

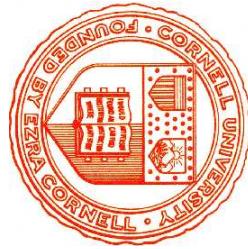
# Theoretische Informatik I

Wintersemester 2011/12

**Christoph Kreitz / Maria Knobelsdorff**

Theoretische Informatik

{kreitz,knobelsdorf}@cs.uni-potsdam.de  
<http://cs.uni-potsdam.de/til-ws1112>



1. Lehrziele und Lernformen
2. Organisatorisches
3. Gedanken zur Arbeitsethik

# THEORETISCHE INFORMATIK: WORUM GEHT ES?

## Die Wissenschaft von der Berechnung

- **Analyse von Grundsatzfragen**

- Was kann man mit Computern lösen, was nicht?
- Für welche Probleme gibt es gute, allgemeingültige Verfahren?
- Welche Probleme sind effizient lösbar – welche nicht?
- Wie flexibel kann man Programmiersprachen gestalten?
- Wie hängen verschiedenartige Computerarchitekturen zusammen?

- **Mathematische Vorgehensweise**

- Abstraktion von irrelevanten Details (Hardware/Programmiersprache)
- Erkenntnisse sind beweisbar und haben weitreichende Gültigkeit

- **Der älteste Zweig der Informatik**

- Theoretische Erkenntnisse gab es lange vor den ersten Computern
- Aussagen sind langlebiger als in den anderen Informatikzweigen
- Aber: Erkenntnisse sind selten “unmittelbar” anwendbar

# WOZU SOLL DAS GUT SEIN?

## Denken ist besser als Hacken

- Verständnis allgemeiner Zusammenhänge
  - Details ändern sich schnell, Modelle und Methoden nicht
- Effizientere Arbeitsweise
  - Abstraktion richtet den Blick auf das Wesentliche
    - Wenn Sie zu sehr auf Details schauen, verlieren Sie die Übersicht
    - Abstrakte Lösungen sind auf spezifische Situationen übertragbar
- Vermeidung der gröbsten Fehler
  - z.B. Korrektheit von Software kann nicht getestet werden
    - Optimale Navigation ist nicht effizient möglich
    - Flexible Programmiersprachen sind nicht (effizient) compilierbar
  - Viele Programmierer haben sich schon in unlösbare Probleme verbissen, weil sie das nicht wußten oder nicht geglaubt haben

# KONKRETE THEMEN DER THEORETISCHEN INFORMATIK

- **Mathematische Methodik in der Informatik**

[TI-1]

- **Automatentheorie und Formale Sprachen**

[TI-1]

- Endliche Automaten und Reguläre Sprachen – Lexikalische Analyse
- Kontextfreie Sprachen und Pushdown Automaten – Syntaxanalyse
- Turingmaschinen, kontextsensitive und allgemeine formale Sprachen

- Theorie der Berechenbarkeit

[TI-2]

- Berechenbarkeitsmodelle
- Aufzählbarkeit, Entscheidbarkeit, Unlösbarer Probleme

- **Komplexitätstheorie**

[TI-2]

- Komplexitätsmaße und -klassen für Algorithmen und Probleme
- Nicht handhabbare Probleme ( $\mathcal{NP}$ -Vollständigkeit)
- Effiziente Alternativen zu konventionellen Verfahren

# DER LEHRSTOFF

- **Reihenfolge und Notation folgt Leittext**

– **J. Hopcroft, R. Motwani, J. Ullman:** *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie, Pearson 2002*

- Vorlesungsfolien sind im Voraus auf dem Webserver erhältlich
- Videomitschnitte der Vorlesungen von 2005 und 2006 online verfügbar

- **Lesenswerte Zusatzliteratur**

- G. Vossen, K.-U. Witt: *Grundkurs Theoretische Informatik*. Vieweg 2004
- M. Sipser: *Introduction to the Theory of Computation*. PWS 2005
- A. Asteroth, C. Baier: *Theoretische Informatik*, Pearson 2002
- J. Hromkovic: *Sieben Wunder der Informatik*, Teubner Verlag 2006
- I. Wegener: *Theoretische Informatik*, Teubner Verlag 1993
- U. Schöning: *Theoretische Informatik - kurzgefaßt*, Spektrum-Verlag 2008
- K. Erk, L. Priese: *Theoretische Informatik*, Springer Verlag 2000
- H. Lewis, C. Papadimitriou: *Elements of the Theory of Computation*, PH 1998
- P. Leypold: *Schneller Studieren*. Pearson 2005

# LEHR- UND LERNFORMEN

## • **Selbststudium ist das wichtigste**

- Lernen durch Bearbeitung verschiedener Quellen (Literatur, Web,...)
- Trainieren durch Lösung von leichten und schweren Beispielaufgaben alleine und im Team mit anderen
- Nachweis von Fähigkeiten in Prüfungen und Projekten
- Ziel ist Verständnis eines Themengebiets (nicht nur der Vorlesung)
- Unsere Aufgabe ist, Ihnen dabei zu helfen

## • **Vorlesung**

- Vorstellung und Illustration zentraler Konzepte und Zusammenhänge
- Knapp und “unvollständig” – nur als Heranführung gedacht
- Die Idee (Verstehen) zählt mehr als das Detail (Aufschreiben)
- Es hilft, schon etwas über das Thema **im Voraus** zu lesen
- **Stellen Sie Fragen**, wenn Ihnen etwas unklar ist !!
- Nutzen Sie das optionale Tutorium      wöchentlich Do 14:15–15:45

**Was soll ich lernen ?**

# LEHR- UND LERNFORMEN (II)

## • Übungen

### **Vertiefung und Anwendung**

- Kurzquiz als Selbsttest – verstehe ich bisher besprochene Konzepte?
- Betreutes Üben in Gruppen: Lösung von Problemen unter Anleitung
- Klärung von Fragen allgemeinen Interesses
- Eigenständige Einarbeitung in neue Konzepte und Fragestellungen
- Bearbeitung von aufwändigeren Hausaufgaben: Feedback & Korrektur
  - Ziel ist verständliches Aufschreiben einer vollständigen Lösung
  - Arbeit in Gruppen sehr zu empfehlen
- Lösungen schwieriger Aufgaben werden im **Tutorium** besprochen
- **Selbst aktiv werden** ist notwendig für erfolgreiches Lernen
- Kommen Sie vorbereitet – Sie lernen mehr dabei

## • Sprechstunden

### **Persönliche Beratung**

- Fachberatung zur Optimierung des individuellen Lernstils
- Klärung von Schwierigkeiten mit der Thematik
- ... aber nicht Lösung der Hausaufgaben

## ORGANISATORISCHES

- **Zielgruppe: ab 1. Semester**
  - Bei geringen mathematischen Vorkenntnissen besser ab 3. Semester
  - Oft ist es sinnvoll erst an Mathematikveranstaltungen teilzunehmen
- **Vorlesung**
  - Wöchentlich Fr 8:30–10:00
- **Tutorium** (optional)
  - Besprechung von allgemeinen Fragen und schwierigen Hausaufgaben
  - Wöchentlich Do 14:15–15:45, ab 3. November
- **Übungen**
  - 8 Gruppen, wöchentlich (Montags – Mittwochs) je 2 Stunden
- **Sprechstunden**
  - C. Kreitz: Fr 10:30–11:30 . . . , und immer wenn die Türe offen ist
  - M. Knobelsdorf: Fr 10:30–12:00
  - Tutoren: individuell in Übungsgruppen vereinbaren

# LEISTUNGSERFASSUNG

- Eine Klausur entscheidet die Note

Anmeldung!

- Hauptklausur 17. Februar 2012, 9:30–12:30 Uhr
- Probeklausur 23. Dezember 2011, 8:20–9:50 Uhr

- Zulassung zur Klausur

- 50% der Punkte in den Hausaufgaben

- Gruppen bis 4 Studenten dürfen gemeinsame Lösungen abgeben
- Gruppen dürfen sich nur nach Rücksprache ändern
- Klausurzulassungen aus Vorjahren sind nicht gültig
- Probeklausur zählt wie ein Hausaufgabenblatt

- Vorbereitung auf die Klausur

- Kurzquiz in jeder Übungsstunde ernsthaft bearbeiten
- Eigenständige Lösung von Haus- und Übungsaufgaben
- Feedback durch Korrektur der Hausaufgaben und der Probeklausur
- Klärung von Fragen in Übung und Sprechstunden

Fangen Sie frühzeitig mit den Vorbereitungen an

# WELCHE VORKENNTNISSE SOLLTEN SIE MITBRINGEN?

## Eine gute Oberstufenmathematik reicht aus

- **Verständnis mathematischer Konzepte**

- Elementare Mengentheorie und die Gesetze von  $\{x | P(x)\}$ ,  $\cup$ ,  $\cap$
- Bezug zwischen Mengen, Relationen und Funktionen
- Datenstrukturen wie Listen, Wörter, Graphen, Bäume ...
- Elementare Gesetze der Algebra und Logik
- Elementare Wahrscheinlichkeitsrechnung

– Zusammenhang zwischen formaler und informaler Beschreibung  
Nötiges Vokabular wird bei Bedarf kurz vorgestellt/wiederholt/eingeübt

- **Verständnis mathematischer Beweismethoden**

Informatiker müssen Korrektheit von Programmen beweisen können

- Deduktive Beweise für Analyse von Befehlssequenzen
- Induktionsbeweise für Analyse von Rekursion / Schleifen
- Widerlegungsbeweise / Gegenbeispiele für Unmöglichkeitsaussagen

[Mehr dazu nach der Pause](#)

# NUTZEN SIE IHRE CHANCEN!

- **Theorie ist bedeutender als viele glauben**
  - Ist Theorie langweilig? überflüssig? unverständlich? ... eine Plage?
  - Alle großen Softwareprojekte benutzten theoretische Modelle
  - Ohne theoretische Kenntnisse begehen Sie viele elementare Fehler
  - Theorie kann durchaus sehr interessant sein
- **Es geht um mehr als nur bestehen**
  - Das wichtige ist **Verstehen**
  - Sie können jetzt umsonst lernen, was später teure Lehrgänge benötigt
  - Wann kommen Sie je wieder mit den Besten des Gebietes in Kontakt?
- **Die Türe steht offen**
  - Lernfrust und mangelnder Durchblick sind normal aber heilbar
  - Kommen Sie in die Sprechstunden und stellen Sie Fragen

# VERTRAUEN IST EIN KOSTBARES GUT

## ... missbrauchen Sie es nicht

- **Abschreiben fremder Lösungen bringt nichts**

- Sie lernen nichts dabei – weder Inhalt noch Durchhaltevermögen
- Sie erkennen Ihre Lücken **nicht** und nehmen Hilfe zu spät wahr
- Sie werden nie ein echtes Erfolgserlebnis haben
- Es schadet Ihrer persönlichen Entwicklung

- **Wir vertrauen Ihrer Ehrlichkeit**

- Benutzen Sie externe Ideen (Bücher/Internet) nur mit Quellenangabe
- Benutzen Sie keine Lösungen von Kommilitonen
- Geben Sie keine Lösungen an Kommilitonen weiter

**Klausurlösungen sollten ausschließlich Ihre eigenen sein**

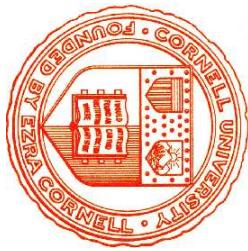
*Keine "Überwachung", aber wenn es dennoch auffliegt ...*

- **Mehr zur Arbeitsethik auf unseren Webseiten**

# Theoretische Informatik I

## Einheit 1

### Mathematische Beweisführung



1. Deduktive Beweise
2. Widerlegungsbeweise
3. Induktion
4. Methodik des Problemlösens

Anhang: Wichtige Grundbegriffe

# Beweisführung in der Informatik – Warum?

- **Testen von Programmen ist unzureichend**

- Nur hilfreich zur Entdeckung grober Fehler
- Viele kleine, aber gravierende Fehler fallen durch das Testraster
  - Pentium Bug (1994), Ariane 5 (1996), Mars Polar Lander (1999), ...

- **Kritische Programme muss man “beweisen”**

- Erfolgreicher Beweis zeigt genau, wie das Programm arbeitet
- Erfolgloser Beweisversuch deutet auf mögliche Fehler im Programm
- Jeder Informatiker sollte die eigenen Programme beweisen

- **Jeder Informatiker muss Beweise verstehen / führen**

- Deduktive Beweise für sequentielle Verarbeitung
- Induktionsbeweise für Rekursion / Schleifen
- Widerlegungsbeweise / Gegenbeispiele für Unmöglichkeitsaussagen

## Wie führt man stichhaltige Beweise?

# METHODIK DER BEWEISFÜHRUNG

- **Ziel:** Zeige, daß eine Behauptung  $B$  aus Annahmen  $A$  folgt

- **Einfachste Methode: Deduktiver Beweis**

- Aneinanderkettung von Argumenten / Aussagen  $A_1, A_2, \dots, A_n = B$
- Zwischenaussagen  $A_i$  müssen schlüssig aus dem Vorhergehenden folgen
- Verwendet werden dürfen nur Annahmen aus  $A$ , Definitionen, bewiesene Aussagen, mathematische Grundgesetze und logische Schlußfolgerungen

- **Beispiel: "Die Summe zweier ungerader Zahlen ist gerade"**

- Informaler Beweis: Es seien  $a$  und  $b$  zwei ungerade Zahlen.  
Per Definition ist  $a = 2x + 1$  und  $b = 2y + 1$  für gewisse  $x$  und  $y$  und die Summe ist  $2(x + y + 1)$ , also eine gerade Zahl.
- **Schematische Darstellung** ist meist kürzer und präziser

Aussage	Begründung
1. $a = 2x + 1$	Gegeben (Auflösung des Begriffs "ungerade")
2. $b = 2y + 1$	Gegeben (Auflösung des Begriffs "ungerade")
3. $a + b = 2(x + y + 1)$	(1,2) und Gesetze der Arithmetik

# AUSFÜHRLICHER SCHEMATISCHER BEWEIS

Wenn  $S$  endliche Teilmenge einer Menge  $U$  ist und das Komplement von  $S$  bezüglich  $U$  endlich ist, dann ist  $U$  endlich

## • Definitionen

$S$  endlich  $\equiv$  Es gibt eine ganze Zahl  $n$  mit  $|S| = n$   
 $T$  Komplement von  $S$   $\equiv T \cup S = U$  und  $T \cap S = \emptyset$

## • Beweis

Aussage	Begründung
1. $S$ endlich	Gegeben
2. $T$ Komplement von $S$	Gegeben
3. $T$ endlich	Gegeben
4. $ S  = n$ für ein $n \in \mathbb{N}$	Auflösen der Definition in (1)
5. $ T  = m$ für ein $m \in \mathbb{N}$	Auflösen der Definition in (3)
6. $T \cup S = U$	Auflösen der Definition in (2)
7. $T \cap S = \emptyset$	Auflösen der Definition in (2)
8. $ U  = m + n$ für $n, m \in \mathbb{N}$	(4),(5),(6), (7) und Gesetze der Kardinalität
9. $U$ endlich	Einsetzen der Definition in (8)

# BEWEISFÜHRUNG DURCH ‘UMKEHRUNG’

## • Kontraposition

- Anstatt zu zeigen, daß die Behauptung  $B$  aus den Annahmen  $A$  folgt, beweise, daß **nicht**  $A$  aus der Annahme **nicht**  $B$  folgt
- Aussagenlogisch ist  $\neg B \Rightarrow \neg A$  äquivalent zu  $A \Rightarrow B$

## • Häufigste Anwendung: Indirekte Beweisführung

- Zeige, daß aus **nicht**  $B$  und  $A$  ein Widerspruch (also  $\neg A$ ) folgt
- Aussagenlogisch ist  $\neg(\neg B \wedge A)$  äquivalent zu  $A \Rightarrow B$
- Beispiel: *Wenn für eine natürliche Zahl  $x$  gilt  $x^2 > 1$ , dann ist  $x \geq 2$*   
Beweis: *Sei  $x^2 > 1$ . Wenn  $x \geq 2$  nicht gilt, dann ist  $x = 1$  oder  $x = 0$ .*  
*Wegen  $1^2 = 1$  und  $0^2 = 0$  ist  $x^2 > 1$  in beiden Fällen falsch.*  
*Also muss  $x \geq 2$  sein*

