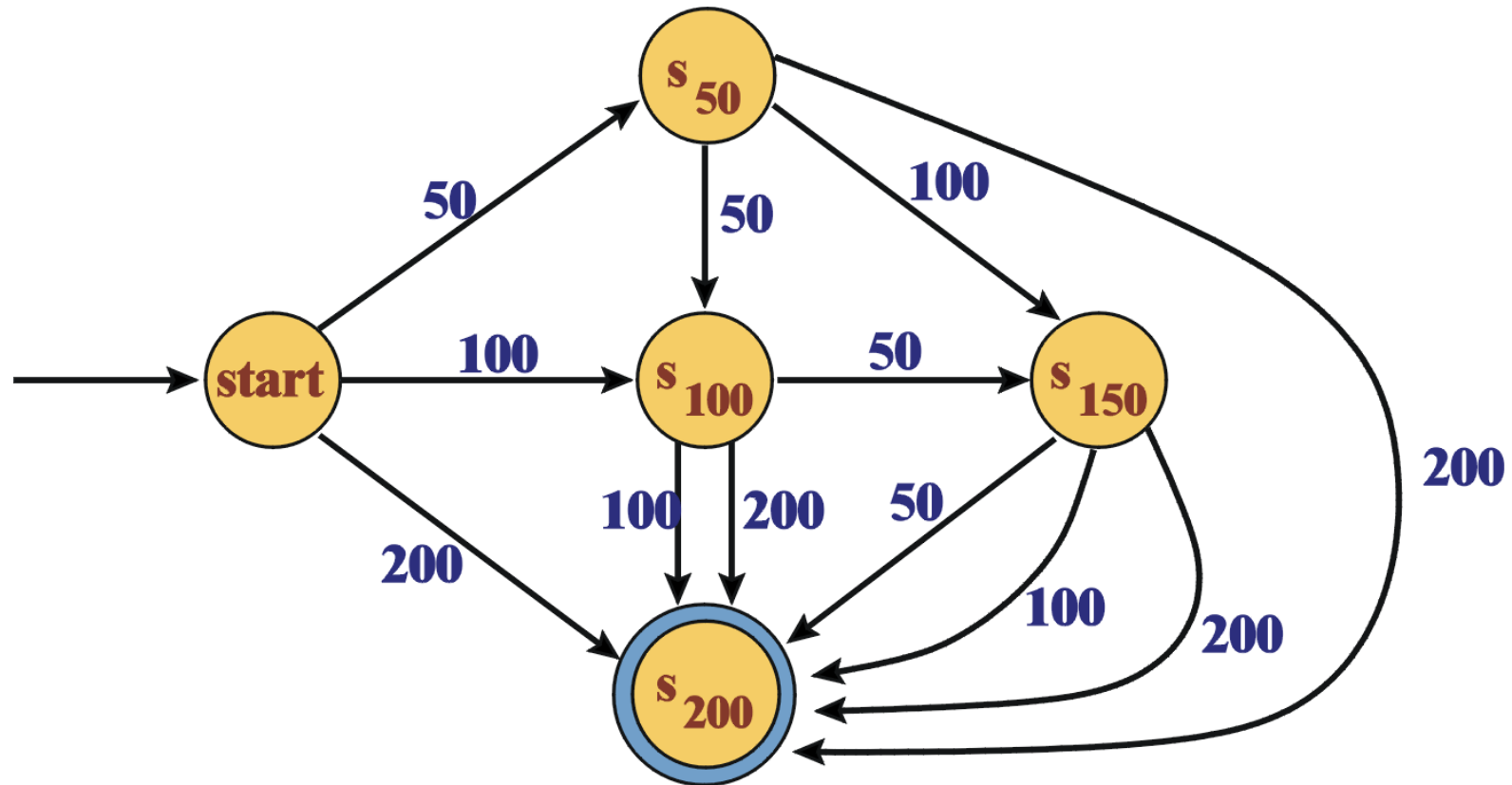
$$(s_0, x_1x_2 \dots x_r) \vdash (s_1, x_2 \dots x_r) \vdash \dots \vdash (s_r, \varepsilon)$$

Mit \vdash^* wird die transitiv-reflexive Hülle von \vdash beschrieben.

Beispiel:

$$(start, \underline{50} \underline{100} \underline{50}) \vdash$$

$(start, \underline{50} \ \underline{100} \ 50) \vdash$



Reguläre Sprachen

- Für einen DFA $A = (\Sigma, S, \delta, s_0, F)$ ist
$$L(A) = \{w \in \Sigma^* ; (s_0, w) \vdash^* (s, \varepsilon), s \in F\}$$
die von A **akzeptierte Sprache**.
- Eine Sprache $L \subseteq \Sigma^*$ heißt **regulär**, wenn es einen DFA A gibt mit $L = L(A)$.
- Zwei DFA A und A' heißen **äquivalent**, falls sie die gleiche Sprache akzeptieren, wenn also gilt: $L(A) = L(A')$.