## WIDERLEGUNGSBEWEISE

### ZEIGE, DASS EINE BEHAUPTUNG $B$ NICHT GILT

- **Widerspruchsbeweise zeigen, daß  $B$  niemals gelten kann**

- Zeige, daß aus Annahme  $B$  ein Widerspruch folgt

**Beispiel:** Ist  $S$  endliche Teilmenge einer unendlichen Menge  $U$ , dann ist das Komplement von  $S$  bezüglich  $U$  nicht endlich

Beweis	Aussage	Begründung
	1. $S$ endlich	Gegeben
	2. $T$ Komplement von $S$	Gegeben
	3. $U$ unendlich	Gegeben
	4. $T$ endlich	Annahme
	5. $U$ endlich	(1), (4) mit Satz auf Folie 3
	6. Widerspruch	(3), (5)
	7. $T$ nicht endlich	Annahme (4) muss falsch sein

- **Gegenbeispiele zeigen, daß  $B$  nicht immer wahr sein kann**

- $B$  ist nicht **allgemeingültig**, wenn es ein einziges Gegenbeispiel gibt
- Beispiel: *Wenn  $x$  eine Primzahl ist, dann ist  $x$  ungerade* ist falsch

Gegenbeispiel: 2 ist eine gerade Zahl, die eine Primzahl ist

# INDUKTIVE BEWEISE

## Beweise eine Behauptung $B$ für alle natürlichen Zahlen

### • Standardinduktion

- Gilt  $B$  für  $i$  und  $B$  für  $n+1$ , wenn  $B$  für  $n$  gilt, dann gilt  $B$  für alle  $n \geq i$
- Beispiel: Wenn  $x \geq 4$ , dann  $2^x \geq x^2$

Induktionsanfang  $x=4$ : Es ist  $2^x = 16 \geq 16 = x^2$

Induktionsschritt: Es gelte  $2^n \geq n^2$  für ein beliebiges  $n \geq 4$

Dann ist  $2^{n+1} = 2 * 2^n \geq 2n^2$  aufgrund der Induktionsannahme

und  $(n+1)^2 = n^2 + 2n + 1 = n(n+2+\frac{1}{n}) \leq n(n+n) = 2n^2$  wegen  $n \geq 4$

also gilt  $2^{n+1} \geq (n+1)^2$

### • Vollständige Induktion

- Folgt  $B$  für  $n$ , wenn  $B$  für alle  $i \leq j < n$  gilt, dann gilt  $B$  für alle  $n \geq i$
- Mächtiger, da man nicht den unmittelbaren Vorgänger benutzen muss

# STRUKTURELLE INDUKTION

## Zeige $B$ für alle Elemente eines rekursiven Datentyps

- Gilt  $B$  für das Basiselement und für ein zusammengesetztes Element, wenn  $B$  für seine Unterelemente gilt, dann gilt  $B$  für alle Elemente
- Beispiel: *Die Summe einer Liste  $L$  von positiven ganzen Zahlen ist mindestens so groß wie ihre Länge*

Induktionsanfang  $L$  ist leer: Die Summe und die Länge von  $L$  sind 0

Induktionsschritt: Es gelte  $\text{sum}(L) \geq |L|$

Betrachte die Liste  $Lox$ , die durch Anhängen von  $x$  an  $L$  entsteht

Dann gilt  $\text{sum}(Lox) = \text{sum}(L) + x \geq \text{sum}(L) + 1 \geq |L| + 1 = |Lox|$

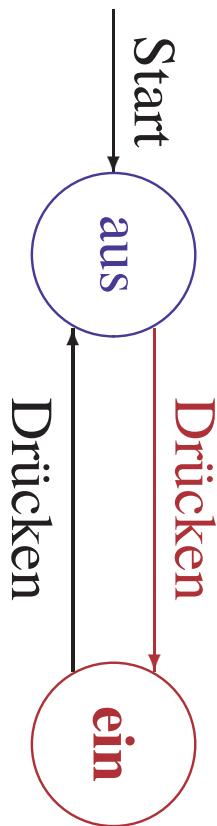
Häufig eingesetzt für Analyse von

- Listen- und Baumstrukturen (Suchen, Sortieren, ...)
- Syntaktische Strukturen (Formeln, Programmiersprachen, ...)

⋮

# SIMULTANE (GEGENSEITIGE) INDUKTION

## Zeige mehrere zusammengehörige Aussagen simultan



### Zeige: Automat ist ein Wechselschalter

$S_1(n)$ : Für gerade  $n$  ist der Automat nach  $n$ -fachem Drücken ausgeschaltet

$S_2(n)$ : Für ungerade  $n$  ist der Automat nach  $n$ -fachem Drücken eingeschaltet

Induktionsanfang  $n=0$ :  $n$  ist gerade, also nicht ungerade; somit gilt  $S_2(0)$   
der Automat ist ausgeschaltet, also gilt  $S_1(0)$

Induktionsschritt: Es gelte  $S_1(n)$  und  $S_2(n)$ . Betrachte  $n+1$

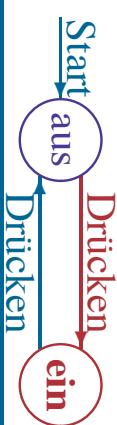
– Falls  $n+1$  ungerade, dann gilt  $S_1(n+1)$  und  $n$  ist gerade.

Wegen  $S_1(n)$  war der Automat aus und wechselt auf ‘ein’. Es gilt  $S_2(n+1)$

– Falls  $n+1$  gerade, dann gilt  $S_2(n+1)$  und  $n$  ist ungerade.

Wegen  $S_2(n)$  war der Automat ein und wechselt auf ‘aus’. Es gilt  $S_1(n+1)$

# SIMULTANE INDUKTION AUSFORMULIERT



Um zu zeigen, das der Automat ein Wechselschalter ist, zeigen wir durch Induktion, daß für alle  $n \in \mathbb{N}$  die folgenden beiden Aussagen gelten

$S_1(n)$ :  $n$  gerade  $\Leftrightarrow$  Automat ist nach  $n$ -fachem Drücken im Zustand “aus”

$S_2(n)$ :  $n$  ungerade  $\Leftrightarrow$  Automat ist nach  $n$ -fachem Drücken im Zustand “ein”

**Induktionsanfang  $n=0$ :**

$S_1(0)$ : 0 ist gerade und der Automat ist zu Beginn im Zustand “aus”, also gilt Aussage  $S_1(0)$ .

$S_2(0)$ : 0 ist nicht ungerade und der Automat ist zu Beginn nicht im Zustand “ein”, also gilt die Äquivalenz  $S_2(0)$ , da jeweils die rechte und linke Seite falsch ist.

**Induktionsannahme:** die Aussagen  $S_1(m)$  und  $S_2(m)$  seien für ein beliebiges  $m \in \mathbb{N}$  gezeigt.

**Induktionsschritt:** Es sei  $n = m+1$ .

$S_1(n)$ : Es ist  $n = m+1$  gerade

$\Leftrightarrow m$  ist ungerade

$\Leftrightarrow$  Automat ist nach  $m$ -fachem Drücken im Zustand “ein” (Induktionsannahme  $S_2(m)$ )

$\Leftrightarrow$  Automat ist nach  $n$ -fachem Drücken im Zustand “aus”

$S_2(n)$ : Es ist  $n = m+1$  ungerade

$\Leftrightarrow m$  ist gerade

$\Leftrightarrow$  Automat ist nach  $m$ -fachem Drücken im Zustand “aus” (Induktionsannahme  $S_1(m)$ )

$\Leftrightarrow$  Automat ist nach  $n$ -fachem Drücken im Zustand “ein”

Aufgrund des Induktionsprinzip gilt  $S_1(n)$  und  $S_2(n)$  für alle  $n \in \mathbb{N}$

# WIE GENAU / FORMAL MUSS EIN BEWEIS SEIN?

*“Ein Beweis ist ein Argument, das den Leser überzeugt”*

- Genau genug, um Details rekonstruieren zu können  
Knapp genug, um übersichtlich und merkbar zu sein  
Also **nicht notwendig formal oder mit allen Details**
  - Text muß **präzise Sprache** verwenden, lesbar und klar verständlich sein  
Formeln / Textfragmente ohne erkennbaren Sinn aneinanderzureihen ist unakzeptabel  
Zwischenschritte müssen **mit “üblichen” Vorkenntnissen erklärbar** sein
  - Gedankensprünge sind erlaubt, wenn Sie die Materie gut genug verstehen,  
dass Sie nichts mehr falsch machen können
  - ... es reicht nicht, dass Sie es einmal richtig gemacht haben
- 
- Tip: ausführliche Lösungen entwickeln, bis Sie genug Erfahrung haben.  
Bei Präsentation für Andere zentrale Gedanken aus Lösung **extrahieren**
  - Test: verstehen Kommilitonen Ihre Lösung und warum sie funktioniert?

[Mehr dazu in den Übungen](#)

# ALLGEMEINE METHODIK DES PROBLEMÖSENS

- **Klärung der Voraussetzungen**

- Welche Begriffe sind zum Verständnis des Problems erforderlich?
- Erstellung eines präzisen Modells: abstrahiere von Details
- Formulierung des Problems im Modell: was genau ist zu tun?

- **Lösungsweg konkretisieren**

- Welche Einzelschritte benötigt man, um das Problem zu lösen?
- Welches Gesamtergebnis ergibt sich aus den Einzelschritten?
- Wie beweist man die Korrektheit des Gesamtergebnisses?

- **Lösung zusammenfassen**

- Kurz und prägnant: Argumente auf das Wesentliche beschränken
- Wo möglich mathematisch präzise Formulierungen verwenden

# DIAGONALISIERUNGSBEWEISE

## Spezielle Form von Widerlegungsbeweisen

### Konstruktion von Gegenbeispielen für Aussagen über unendliche Objekte

- **Zeige:** Terminierung von Programmen ist unentscheidbar

*Es gibt kein Programm, das testen kann, ob ein beliebiges Programm bei einer bestimmten Eingabe überhaupt anhält*

- **Beweis stützt sich auf wenige Grundannahmen**

1. Programme und ihre Daten sind als Zahlen codierbar

**Schreibweise:**  $p_i(j) \hat{=} \text{Anwendung des } i\text{-ten Programms auf die Zahl } j$

2. Computer sind universelle Maschinen

Bei Eingabe von Programm und Daten berechnen sie das Ergebnis

3. Man kann Programme beliebig zu neuen Programmen zusammensetzen  
... und die Nummer des neuen Programms berechnen

# PROGRAMMTERMINIERUNG IST UNENTSCHEIDBAR

- Annahme: es gibt ein Programm für den Terminierungstest

–  $\text{Term}(i, j) = 1$  falls  $p_i(j)$  anhält (sonst 0)

- Konstruiere ein Programm **Unsinn** wie folgt:

$$\text{Unsinn}(i) = \begin{cases} 0 & \text{wenn } \text{Term}(i, i) = 0 \\ \perp & \text{sonst} \end{cases}$$

- Weil **Unsinn** ein Programm ist, muß es

eine Nummer **k** haben

$\times \hat{=} \text{Terminierung}, \perp \hat{=} \text{hält nicht}$

- Was macht **Unsinn**= $p_k$  bei Eingabe der eigenen Nummer als Daten?
  - Wenn  $p_k(k)$  hält, dann  $\text{Term}(k, k) = 1$ , also hält **Unsinn**( $k$ ) nicht an ???
  - Wenn  $p_k(k)$  nicht hält, dann  $\text{Term}(k, k) = 0$ , also hält **Unsinn**( $k$ ) an ???
- Dies ist ein Widerspruch,

Also kann es den Test auf Terminierung nicht geben

# ANTEILANTΩ

# MATHEMATISCHES VOKABULAR I: WÖRTER UND SPRACHEN

- **Alphabet  $\Sigma$** : endliche Menge von Symbolen,  
z.B.  $\Sigma = \{0, 1\}$ ,  $\Sigma = \{0, \dots, 9\}$ ,  $\Sigma = \{A, \dots, Z, a, \dots, z, , ?, !, ..\}$
- **Wörter**: endliche Folge  $w$  von Symbolen eines Alphabets  
Auch **Zeichenreihen** oder **Strings** genannt
- $\epsilon$ : Leeres Wort (ohne jedes Symbol)
- $w \ v$ : Konkatenation (Aneinanderhängung) der Wörter  $w$  und  $v$
- $u^i$ :  $i$ -fache Konkatenation des Wortes (oder Symbols)  $u$
- $|w|$ : Länge des Wortes  $w$  (Anzahl der Symbole)
- $v \sqsubseteq w$ :  $v$  Präfix von  $w$ , wenn  $w = v u$  für ein Wort  $u$
- $\Sigma^k$ : Menge der Wörter der Länge  $k$  mit Symbolen aus  $\Sigma$
- $\Sigma^*$ : Menge aller Wörter über  $\Sigma$
- $\Sigma^+$ : Menge aller nichtleeren Wörter über  $\Sigma$
- **Sprache  $L$** : Beliebige Menge von Wörtern über einem Alphabet  $\Sigma$   
Üblicherweise in abstrakter Mengennotation gegeben
- z.B.  $\{w \in \{0, 1\}^* \mid |w| \text{ ist gerade}\} \quad \{0^n 1^n \mid n \in \mathbb{N}\}$
- **Problem  $P$** : Menge von Wörtern über einem Alphabet  $\Sigma$   
Das “Problem” ist, Zugehörigkeit zur Menge  $P$  zu testen

## MATHEMATISCHES VOKABULAR II: FUNKTIONEN

- **Funktion  $f : S \rightarrow S'$ :** Abbildung zwischen den Grundmengen  $S$  und  $S'$ 
  - nicht unbedingt auf allen Elementen von  $S$  definiert
- **Domain von  $f$ :**  $\text{domain}(f) = \{x \in S \mid f(x) \text{ definiert}\}$  (Definitionsbereich)
- **Range von  $f$ :**  $\text{range}(f) = \{y \in S' \mid \exists x \in S. f(x) = y\}$  (Wertebereich)
- **$f$  total:**  $\text{domain}(f) = S$  (andernfalls ist  $f$  partiell)
- **$f$  injektiv:**  $x \neq y \Rightarrow f(x) \neq f(y)$
- **$f$  surjektiv:**  $\text{range}(f) = S'$
- **$f$  bijektiv:**  $f$  injektiv und surjektiv
- **Umkehrfunktion  $f^{-1} : S' \rightarrow S$ :**  $f^{-1}(y) = x \Leftrightarrow f(x) = y$  ( $f$  injektiv!)
- **Urbild  $f^{-1}(L)$ :** Die Menge  $\{x \in S \mid f(x) \in L\}$
- **Charakteristische Funktion  $\chi_L$  von  $L \subseteq S$ :** 
$$\chi_L(w) = \begin{cases} 1 & \text{falls } w \in L, \\ 0 & \text{sonst} \end{cases}$$
- **Partiell-charakteristische Funktion  $\psi_L$ :** 
$$\psi_L(w) = \begin{cases} 1 & \text{falls } w \in L, \\ \perp & \text{sonst} \end{cases}$$

---

Mehr Vokabular wird bei Bedarf vorgestellt

# Theoretische Informatik I

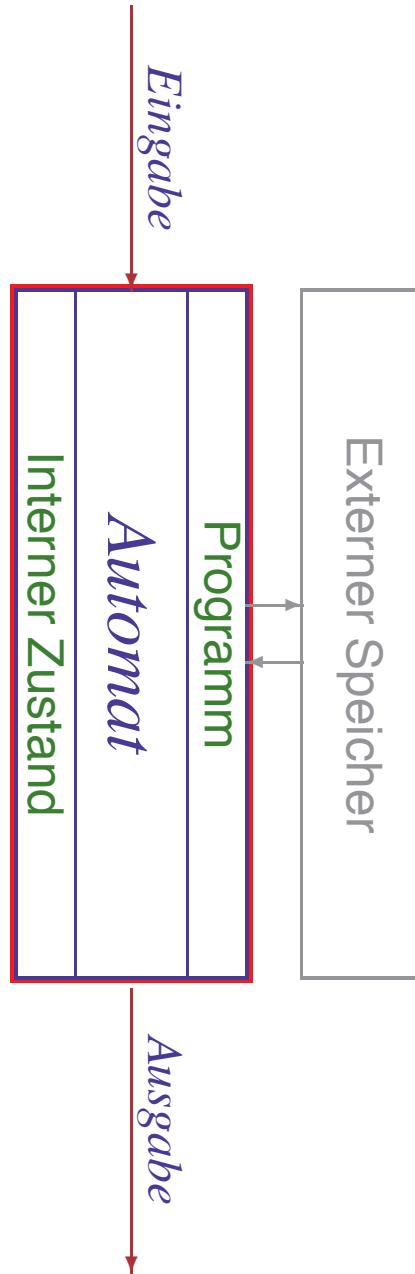
## Einheit 2

### Endliche Automaten & Reguläre Sprachen



1. Deterministische endliche Automaten
2. Nichtdeterministische Automaten
3. Reguläre Ausdrücke
4. Grammatiken
5. Eigenschaften regulärer Sprachen

# AUTOMATEN: DAS EINFACHSTE MASCHINENMODELL



- **Automaten stehen im Kern jeder Berechnung**
  - Schnelle, direkte Verarbeitung von Eingaben
  - Keine interne Speicherung von Daten
  - Speicher sind Teil der Umgebung
- **Endliche Automaten sind leicht zu analysieren**
  - Jede Berechnung endet nach einer festen Anzahl von Schritten
  - Keine Schleifen oder Seiteneffekte

# VERWENDUNGSZWECKE FÜR ENDLICHE AUTOMATEN

## Basismodell für viele Arten von Hard- & Software

- **Steuerungsautomaten**
  - Alle Formen rein Hardware-gesteuerter automatischer Maschinen  
Waschmaschinen, Autos, Unterhaltungselektronik, Ampelanlagen, Computerprozessoren
- **Entwurf und Überprüfung digitaler Schaltungen**
  - Entwicklungswerzeuge & Testsoftware beschreiben endliches Verhalten
- **Lexikalische Analyse in Compilern**
  - Schnelle Identifizierung von Bezeichnern, Schlüsselwörtern, ...
- **Textsuche in umfangreichen Dokumenten**
  - Z.B. Suche nach Webseiten mithilfe von Schlüsselwörtern
- **Software mit endlichen Alternativen**
  - Kommunikationsprotokolle, Protokolle zum sicheren Datenaustausch ...

# AUTOMATEN BESCHREIBEN SPRACHEN

- **Generierte Sprache**

- Menge aller möglichen Ausgaben des Automaten

- **Erkannte Sprache**

- Menge aller Eingaben, die zur Ausgabe “ja” führen
- Alternativ: letzter Zustand des Automaten muß ein “Endzustand” sein

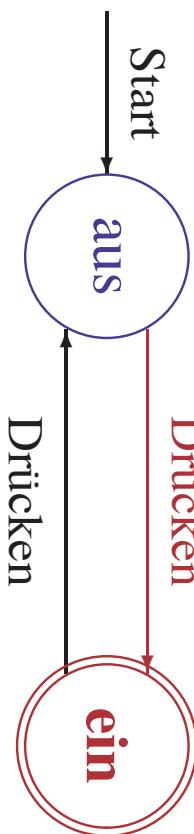
- **Sprachen endlicher Automaten sind einfach**

- Nur sehr einfach strukturierte Sprachen können beschrieben werden
- Durch endliche Automaten beschreibbare Sprachen heißen **regular**

# MODELLE ZUR BESCHREIBUNG REGULÄRER SPRACHEN

- **Automaten:** erkennen von Wörtern

- z.B. Wechselschalter: Verarbeitung von “Drück”-Eingaben



- **Zustände:** aus, ein – **Startzustand:** aus – **Endzustand:** ein

- **Eingabesymbol:** Drücken

- Endzustand wird erreicht bei ungerader Anzahl von Drücken

- **Mathematische Mengennotation**

- z.B.:  $\{\text{Drücken}^{2i+1} \mid i \in \mathbb{N}\}$  oder  $\{w \in \{\text{Drücken}\}^* \mid \exists i \in \mathbb{N}. |w| = 2i+1\}$

- **Reguläre Ausdrücke: algebraische Strukturen**

- z.B.:  $(\text{DrückenDrücken})^* \text{Drücken}$

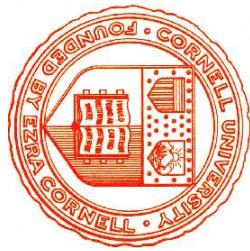
- **Grammatiken: Vorschriften für Spracherzeugung**

- z.B.:  $S \rightarrow \text{Drücken}, S \rightarrow S \text{DrückenDrücken}$
- Erzeugt nur ungerade Anzahl von Drücken-Symbolen

# Theoretische Informatik I

## Einheit 2.1

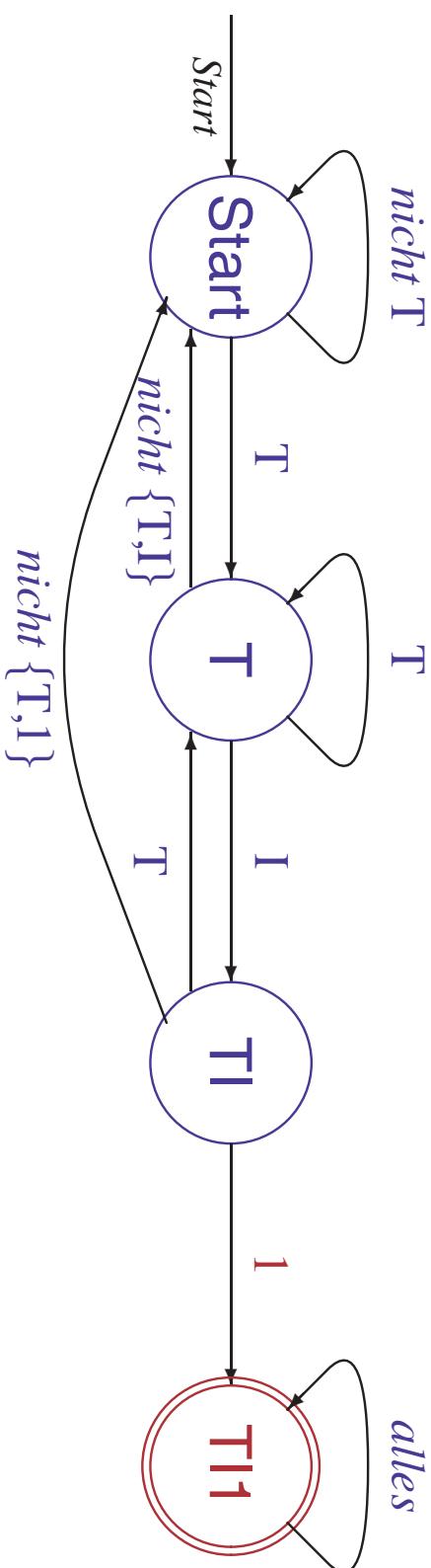
### Deterministische Endliche Automaten



1. Arbeitsweise
2. Akzeptierte Sprache
3. Entwurf und Analyse
4. Automaten mit Ausgabe

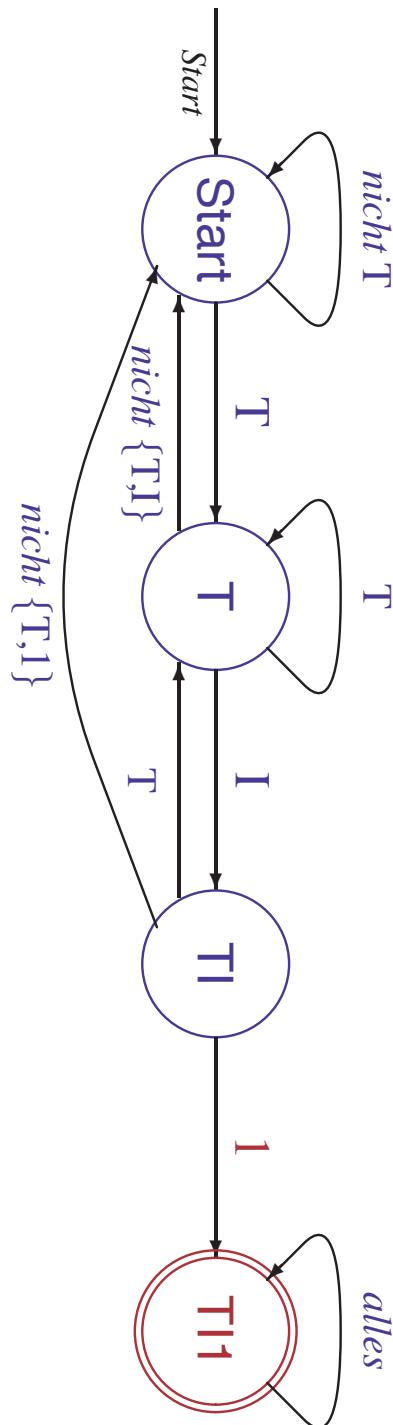


# ERKENNUNG VON WÖRTERN MIT AUTOMATEN



- Endliche Anzahl von **Zuständen**
- Ein **Startzustand**
- Regeln für **Zustandsübergänge**
- **Eingabealphabet:**  $\{A, \dots, Z, a, \dots, z, 0, \dots, 9, ?, !, \dots\}$
- Ein oder mehrere akzeptierende **Endzustände**

# ENDLICHE AUTOMATEN – MATHEMATISCH PRÄZISIERT



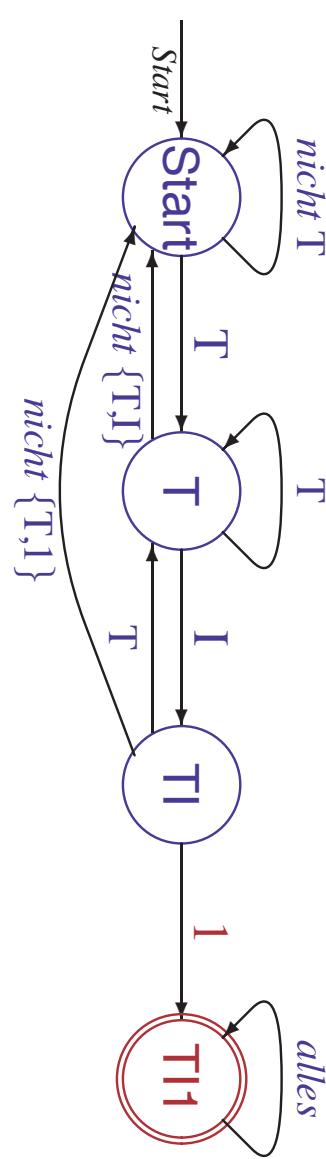
## Ein Deterministischer Endlicher Automat (DEA)

ist ein 5-Tupel  $A = (Q, \Sigma, \delta, q_0, F)$  mit

- $Q$  nichtleere endliche **Zustandsmenge**
- $\Sigma$  (endliches) **Eingabealphabet**
- $\delta: Q \times \Sigma \rightarrow Q$  **Zustandsübergförfungsfunktion**
- $q_0 \in Q$  **Startzustand**
  - (Anfangszustand)
  - (Endzustände)
  - (Finale Zustände)
- $F \subseteq Q$  Menge von **akzeptierenden Zuständen**

# BESCHREIBUNG VON ENDLICHEN AUTOMATEN

- Übergangsdiagramm



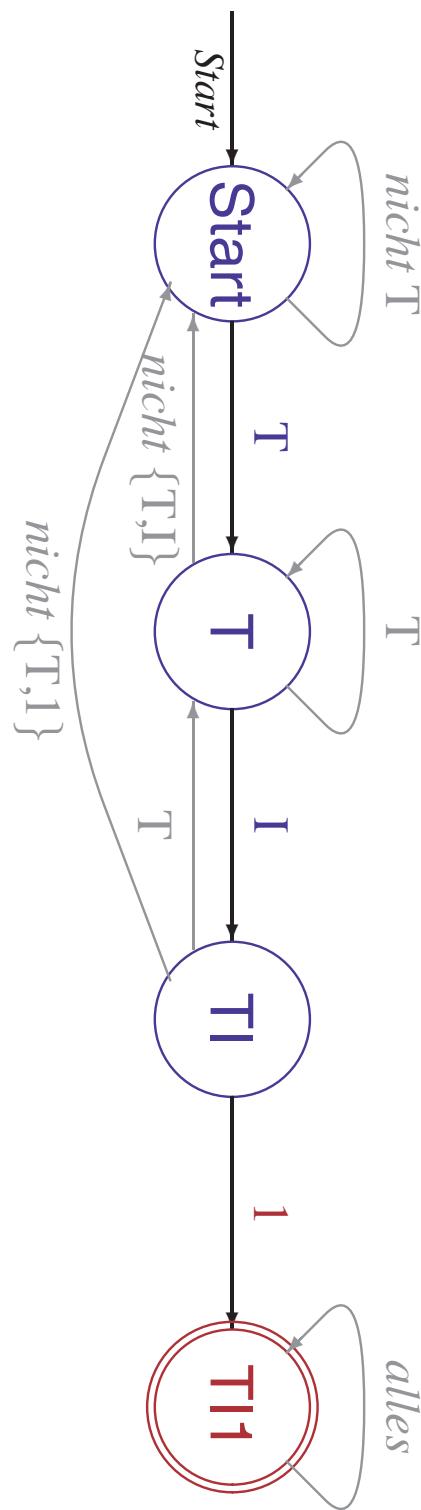
- Jeder Zustand in  $Q$  wird durch einen Knoten (Kreise) dargestellt
- Ist  $\delta(q, a) = p$ , so verläuft eine Kante von  $q$  nach  $p$  mit Beschriftung  $a$  (mehrere Beschriftungen derselben Kante möglich)
- $q_0$  wird durch einen mit *Start* beschrifteten Pfeil angezeigt
- Endzustände in  $F$  werden durch **doppelte Kreise** gekennzeichnet
- $\Sigma$  meist implizit durch Diagramm bestimmt

- Übergangstabelle

- Tabellarische Darstellung der Funktion  $\delta$
- Kennzeichnung von  $q_0$  durch einen Pfeil
- Kennzeichnung von  $F$  durch Sterne
- $\Sigma$  und  $Q$  meist implizit durch Tabelle bestimmt

	T	I	I	sonst
→ S	T	S	S	S
T	T	I	S	S
I	T	S	1	S
*	1	1	1	1
1	1	1	1	1

# ARBEITSWEISE VON ENDLICHEN AUTOMATEN



- **Anfangssituation**
  - Automat befindet sich im Startzustand  $q_0$
- **Arbeitschritt**
  - Im Zustand  $q$  lese Eingabesymbol  $a$ ,
  - Bestimme  $\delta(q,a)=p$  und wechsle in neuen Zustand  $p$
- **Terminierung**
  - Eingabewort  $w = a_1..a_n$  ist komplett gelesen, Automat im Zustand  $q_n$
- **Ergebnis**
  - Eingabewort  $w$  wird akzeptiert, wenn  $q_n \in F$ , sonst wird  $w$  abgewiesen

# ARBEITSWEISE VON DEAs – MATHEMATISCH PRÄZIERT

## • Erweiterte Überführungsfunktion $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$

- Schrittweise Abarbeitung der Eingabe mit  $\delta$  von links nach rechts
- Informal:  $\hat{\delta}(q, w_1 w_2 \dots w_n) = \delta(\dots(\delta(\delta(q, w_1), w_2), \dots), w_n)$
- Mathematisch präzise Beschreibung benötigt induktive Definition

$$\hat{\delta}(q, w) = \begin{cases} q & \text{falls } w = \epsilon, \\ \delta(\hat{\delta}(q, v), a) & \text{falls } w = v a \text{ für ein } v \in \Sigma^*, a \in \Sigma \end{cases}$$

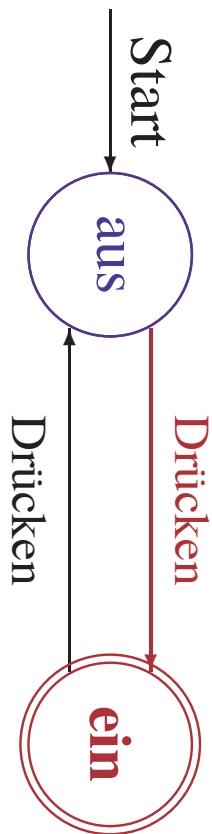
## • Von A akzeptierte Sprache

- Menge der Eingabewörter  $w$ , für die  $\hat{\delta}(q_0, w)$  akzeptierender Zustand ist
- $L(A) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}$
- Auch: die von A erkannte Sprache

## • Reguläre Sprache

- Sprache, die von einem DEA A akzeptiert wird

# ANALYSE DER SPRACHE DES WECHSELALSCHALTERS



- **Sprache: Eingaben, für die Automat eingeschaltet ist**

- Teilmenge der Wörter über dem Alphabet  $\Sigma = \{\text{Drücken}\}$

- **Automat  $A$  ist ein Wechselschalter**

Zwei Eigenschaften sind zu zeigen:

- $S_1(n)$ : Ist  $n$  gerade, so ist  $\hat{\delta}(\text{aus}, \text{Drücken}^n) = \text{aus}$
- $S_2(n)$ : Ist  $n$  ungerade, so ist  $\hat{\delta}(\text{aus}, \text{Drücken}^n) = \text{ein}$

Beweis durch simultane Induktion (vgl. Einheit 1, Folie 8)

- **Formale Beschreibung der Sprache von  $A$**

$$L(A) = \{w \in \text{Drücken}^* \mid \hat{\delta}(\text{aus}, w) \in \{\text{ein}\}\} = \{\text{Drücken}^{2i+1} \mid i \in \mathbb{N}\}$$

# ENTWURF UND ANALYSE ENDLICHER AUTOMATEN

Entwerfe Automaten für  $L = \{u01v \mid u, v \in \{0, 1\}^*\}$

- Drei Zustände sind erforderlich

- Zustand  $q_0$ : A hat noch keine 0 gelesen
- Zustand  $q_1$ : A hat eine 0 aber noch keine 1 gelesen
- Zustand  $q_2$ : A hat eine Zeichenkette 01 gelesen

$1^i$  bleibt in  $q_0$

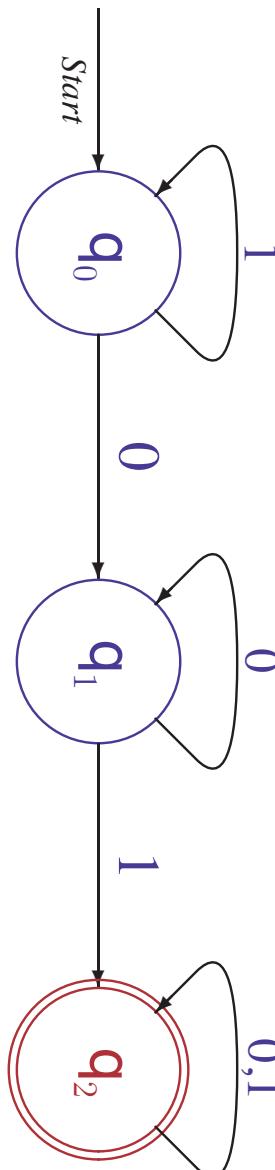
$1^{i+1}0^{j+1}$  bleibt in  $q_1$

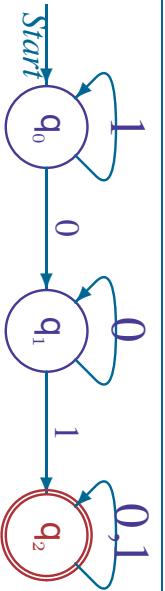
$1^i0^j01v$  bleibt in  $q_2$

- Zustandsübergänge erhalten ‘Bedeutung’

- Zustand  $q_0$ : Mit 1 bleibe in  $q_0$ , sonst wechsele nach  $q_1$
- Zustand  $q_1$ : Mit 0 bleibe in  $q_1$ , sonst wechsele nach  $q_2$
- Zustand  $q_2$ : Bleibe bei jeder Eingabe in  $q_2$ , Endzustand

- Zugehöriger DEA mit Alphabet  $\Sigma = \{0, 1\}$





$ZEIGE \ L(A) = L = \{u01v \mid u, v \in \{0, 1\}^*\}$

- Zeige durch strukturelle Induktion über  $w$ :

$$-\hat{\delta}(q_0, w) = q_0 \Leftrightarrow \text{es gibt ein } i \in \mathbb{N} \text{ mit } w = 1^i$$

Basisfall  $w = \epsilon$ : Per Definition ist  $\hat{\delta}(q_0, \epsilon) = q_0$  und  $w = 1^i$  für  $i = 0$  ✓

Schrittfall  $w = ua$  für ein  $u \in \Sigma^*, a \in \Sigma$ :

- Es gelte  $\hat{\delta}(q_0, w) = q_0$ . Dann ist  $\hat{\delta}(q_0, u) = q_0$  und  $\delta(q_0, a) = q_0$ .

Es folgt  $a = 1$  und per Annahme  $u = 1^i$  für ein  $i$ , also  $w = 1^{i+1}$ . ✓

- Es gelte  $w = 1^i$ . Dann ist  $a = 1$  und  $u = 1^{i-1}$ . Mit der Induktions-

annahme folgt  $\hat{\delta}(q_0, w) = \delta(\hat{\delta}(q_0, u), a) = \delta(q_0, a) = q_0$  ✓

$$-\hat{\delta}(q_0, w) = q_1 \Leftrightarrow \text{es gibt } i, j \in \mathbb{N} \text{ mit } w = 1^i 0^j + 1$$

$$-\hat{\delta}(q_0, w) = q_2 \Leftrightarrow \text{es gibt } i, j \in \mathbb{N}, v \in \Sigma^* \text{ mit } w = 1^i 0^j + 1 v$$

analog

- Zeige:  $w \in L \Leftrightarrow \text{es gibt } i, j \in \mathbb{N}, v \in \Sigma^* \text{ mit } w = 1^i 0^j 0^i v$

⇒ Für  $w \in L$  gibt es  $u, v \in \Sigma^*$  mit  $w = u01v$

Wenn  $u$  nicht die Form  $1^i 0^j$  hat, dann folgt in  $u$  eine 1 auf eine 0.

Das erste solche Vorkommen von 01 liefert die gewünschte Zerlegung ✓

- Es folgt  $w \in L \Leftrightarrow \hat{\delta}(q_0, w) = q_2 \in F \Leftrightarrow w \in L(A)$

- **Konfiguration:** ‘Gesamtzustand’ von Automaten

- Mehr als  $q \in Q$ : auch die noch unverarbeitete Eingabe zählt
- Formal dargestellt als Tupel  $\textcolor{red}{K} = (q, w) \in Q \times \Sigma^*$

- **Konfigurationsübergangsrelation**  $\vdash^*$

- Wechsel zwischen Konfigurationen durch Abarbeitung von Wörtern
- $(q, aw) \vdash (p, w)$ , falls  $\delta(q, a) = p$
- $\textcolor{red}{K}_1 \vdash^* \textcolor{red}{K}_2$ , falls  $K_1 = K_2$  oder  
es gibt eine Konfiguration  $K$  mit  $K_1 \vdash K$  und  $K \vdash^* K_2$

- **Akzeptierte Sprache**

- Menge der Eingaben, für die  $\vdash^*$  zu akzeptierendem Zustand führt

$$L(A) = \{w \in \Sigma^* \mid \exists p \in F. (q_0, w) \vdash^* (p, \epsilon)\}$$

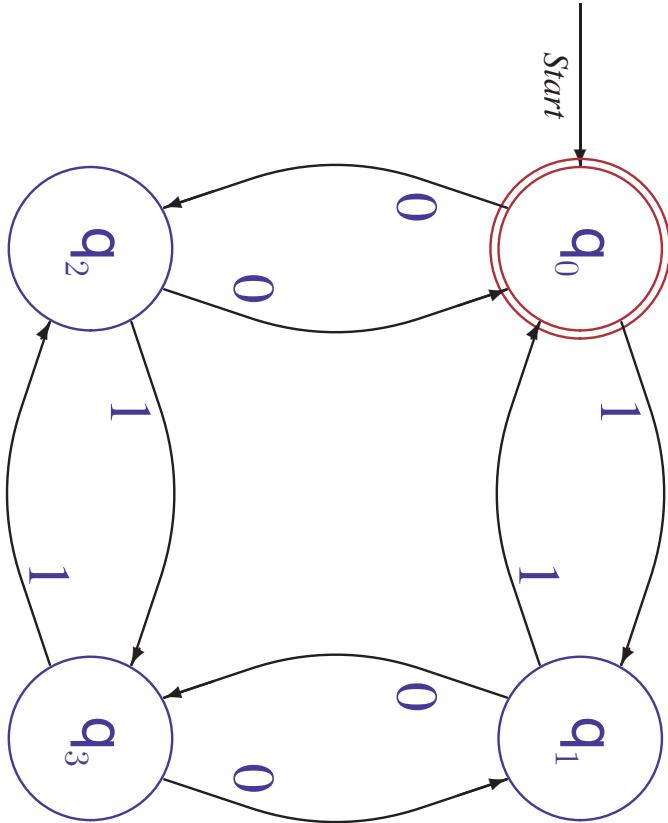
Für DEAs weniger intuitiv, aber leichter zu verallgemeinern

DEA FÜR  $L = \{w \in \{0, 1\}^* \mid w \text{ enthält gerade Anzahl von } 0 \text{ und } 1\}$

## Codiere Anzahl der gelesener 0/1 im Zustand

$q_0 \hat{=} (\text{gerade}, \text{gerade}) \quad q_1 \hat{=} (\text{gerade}, \text{ungerade})$

$q_2 \hat{=} (\text{ungerade}, \text{gerade}) \quad q_3 \hat{=} (\text{ungerade}, \text{ungerade})$



## Korrektheit: gegenseitige strukturelle Induktion

# KORREKTHEITSBEWEIS MIT KONFIGURATIONEN

- Zeige simultan für alle Wörter  $w, v \in \{0, 1\}^*$ :

- (1)  $(q_0, wv) \vdash^* (q_0, v) \Leftrightarrow$  es gilt  $g_0(w)$  und  $g_1(w)$
- (2)  $(q_0, wv) \vdash^* (q_1, v) \Leftrightarrow$  es gilt  $g_0(w)$  und  $u_1(v)$
- (3)  $(q_0, wv) \vdash^* (q_2, v) \Leftrightarrow$  es gilt  $u_0(w)$  und  $g_1(v)$
- (4)  $(q_0, wv) \vdash^* (q_3, v) \Leftrightarrow$  es gilt  $u_0(w)$  und  $u_1(v)$

$g_0(w) \hat{=} w$  hat gerade Anzahl von Nullen,  $u_0(w) \hat{=} w$  hat ungerade Anzahl von Nullen, ...

- Basisfall  $w = \epsilon$ :

– Per Definition gilt  $(q_0, v) \vdash^* (q_0, v)$  und  $g_0(w)$  und  $g_1(w)$  ✓

- Schrittfall  $w = ua$  für ein  $u \in \Sigma^*, a \in \Sigma$ :

- (1) Es gelte  $(q_0, wv) \vdash^* (q_0, v)$ .

Dann gilt  $(q_0, uav) \vdash^* (p, av) \vdash (q_0, v)$  für einen Zustand  $p$ .

Falls  $a = 0$ , dann ist  $p = q_2$  und nach (3) folgt  $u_0(u)$  und  $g_1(v)$ .

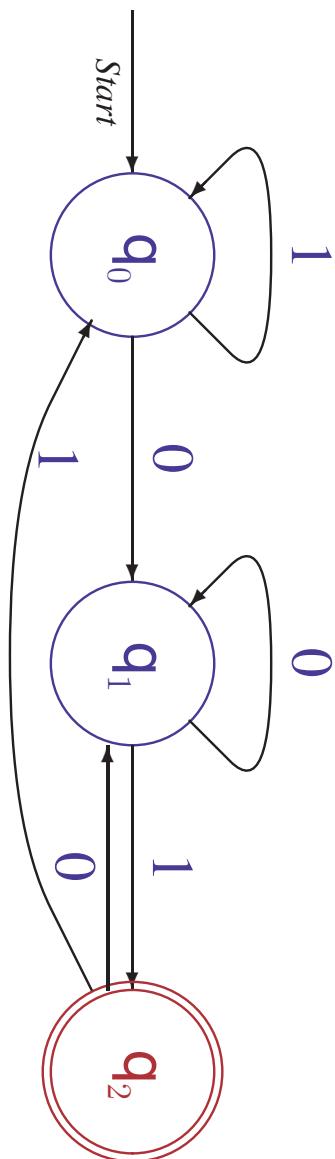
Für  $w = ua$  folgt somit  $g_0(w)$  und  $g_1(w)$ . ✓

Fall  $a=1$  analog. Gegenrichtung durch Umkehrung des Arguments. (2), (3), (4) analog.

- Es folgt  $w \in L(A) \Leftrightarrow (q_0, w) \vdash^* (q_0, \epsilon)$   
 $\Leftrightarrow g_0(w)$  und  $g_1(w) \Leftrightarrow w \in L$

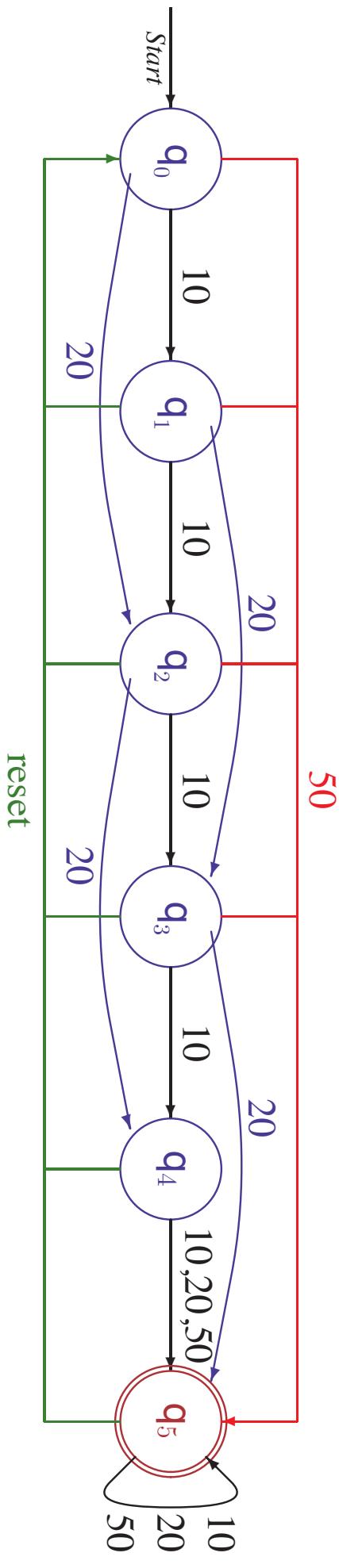
# WEITERE BEISPIELE ENDLICHER AUTOMATEN

- Erkenne Strings, die mit 0 1 enden



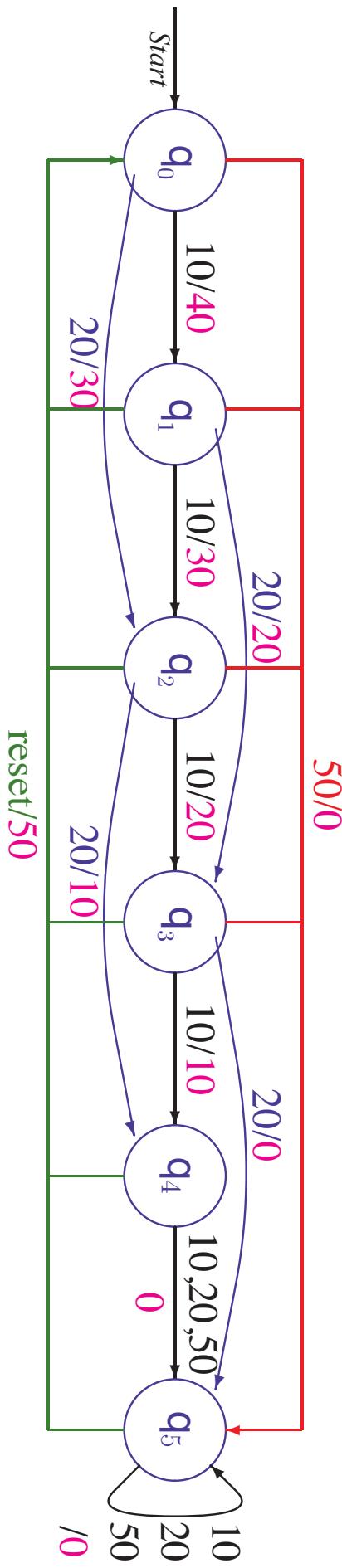
## • 50c Kaffeeautomat

- Akzeptiert 10,20,50c Münzen, gibt kein Geld zurück, mit Reset-Taste



# ENDLICHE AUTOMATEN MIT AUSGABEFUNKTION

- **50c Kaffeearmat mit Restbetragsanzeige**



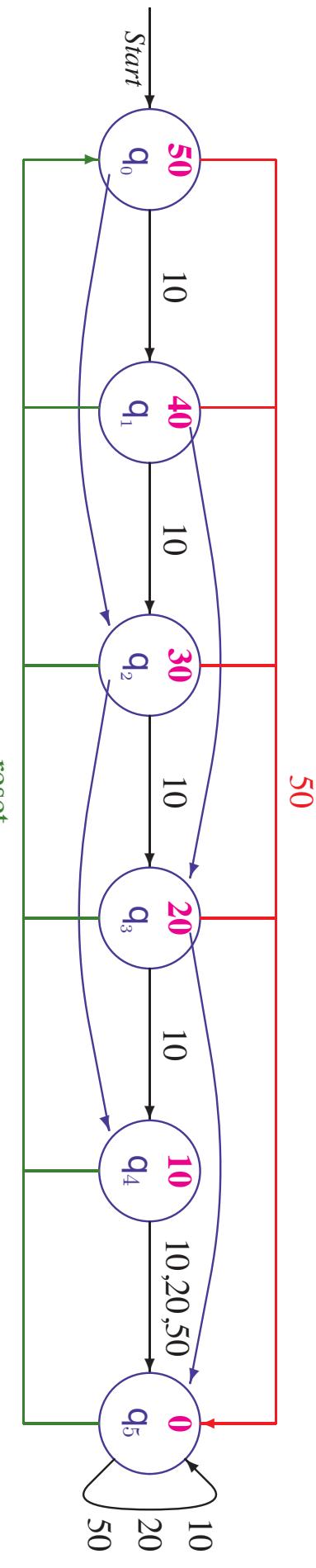
- Münzeinwurf führt zu Zustandsänderung und erzeugt Ausgabe

- **Formalisierungen von Automaten mit Ausgabe**

- **Mealy-Automaten**: Ausgabefunktion abhängig von Eingabe & Zustand
- **Moore-Automaten**: Ausgabefunktion nur von Zustand abhängig

**Beide Modelle sind äquivalent**

# MOORE-AUTOMATEN – MATHEMATISCH PRÄZIERT



Ein **Moore-Automat** ist ein 6-Tupel  $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$

- $Q$  nichtleere endliche **Zustandsmenge**
- $\Sigma$  (endliches) **Eingabealphabet**
- $\Delta$  (endliches) **Ausgabealphabet**
- $\delta: Q \times \Sigma \rightarrow Q$  **Zustandsübergföhrungsfunktion**
- $\lambda: Q \rightarrow \Delta$  **Ausgabefunktion**
- $q_0 \in Q$  **Startzustand**

# ARBEITSWEISE VON MOORE-AUTOMATEN ANALOG ZU DEAS

- **Anfangssituation:** Automat im Startzustand  $q_0$ , Ausgabe  $x_0 = \lambda(q_0)$
- **Arbeitschritt**
  - Im Zustand  $q$  lese Eingabesymbol  $a$ ,
  - Bestimme  $\delta(q,a)=p$  und wechsele in neuen Zustand  $p$
  - Bestimme  $x = \lambda(p)$  und gebe dieses Symbol aus
- **Terminierung:** Eingabewort  $w = a_1..a_n$  ist komplett gelesen
- **Ausgabewort:** Verkettung der ausgegebenen Symbole  $x_0x_1..x_n$

---

- **Erweiterte Ausgabefunktion**  $\hat{\lambda}: Q \times \Sigma^* \rightarrow \Delta^*$ 
  - Schrittweise Erzeugung der Ausgabe mit Abarbeitung der Eingabe
  - Formal: Induktive Definition
- $$\hat{\lambda}(q, w) = \begin{cases} \lambda(q) & \text{falls } w=\epsilon, \\ \hat{\lambda}(q, v) \circ \lambda(\hat{\delta}(q, v), a) & \text{falls } w=v a \text{ für ein } a \in \Sigma \end{cases}$$
- **Von  $M$  berechnete Funktion:**  $f_M(w) = \hat{\lambda}(q_0, w)$

# MOORE-AUTOMAT FÜR DIVISIONSREST

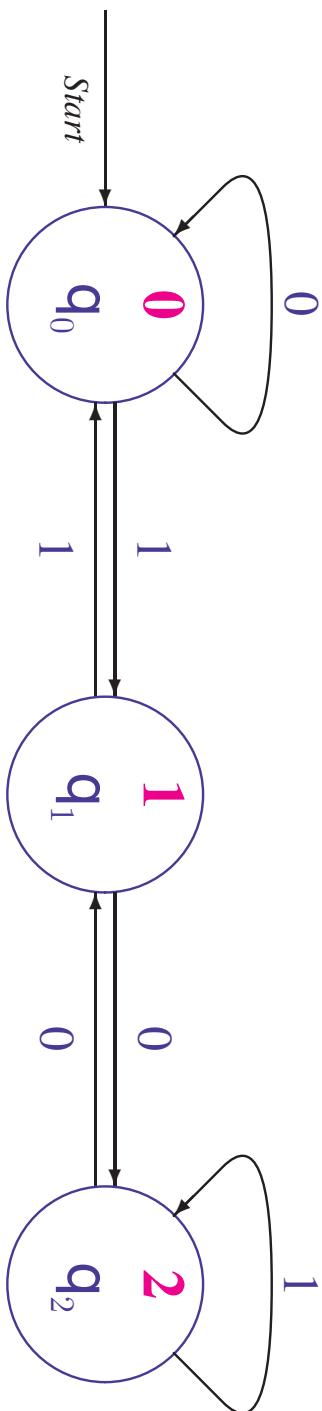
- **Eingabe einer Bitfolge von links nach rechts**

- Ausgabe ist jeweils “ $n \bmod 3$ ” für die bisher dargestellte Zahl  $n$
- Eingabealphabet  $\Sigma = \{0, 1\}$ , Ausgabealphabet  $\Delta = \{0, 1, 2\}$

- **Drei Zustände sind erforderlich**

- Zustand  $q_0$ : Bisheriger Divisionsrest ist 0 (Ausgabe 0)
- Zustand  $q_1$ : Bisheriger Divisionsrest ist 1 (Ausgabe 1)
- Zustand  $q_2$ : Bisheriger Divisionsrest ist 2 (Ausgabe 2)
- Zustandsübergangsregel  $\delta(q_i, j) = q_{2*i + j \bmod 3}$

- **Zugehöriger Moore-Automat**



# MOORE-AUTOMATEN SIND ÄQUIVALENT ZU DEAS

## Gegenseitige Simulation ist möglich

- Jede Sprache  $L$  ist als Funktion beschreibbar

$$-\chi_L(w) = \begin{cases} 1 & \text{falls } w \in L, \\ 0 & \text{sonst} \end{cases}$$

charakteristische Funktion von  $L$

– Charakteristische Funktionen akzeptierter Sprachen sind berechenbar

Satz:  $L$  regulär  $\Leftrightarrow \chi_L$  “Moore-berechenbar”

- Jede Funktion  $f$  ist als Menge beschreibbar

- $\text{graph}(f) = \{(w, v) \mid f(w) = v\}$
- $\text{graph}^*(f) = \{(w_1, v_0, v_1) .. (w_n, v_n) \mid f(w_1..w_n) = v_0..v_n\}$
- DEAs können Graphen berechneter Funktionen akzeptieren

Satz:  $f$  Moore-berechenbar  $\Leftrightarrow \text{graph}^*(f)$  reguläre Sprache

## Beweis der Äquivalenz (Skizze)

- **$L$  regulär  $\Leftrightarrow \chi_L$  “Moore-berechenbar”**

– Zu  $A = (Q, \Sigma, \delta, q_0, F)$  konstruiere  $M = (Q, \Sigma, \{0,1\}, \delta, \lambda, q_0)$

mit  $\lambda(q) = \begin{cases} 1 & \text{falls } q \in F, \\ 0 & \text{sonst} \end{cases}$

- Dann ist  $w \in L(A)$  genau dann, wenn  $f_M(w) = v1$  für ein  $v \in \{0, 1\}^*$
- $\chi_L(w)$  ist das letzte Ausgabesymbol von  $f_M(w)$

- **$f$  Moore-berechenbar  $\Leftrightarrow$  graph\*( $f$ ) regulär**

– Zu  $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$  konstruiere  $A = (Q \cup \{q_s, q_f\}, \Sigma', \delta', q_s, Q)$

mit  $\Sigma' = \Sigma \times (\Delta \cup \{\lambda(q_0)\} \times \Delta)$

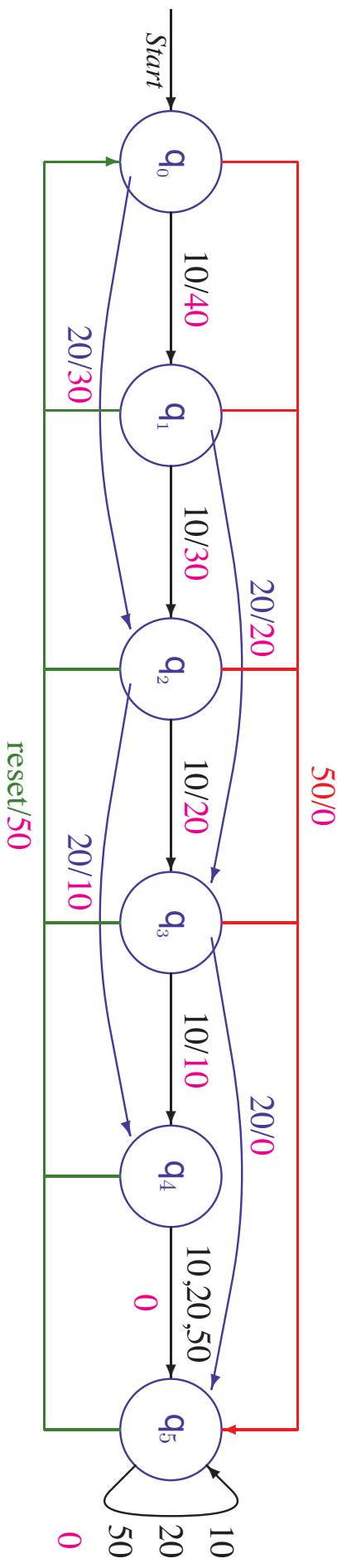
$$\delta'(q, (a, b)) = \begin{cases} \delta(q_0, a) & \text{falls } q = q_s, b = (\lambda(q_0), b_1), \lambda(\delta(q_0, a)) = b_1, \\ \delta(q, a) & \text{falls } \lambda(\delta(q, a)) = b, \\ q_f & \text{sonst} \end{cases}$$

- Dann  $f_M(w_1..w_n) = v_0..v_n$  genau dann, wenn  $(w_1, v_0, v_1)..(w_n, v_n) \in L(A)$

Mehr zu Automaten mit Ausgabe im Buch von Vossen & Witt

# ANTEILANG

# MEALY-AUTOMATEN – MATHEMATISCH PRÄZIERT



Ein **Mealy-Automat** ist ein 6-Tupel  $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$

- $Q$  nichtleere endliche **Zustandsmenge**
- $\Sigma$  (endliches) **Eingabealphabet**
- $\Delta$  (endliches) **Ausgabealphabet**
- $\delta: Q \times \Sigma \rightarrow Q$  **Zustandsübergangsfunktion**
- $\lambda: Q \times \Sigma \rightarrow \Delta$  **Ausgabefunktion**
- $q_0 \in Q$  **Startzustand**

# ARBEITSWEISE VON MEALY-AUTOMATEN ANALOG ZU DEAS

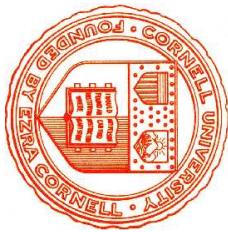
- **Anfangssituation:** Automat im Startzustand  $q_0$
  - **Arbeitschritt**
    - Im Zustand  $q$  lese Eingabesymbol  $a$ ,
    - Bestimme  $\delta(q,a)=p$  und wechsele in neuen Zustand  $p$
    - Bestimme  $x = \lambda(q,a)$  und gebe dieses Symbol aus
  - **Terminierung:** Eingabewort  $w = a_1..a_n$  ist komplett gelesen
  - **Ausgabewort:** Verkettung der ausgegebenen Symbole  $x_1..x_n$
- 
- **Erweiterte Ausgabefunktion**  $\hat{\lambda}: Q \times \Sigma^* \rightarrow \Delta^*$ 
    - Schrittweise Erzeugung der Ausgabe mit Abarbeitung der Eingabe
    - Formal: Induktive Definition
  - **Von  $M$  berechnete Funktion:**  $f_M(w) = \hat{\lambda}(q_0, w)$

$$\hat{\lambda}(q, w) = \begin{cases} \epsilon & \text{falls } w=\epsilon, \\ \hat{\lambda}(q, v) \circ \lambda(\hat{\delta}(q, v), a) & \text{falls } w=v a \text{ für ein } a \in \Sigma \end{cases}$$

# Theoretische Informatik I

## Einheit 2.2

### Nichtdeterministische Automaten

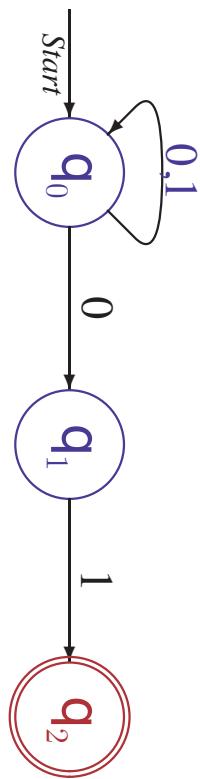


1. Arbeitsweise
2. Akzeptierte Sprache
3. Äquivalenz zu deterministischen Automaten

# WAS IST NICHTDETERMISMUS?

- **Verhalten nicht eindeutig bestimmt**

- Automat wählt Folgezustand aus mehreren Möglichkeiten

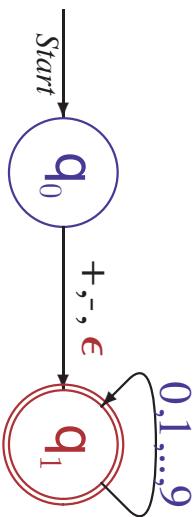


- Automat erkennt Strings, die mit 01 enden

- Eine 0 kann das erste Symbol des Endes 01 sein ... oder auch nicht

- **Spontane Übergänge zwischen Zuständen**

- Automat geht ohne Eingabe in anderen Zustand über ( $\epsilon$ -Übergang)



- Automat erkennt ganze Zahlen mit und ohne Vorzeichen

- **Hilfreiches Modell für Entwurfsphase**

- Elegantere Beschreibungsform, leichter als korrekt nachzuweisen
- Begrenzte physikalische Realisierung durch Parallelrechner möglich

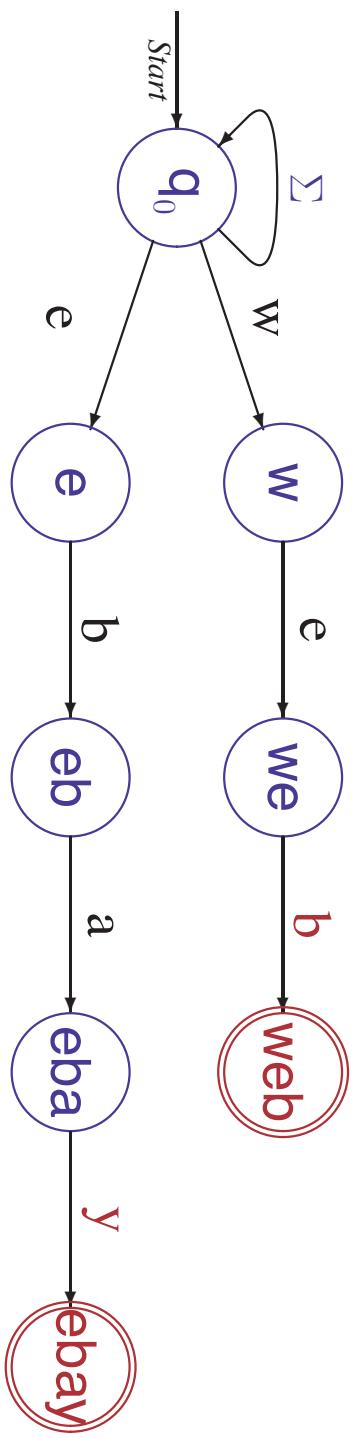
# NICHTDETERMINISTISCHE AUTOMATEN – WOZU?

- **Elegante Form** der Textsuche in Dokumenten

- Viele verschiedene Wörter in großen Textsammlungen (Internet)
- Leichte Beschreibung der Suchanfrage
- Deterministisches Erkennungsverfahren mühsam zu beschreiben

- **Idee: Simultane Verarbeitung von Alternativen**

- z.B. Suche nach den Wörtern **web** und **ebay** am Ende eines Wortes



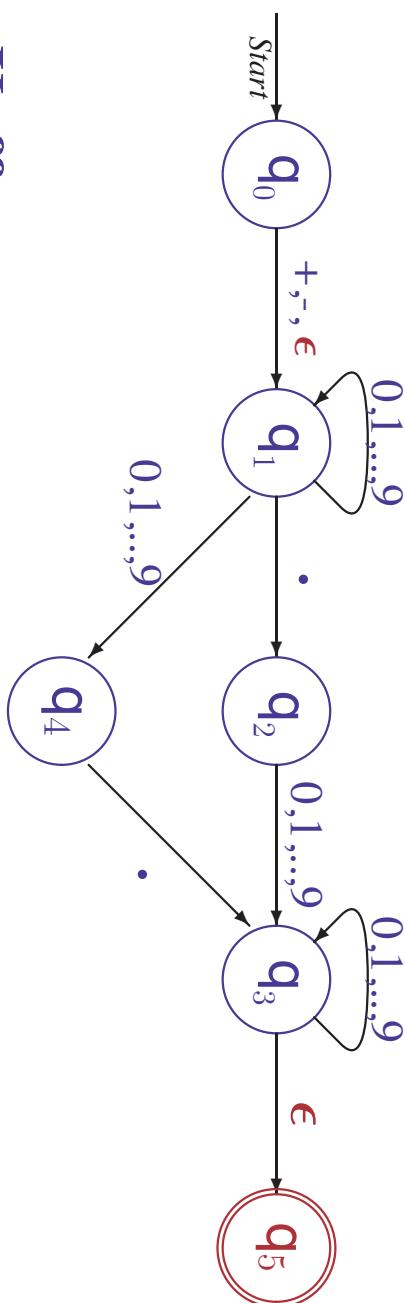
- Ein **w** könnte der Anfang von **web** sein
- Ein **e** könnte der Anfang von **ebay** sein
- Aber vor den Wörtern könnte noch etwas anderes stehen

**Nichtdeterminismus  $\hat{=}$  verfolge alle Möglichkeiten simultan**

## • ÜBERGÄNGE – VERARBEITUNG OPTIONALER EINGÄBEN

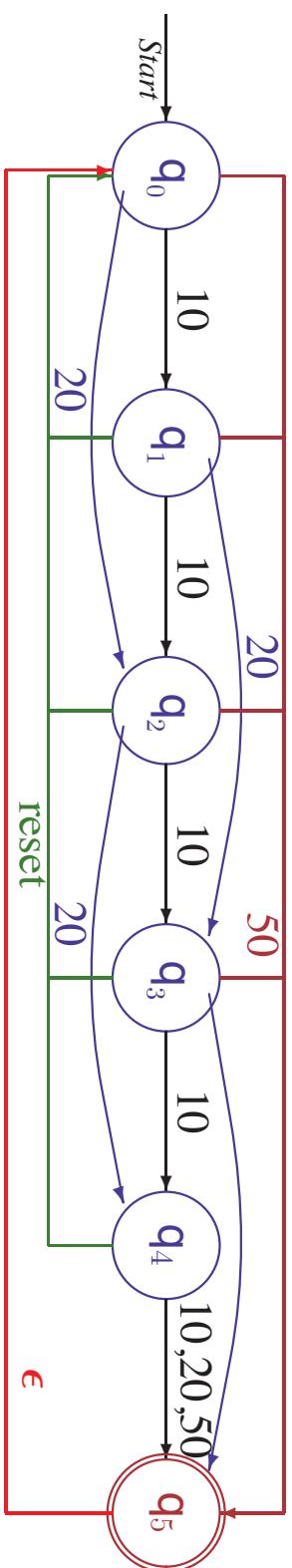
### • Erkenne Dezimalzahlen im Programmcode

- Zwei Zeichenreihen von Ziffern getrennt durch Dezimalpunkt
- Eine der beiden Zeichenreihen darf leer sein, aber nicht beide
- Optionales Vorzeichen + oder –

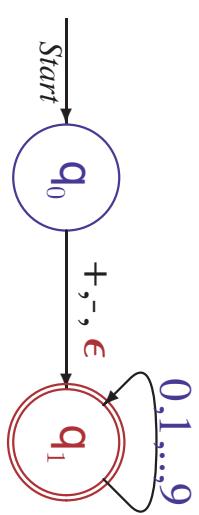


### • 50c Kaffeautomat

- Akzeptiert 10,20,50c, mit Reset-Taste und automatischer Rücksetzung



# NICHTDETERMINISTISCHE AUTOMATEN – PRÄZISIERT



Ein  **$\epsilon$ -NEA** (nichtdeterministischer endlicher Automat mit  $\epsilon$ -Übergängen) ist ein 5-Tupel  $A = (Q, \Sigma, \delta, q_0, F)$  mit

- $Q$  nichtleere endliche **Zustandsmenge**
- $\Sigma$  (endliches) **Eingabealphabet** mit  $\epsilon \notin \Sigma$
- $\delta: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)$  **Zustandsübergfungsfunktion** \*
- $q_0 \in Q$  **Startzustand**
- $F \subseteq Q$  Menge von **akzeptierenden** (End-) **Zuständen**

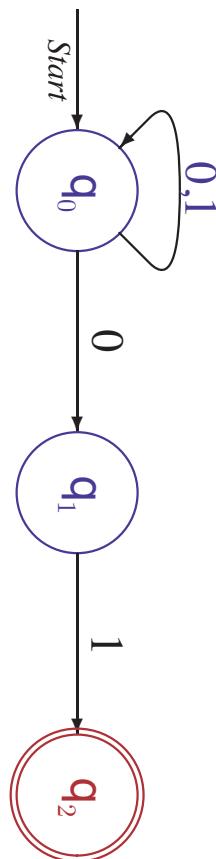
Ein **NEA** ist ein nichtdet. endlicher Automat ohne  $\epsilon$ -Übergänge

\*  $\mathcal{P}(Q) = \{S \mid S \subseteq Q\}$  (**Potenzmenge** von  $Q$ )

Bei  $\epsilon$ -NEAs ist  $\delta(q', a)$  ist eine (möglicherweise leere) Menge von Zuständen

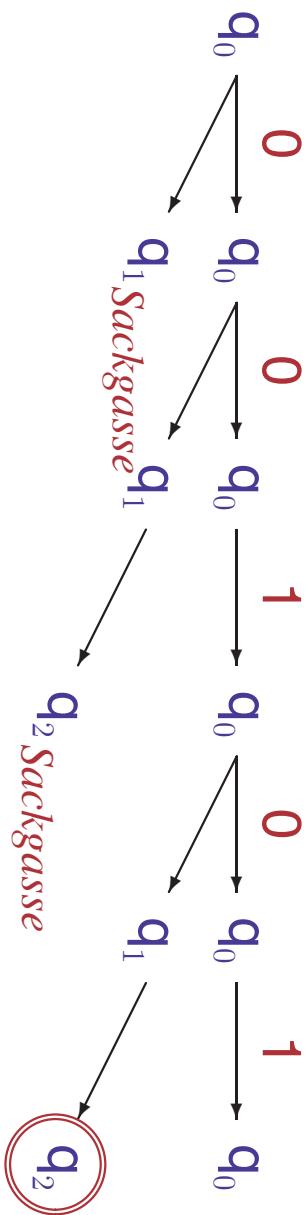
## ARBEITSWEISE VON NEAS

### Erkenne Strings, die mit 01 enden



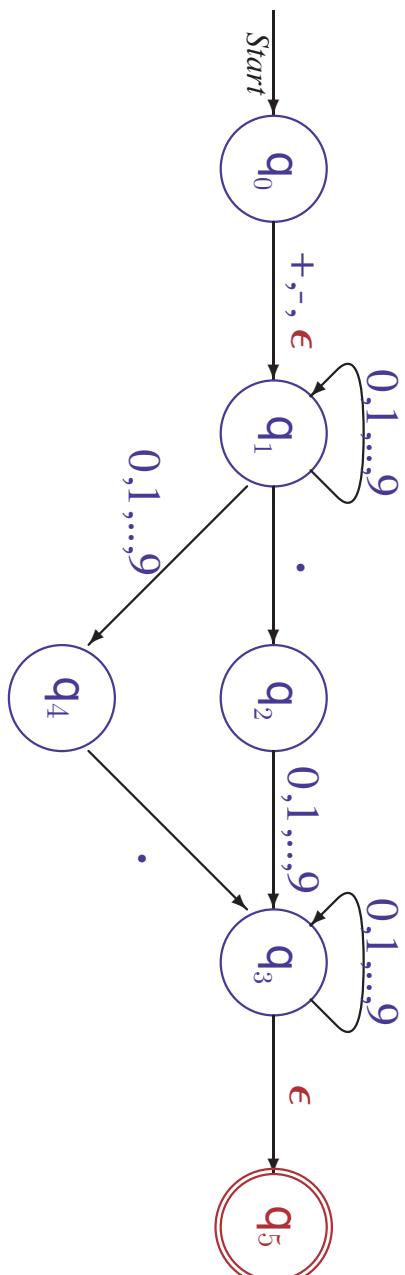
- (1) Jedes Teilwort kann in  $q_0$  bleiben
- (2) Ein Teilwort muss mit 0 enden, um nach  $q_1$  zu führen
- (3) Ein Teilwort muss mit 01 enden, um nach  $q_2$  zu führen
- (4) In  $q_2$  muss das Wort abgearbeitet sein

Beispiel: Abarbeitung von 00101



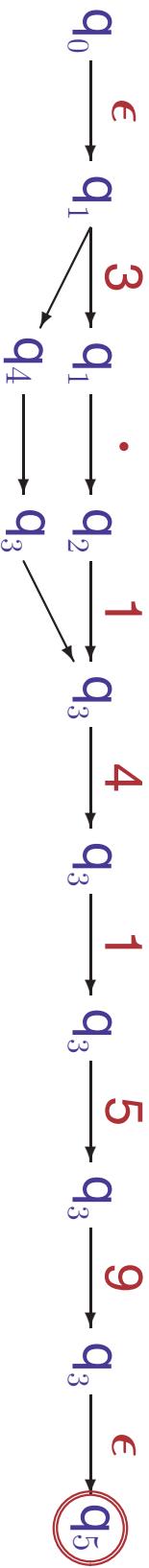
Ein Abarbeitungsweg führt zu einem akzeptierenden Zustand

# ARBEITSWEISE VON NEAS MIT $\epsilon$ -ÜBERGÄNGEN



- (1) Die Teilwörter  $+$ ,  $-$ , und  $\epsilon$  führen nach  $q_1$
- (2) Teilwörter der Form  $v\{0 \dots 9\}^+$  mit  $v \in \{+, -, \epsilon\}$  führen nach  $q_1$  oder  $q_4$
- (3) Teilwörter der Form  $v\{0 \dots 9\}^+ \cdot$  führen nach  $q_2$  oder  $q_3$
- (4) Teilwörter der Form  $v\{0 \dots 9\}^* \cdot \{0 \dots 9\}^+$  führen nach  $q_3$
- (5) Wörter die nach  $q_3$  führen, führen auch zum Endzustand  $q_5$

**Beispiel: Abarbeitung von 3.14159**

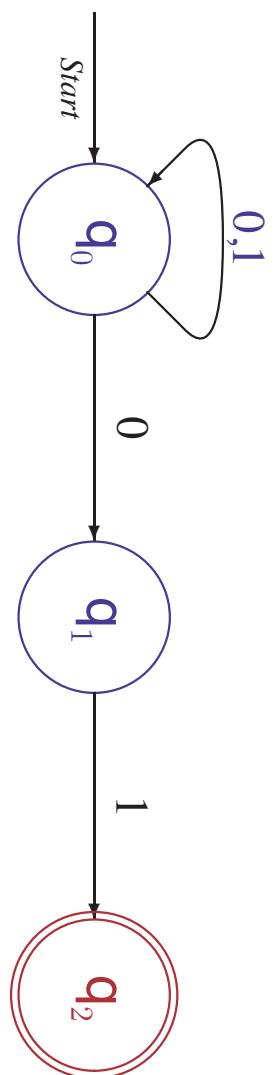


**Ein Abarbeitungsweg mit  $\epsilon$ -Übergängen führt zu einem Endzustand**

# ARBEITSWEISE VON $\epsilon$ -NEAS – PRÄZISIERT

- **Beschreibe  $\epsilon$ -Hülle eines Zustands  $q$** 
  - Die von  $q$  mit  $\epsilon$ -Übergängen erreichbaren Zustände
  - Iterative Definition: Kleinste Menge mit der Eigenschaft
$$q \in \epsilon\text{-Hülle}(q) \text{ und } p \in \epsilon\text{-Hülle}(q) \wedge r \in \delta(p, \epsilon) \Rightarrow r \in \epsilon\text{-Hülle}(q)$$
- **Erweiterte Überführungsfunktion  $\hat{\delta}: Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$** 
  - Aufsammeln aller bei der Abarbeitung erreichbaren Zustände einschließlich derjenigen, die ohne Eingabe erreicht werden
  - Induktive Definition (kaskadisches Aufsammeln von Zuständen)
- $$\hat{\delta}(q, w) = \begin{cases} \epsilon\text{-Hülle}(q) & \text{falls } w = \epsilon, \\ \bigcup_{q' \in \hat{\delta}(q, v)} \bigcup_{q'' \in \delta(q', a)} \epsilon\text{-Hülle}(q'') & \text{falls } w = v \text{ } a \text{ } (a \in \Sigma) \end{cases}$$
  - \* d.h.  $p \in \hat{\delta}(q, w)$  gdw. es gibt ein  $q' \in \hat{\delta}(q, v)$  und  $q'' \in \delta(q', a)$  so dass  $p \in \epsilon\text{-Hülle}(q'')$
- **Von A akzeptierte Sprache**
  - Menge der Eingaben  $w$ , für die  $\hat{\delta}(q_0, w)$  einen Endzustand enthält
- $L(A) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$

# ÜBERFÜHRUNGSFUNKTION $\hat{\delta}$ (OHNE $\epsilon$ -ÜBERGÄNGE)

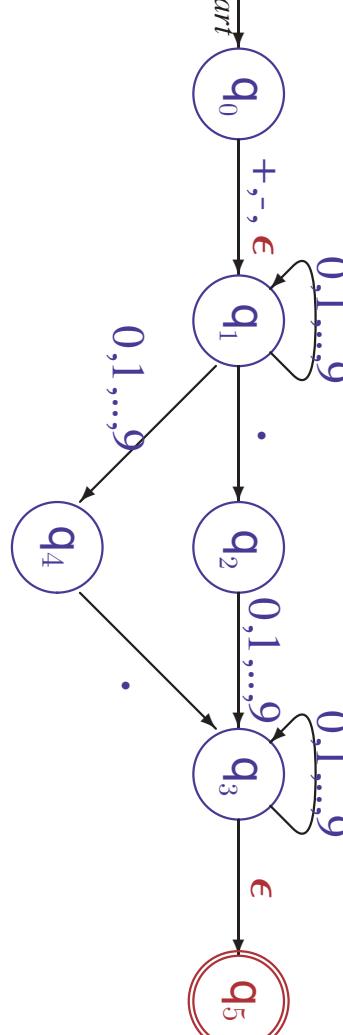


- Abarbeitung von 00101

- $-\hat{\delta}(q_0, \epsilon) = \{q_0\}$
- $-\hat{\delta}(q_0, 0) = \delta(q_0, 0) = \{q_0, q_1\}$
- $-\hat{\delta}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$
- $-\hat{\delta}(q_0, 001) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$
- $-\hat{\delta}(q_0, 0010) = \delta(q_0, 0) \cup \delta(q_2, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$
- $-\hat{\delta}(q_0, 00101) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$
- 00101 wird akzeptiert da  $\hat{\delta}(q_0, 00101) \cap F = \{q_2\}$

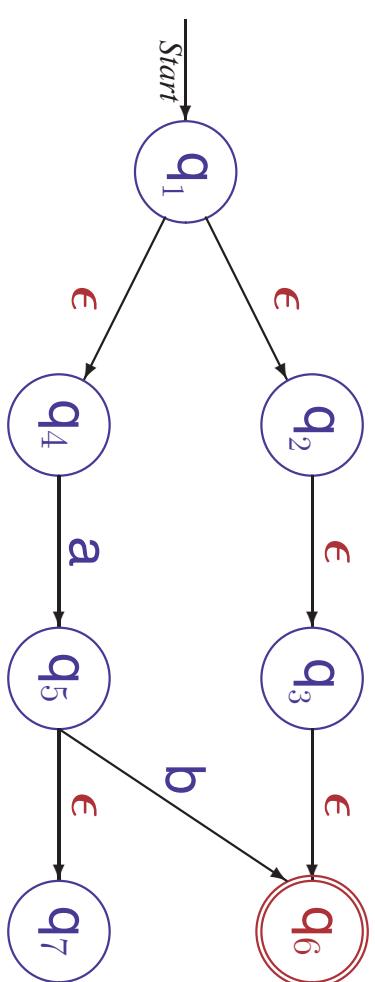
## BESTIMMUNG DER $\epsilon$ -HÜLLE

- Dezimalautomat

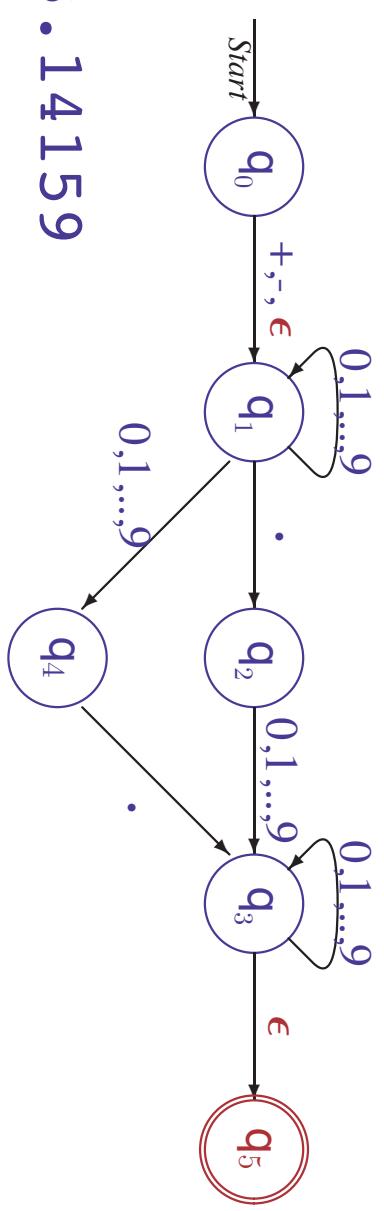
- Nur zwei  $\epsilon$ -Übergänge
- $\epsilon$ -Hülle( $q_0$ ) = { $q_0, q_1$ } 
- $\epsilon$ -Hülle( $q_3$ ) = { $q_3, q_5$ }
- $\epsilon$ -Hülle( $q_i$ ) = { $q_i$ } sonst

- Viele  $\epsilon$ -Übergänge

- $\epsilon$ -Hülle( $q_1$ ) = { $q_1, q_2, q_3, q_4, q_6$ }
- $\epsilon$ -Hülle( $q_2$ ) = { $q_2, q_3, q_6$ }
- $\epsilon$ -Hülle( $q_3$ ) = { $q_3, q_6$ }
- $\epsilon$ -Hülle( $q_4$ ) = { $q_4$ }
- $\epsilon$ -Hülle( $q_5$ ) = { $q_5, q_7$ }
- $\epsilon$ -Hülle( $q_6$ ) = { $q_6$ }
- $\epsilon$ -Hülle( $q_7$ ) = { $q_7$ }



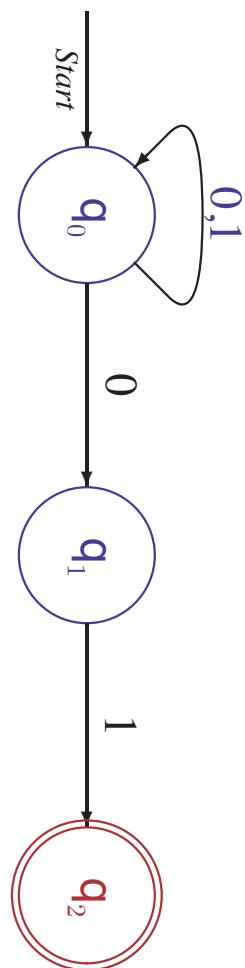
# ÜBERFÜHRUNGSFUNKTION $\hat{\delta}$ (MIT $\epsilon$ -ÜBERGÄNGEN)



## Abarbeitung von 3. • 14 159

- $\hat{\delta}(q_0, \epsilon) = \epsilon\text{-H\"ulle}(q_0) = \{q_0, q_1\}$
- $\hat{\delta}(q_0, 3) : \delta(q_0, 3) \cup \delta(q_1, 3) = \emptyset \cup \{q_1, q_4\} = \{q_1, q_4\}$
- $\hat{\delta}(q_0, 3) = \epsilon\text{-H\"ulle}(q_1) \cup \epsilon\text{-H\"ulle}(q_4) = \{q_1\} \cup \{q_4\} = \{q_1, q_4\}$
- $\hat{\delta}(q_0, 3.) : \delta(q_1, .) \cup \delta(q_4, .) = \{q_2\} \cup \{q_3\} = \{q_2, q_3\}$
- $\hat{\delta}(q_0, 3.) = \epsilon\text{-H\"ulle}(q_2) \cup \epsilon\text{-H\"ulle}(q_3) = \{q_2\} \cup \{q_3, q_5\} = \{q_2, q_3, q_5\}$
- $\hat{\delta}(q_0, 3.1) : \delta(q_2, 1) \cup \delta(q_3, 1) \cup \delta(q_5, 1) = \{q_3\} \cup \{q_3\} \cup \emptyset = \{q_3\}$
- $\hat{\delta}(q_0, 3.14) = \epsilon\text{-H\"ulle}(q_3) = \{q_3, q_5\}$
- $\hat{\delta}(q_0, 3.14159) = \epsilon\text{-H\"ulle}(q_3) = \{q_3, q_5\}$

## NACHWEIS DER ERKANNTEN SPRACHE (OHNE $\epsilon$ -ÜBERGÄNGE)



$$L(A) = \{w \in \{0, 1\}^* \mid w \text{ endet mit } 01\}$$

- Zeige durch simultane Induktion für alle  $w \in \{0, 1\}^*$

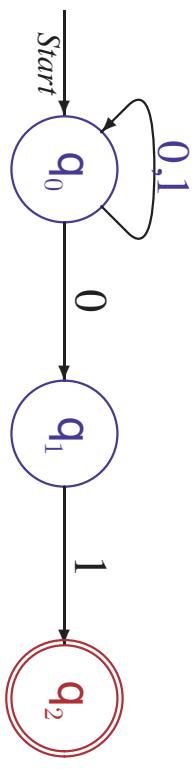
- $q_0 \in \hat{\delta}(q_0, w)$
- $q_1 \in \hat{\delta}(q_0, w)$  genau dann, wenn  $w$  mit 0 endet
- $q_2 \in \hat{\delta}(q_0, w)$  genau dann, wenn  $w$  mit 01 endet

Dann folgt  $w \in L(A) \Leftrightarrow \hat{\delta}(q_0, w) \cap \{q_2\} \neq \emptyset \Leftrightarrow w \text{ endet mit } 01$

- Induktionsanfang  $w = \epsilon$

- Per Definition ist  $\hat{\delta}(q_0, \epsilon) = \{q_0\}$ . Also gilt Aussage a)
- $w$  endet weder mit 0 noch mit 01. Aussagen b) und c) gelten trivialerweise

## NACHWEIS DER ERKANNTEN SPRACHE II



- a)  $q_0 \in \hat{\delta}(q_0, w)$
- b)  $q_1 \in \hat{\delta}(q_0, w)$  genau dann, wenn  $w$  mit 0 endet
- c)  $q_2 \in \hat{\delta}(q_0, w)$  genau dann, wenn  $w$  mit 01 endet

### • Induktionssschritt: $w = va$ für $v \in \{0, 1\}^*$ , $a \in \{0, 1\}$

– Die Aussagen a), b), und c) seien für  $v$  gültig

- a) Wegen  $q_0 \in \hat{\delta}(q_0, v)$  und  $q_0 \in \delta(q_0, a)$  für  $a \in \{0, 1\}$  folgt  $q_0 \in \hat{\delta}(q_0, w)$
- b) Sei  $q_1 \in \hat{\delta}(q_0, w)$ . Wegen  $q_1 \in \delta(q, a) \Leftrightarrow q=q_0 \wedge a=0$  muss  $w$  mit 0 enden

Wenn umgekehrt  $w$  mit 0 endet, dann ist  $a=0$ .

Wegen  $q_0 \in \hat{\delta}(q_0, v)$  und  $q_1 \in \delta(q_0, a)$  folgt  $q_1 \in \hat{\delta}(q_0, w)$

- c) Sei  $q_2 \in \hat{\delta}(q_0, w)$ . Wegen  $q_2 \in \delta(q, a) \Leftrightarrow q=q_1 \wedge a=1$  muss  $w$  mit 1 enden und  $q_1 \in \hat{\delta}(q_0, v)$  gelten. Wegen b) für  $v$  endet  $w$  mit 0, also  $w$  mit 01

Wenn umgekehrt  $w$  mit 01 endet, dann ist  $a=1$  und  $v$  endet mit 0.

Wegen  $q_1 \in \hat{\delta}(q_0, v)$  nach b) und  $q_2 \in \delta(q_1, a)$  folgt  $q_2 \in \hat{\delta}(q_0, w)$

# ARBEITSWEISE VON $\epsilon$ -NEAs – ALTERNATIVE BESCHREIBUNG MIT KONFIGURATIONSÜBERGÄNGEN

- **Definiere Konfigurationen**

- Formal dargestellt als Tupel  $\textcolor{red}{K} = (q, w) \in Q \times \Sigma^*$

- **Definiere Konfigurationsübergangsrelation**  $\vdash^*$

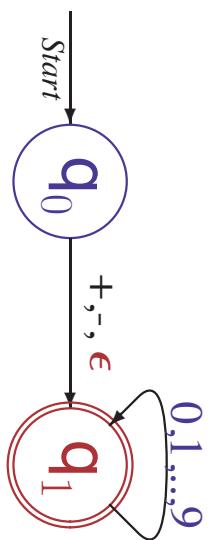
- Wechsel zwischen Konfigurationen durch Abarbeitung von Wörtern
  - $(q, aw) \vdash (p, w)$ , falls  $p \in \delta(q, a)$
  - $(q, w) \vdash (p, w)$ , falls  $p \in \delta(q, \epsilon)$
  - $K_1 \vdash^* K_2$ , falls  $K_1 = K_2$  oder
    - es gibt eine Konfiguration  $K$  mit  $K_1 \vdash K$  und  $K \vdash^* K_2$

- **Akzeptierte Sprache**

- Menge der Eingaben, für die  $\vdash^*$  zu akzeptierenden Zustand führt

$$L(A) = \{w \in \Sigma^* \mid \exists p \in F. (q_0, w) \vdash^* (p, \epsilon)\}$$

# NACHWEIS DER ERKANNTEN SPRACHE (MIT $\epsilon$ -ÜBERGÄNGEN)



$$L(A) = \{w \in \Sigma^* \mid \exists u \in \{0, \dots, 9\}^*. w = u \vee w = +u \vee w = -u\}$$

• Zeige  $(q_1, w v) \vdash^* (q_1, v) \Leftrightarrow w \in \{0, \dots, 9\}^*$  für alle  $w, v \in \Sigma^*$

• Basisfall  $w = \epsilon$ :

– Per Definition gilt  $(q_1, v) \vdash^* (q_1, v)$  und  $\epsilon \in \{0, \dots, 9\}^*$

• Schrittfall  $w = ua$  für ein  $u \in \Sigma^*, a \in \Sigma$ :

$\Rightarrow : \text{Es gelte } (q_1, w v) \vdash^* (q_1, v).$

Dann gilt  $(q_1, ua v) \vdash^* (p, av) \vdash (q_1, v)$  für einen Zustand  $p$ .

Es folgt  $p = q_1$ ,  $a \in \{0, \dots, 9\}$  und per Induktion  $w \in \{0, \dots, 9\}^*$

$\Leftarrow : \text{Es sei } w \in \{0, \dots, 9\}^*. \text{ Dann ist } u \in \{0, \dots, 9\}^* \text{ und } a \in \{0, \dots, 9\}.$

Mit der Induktionsannahme folgt  $(q_1, ua v) \vdash^* (q_1, av) \vdash (q_1, v) \quad \checkmark$

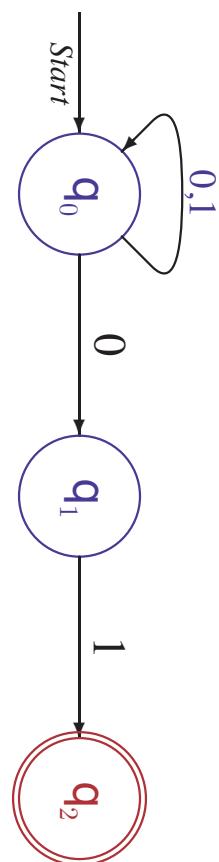
• Es folgt

$$\begin{aligned} w \in L(A) &\Leftrightarrow (q_0, w) \vdash^* (q_1, \epsilon) \\ &\Leftrightarrow \exists u \in \Sigma^*. w \in \{u, +u, -u\}. (q_0, w) \vdash (q_1, u) \vdash^* (q_1, \epsilon) \\ &\Leftrightarrow \exists u \in \{0, \dots, 9\}^*. w = u \vee w = +u \vee w = -u \Leftrightarrow w \in L \end{aligned}$$

## BEZIEHUNG ZU DETERMINISTISCHEN AUTOMATEN

- **Nichtdeterministische Automaten sind flexibler**
  - Man muss sich nicht auf eine genaue Verarbeitungsfolge festlegen
  - Man kann optionale Eingaben elegant verarbeiten
- **DEAs sind genauso ausdrucksstark wie  $\epsilon$ -NEAs**
  - Man kann Mengen von  $\epsilon$ -NEA-Zuständen als DEA Zustände codieren
  - Man kann mengenwertige Zustandsübergangsfunktionen codieren
- **(Potenzmengen- oder) Teilmengenkonstruktion**
  - Sei  $A_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$  ein nichtdeterministischer Automat
  - Konstruiere äquivalenten DEA  $A_D = (Q_D, \Sigma, \delta_D, q_D, F_D)$  mit
    - $Q_D = \mathcal{P}(Q_N)$
    - $q_D = \epsilon\text{-Hülle}(q_0)$   
 $(= \{q_0\} \text{ bei NEAs})$
    - $F_D = \{S \in Q_D \mid S \cap F_N \neq \emptyset\}$
    - $\hat{\delta}_D(S, a) = \bigcup_{q \in S} \hat{\delta}_N(q, a) = \{p \mid \exists q \in S. p \in \hat{\delta}_N(q, a)\}$  (erfaßt  $\epsilon$ -Hülle)
  - Dann gilt  $L(A_D) = L(A_N)$
  - Konstruktion benötigt  $2^{|Q_N|}$  Zustände  
(Optimierung möglich)

# TEILMENGENKONSTRUKTION AM BEISPIEL



## Konstruiertter deterministischer Automat

$Q_D$	$\mathcal{P}(\{q_0, q_1, q_2\})$		
$q_D$	$\epsilon$ -Hülle( $q_0$ ) = $\{q_0\}$	$0$	$1$
$F_D$	$\{S \subseteq \{q_0, q_1, q_2\} \mid q_2 \in S\}$	$\rightarrow \{q_0\}$	
		$\{q_0, q_1\}$	$\{q_0\}$
		$\{q_1\}$	$\emptyset$
		$\{q_2\}$	$\{q_2\}$
		$\emptyset$	$\emptyset$
		$\{q_0, q_1\}$	$\{q_0, q_2\}$
	*	$\{q_0, q_2\}$	$\{q_0\}$
	*	$\{q_1, q_2\}$	$\emptyset$
	*	$\{q_2\}$	$\{q_2\}$
	$^*\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

Viele überflüssige Zustände (nur drei von  $\{q_0\}$  erreichbar)

- **Optimierung:**  $Q_D \stackrel{\hat{\wedge}}{=} \text{erreichbare Zustände}$

- Sei  $A_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$  ein nichtdeterministischer Automat
- Konstruiere Zustandsmenge  $Q_D$  iterativ gleichzeitig mit  $\delta_D$

Start:  $Q_0 := \{q_D\} = \{\epsilon\text{-Hülle}(q_0)\}$

Schritt:  $Q_{i+1} := Q_i \cup \{\delta_D(S, a) \mid S \in Q_i, a \in \Sigma\}$

Dabei konstruiere die nötigen Werte  $\delta_D(S, a) = \bigcup_{q \in S} \hat{\delta}_N(q, a)$

Abschluss: Wenn  $Q_{i+1} = Q_i$ , dann halte an und setze  $Q_D := Q_i$

- Setze  $q_D = \epsilon\text{-Hülle}(q_0)$  und  $F_D = \{S \in Q_D \mid S \cap F_N \neq \emptyset\}$
- DEA  $A_D = (Q_D, \Sigma, \delta_D, q_D, F_D)$  enthält keine überflüssigen Zustände

- **$\epsilon$ -NEAs und DEAs akzeptieren dieselben Sprachen**

- Jeder DEA ist als “eindeutiger”  $\epsilon$ -NEA beschreibbar

## OPTIMIERTE TEILMENGENKONSTRUKTION: KORREKTHEIT

Für den konstruierten DEA gilt  $L(A_D) = L(A_N)$

Zeige:  $\hat{\delta}_D(q_D, w) = \hat{\delta}_N(q_0, w)$  für alle  $w \in \Sigma^*$

Beweis durch strukturelle Induktion über den Aufbau der Wörter aus  $\Sigma^*$

– Basisfall: Sei  $w = \epsilon$ :

$$\hat{\delta}_D(q_D, \epsilon) = q_D = \epsilon\text{-Hülle}(q_0) = \hat{\delta}_N(q_0, \epsilon)$$

– Induktionssschritt: Sei  $w = va$  für ein  $v \in \Sigma^*$  und  $a \in \Sigma$ :

– Induktionsannahme: Es gelte  $\hat{\delta}_D(q_D, v) = \hat{\delta}_N(q_0, v)$

Dann gilt  $\hat{\delta}_D(q_D, w)$

$$= \hat{\delta}_D(\hat{\delta}_D(q_D, v), a)$$

$$= \hat{\delta}_D(\hat{\delta}_N(q_0, v), a)$$

$$= \hat{\delta}_N(\hat{\delta}_N(q_0, v), a)$$

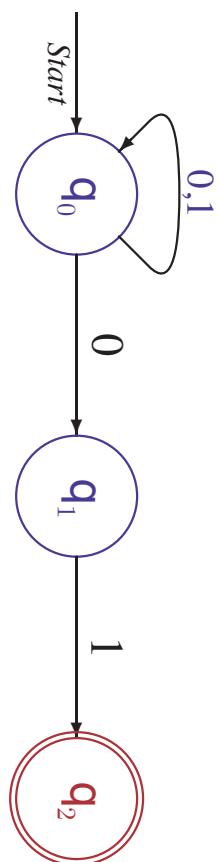
$$= \bigcup_{q' \in \hat{\delta}_N(q_0, v)} \hat{\delta}_N(q', a)$$

$$= \bigcup_{q' \in \hat{\delta}_N(q_0, v)} \bigcup_{q'' \in \delta_N(q', a)} \epsilon\text{-Hülle}(q'')$$

$$= \hat{\delta}_N(q_0, w)$$

Es folgt  $L(A_D) = \{w \mid \hat{\delta}_D(q_D, w) \in F_D\} = \{w \mid \hat{\delta}_N(q_0, w) \cap F_N \neq \emptyset\} = L(A_N)$

# OPTIMIERTE TEILMENGENKONSTRUKTION FÜR NEAS

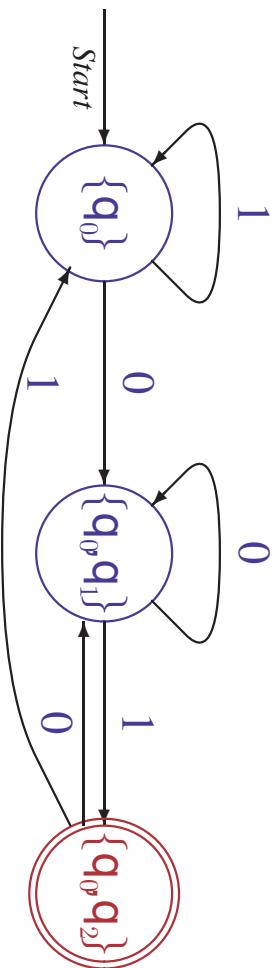


- Konstruktion von Zustandsmengen und (reduzierter) Überführungsfunktion

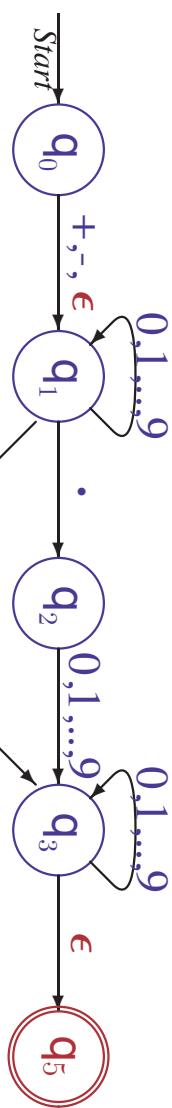
	0	1	
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$	$Q_0 := \{\{q_0\}\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$	$Q_1 := \{\{q_0\}, \{q_0, q_1\}\}$
$* \{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$	$Q_2 := \{\{q_0\}, \{q_0, q_1\}, \{q_0, q_2\}\}$

$Q_3 = Q_2$ , also  $Q_D = Q_2$ ,

- Resultierender deterministischer Automat



# OPTIMIERTE TEILMENGENKONSTRUKTION FÜR $\epsilon$ -NEAs



- Konstruiere  $Q_D$  und  $\delta_D$

$$Q_0 = \{q_D\} = \{\epsilon\text{-H\"ulle}(q_0)\} = \{\{q_0, q_1\}\}$$

$$-\delta_D(\{q_0, q_1\}, +) = \{q_1\}, \quad \delta_D(\{q_0, q_1\}, -) = \{q_1\}$$

$$-\delta_D(\{q_0, q_1\}, \mathbf{0}) = \{q_1, q_4\}, \dots \delta_D(\{q_0, q_1\}, \mathbf{9}) = \{q_1, q_4\}, \quad \delta_D(\{q_0, q_1\}, \cdot) = \{q_2\}$$

$$Q_1 = \{ \{q_0, q_1\} \{q_1\}, \{q_1, q_4\}, \{q_2\} \}$$

$$-\delta_D(\{q_1\}, +) = \delta_D(\{q_2\}, +) = \delta_D(\{q_1, q_4\}, +) = \emptyset, \dots$$

$$-\delta_D(\{q_1\}, \mathbf{0}) = \delta_D(\{q_1, q_4\}, \mathbf{0}) = \{q_1, q_4\} \quad \delta_D(\{q_2\}, \mathbf{0}) = \{q_3, q_5\}, \dots$$

$$-\delta_D(\{q_1\}, \cdot) = \{q_2\}, \quad \delta_D(\{q_2\}, \cdot) = \emptyset \quad \delta_D(\{q_1, q_4\}, \cdot) = \{q_2, q_3, q_5\}$$

$$Q_2 = \{ \{q_0, q_1\} \{q_1\}, \{q_1, q_4\}, \{q_2\}, \emptyset, \{q_3, q_5\}, \{q_2, q_3, q_5\} \}$$

$$-\delta_D(\emptyset, +) = \delta_D(\{q_2, q_3, q_5\}, +) = \delta_D(\{q_3, q_5\}, +) = \emptyset, \dots$$

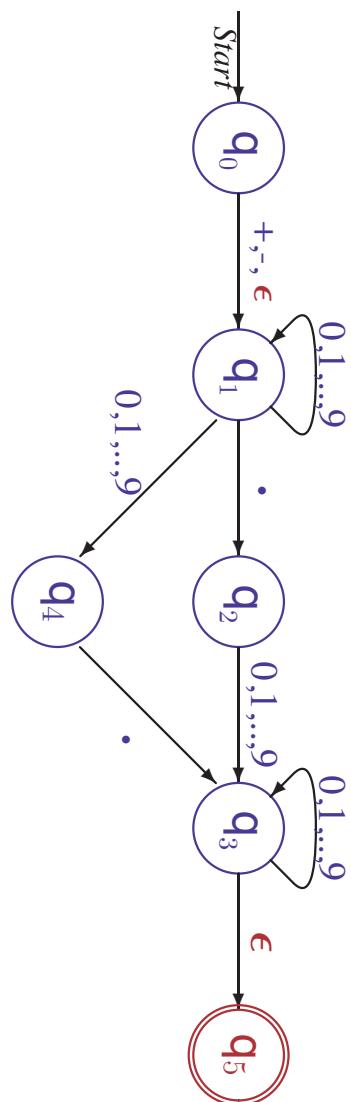
$$-\delta_D(\emptyset, \mathbf{0}) = \emptyset, \quad \delta_D(\{q_2, q_3, q_5\}, \mathbf{0}) = \delta_D(\{q_3, q_5\}, \mathbf{0}) = \{q_3, q_5\}, \dots$$

$$-\delta_D(\emptyset, \cdot) = \delta_D(\{q_2, q_3, q_5\}, \cdot) = \delta_D(\{q_3, q_5\}, \cdot) = \emptyset$$

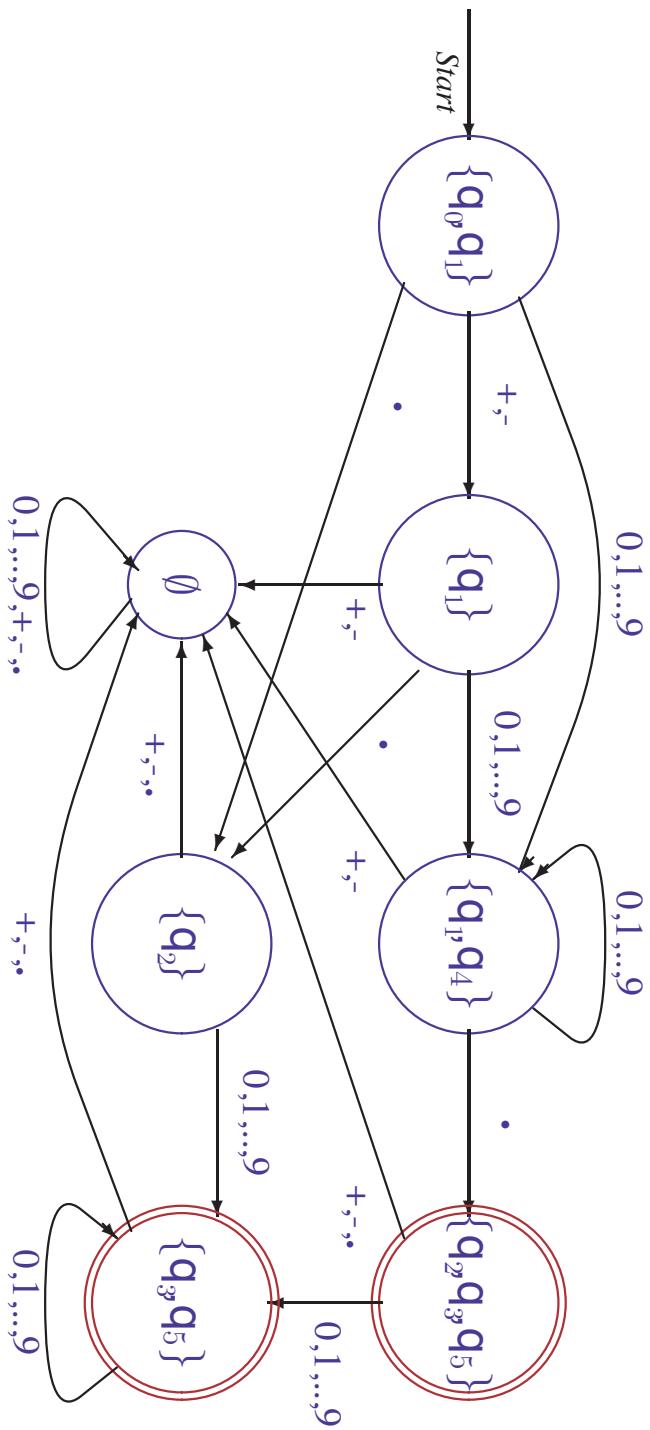
$$Q_3 = \{ \{q_0, q_1\} \{q_1\}, \{q_1, q_4\}, \{q_2\}, \emptyset, \{q_3, q_5\}, \{q_2, q_3, q_5\} \} = Q_2 =: Q_D$$

# ERZUGTER DEA FÜR DEZIMALZAHLERKENNUNG

## Ursprünglicher $\epsilon$ -NEA

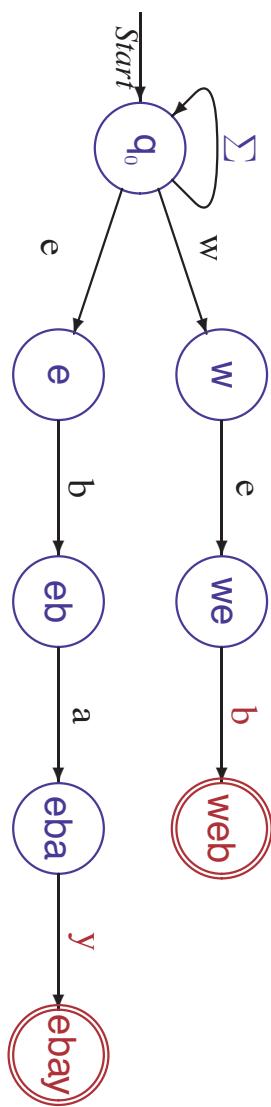


## Generierter DEA

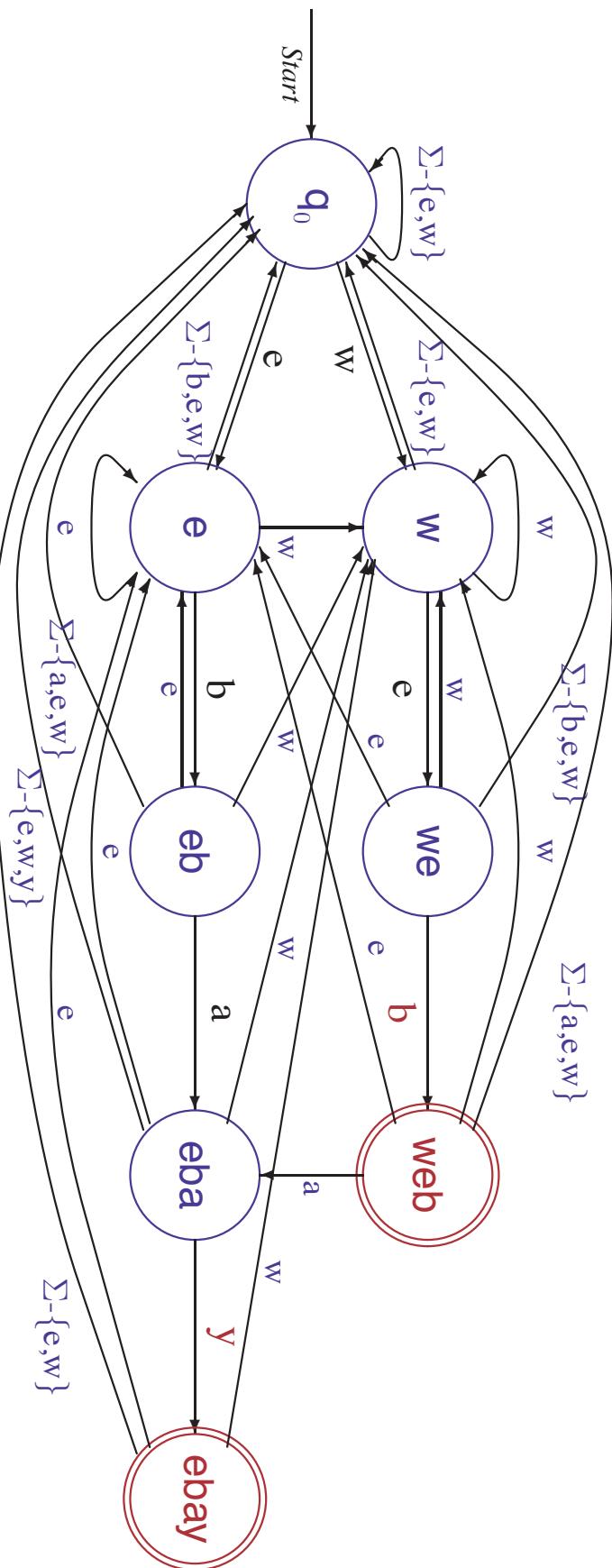


# DETERMINISTISCHE AUTOMATEN FÜR TEXTANALYSE

## Ursprünglicher $\epsilon$ -NEA



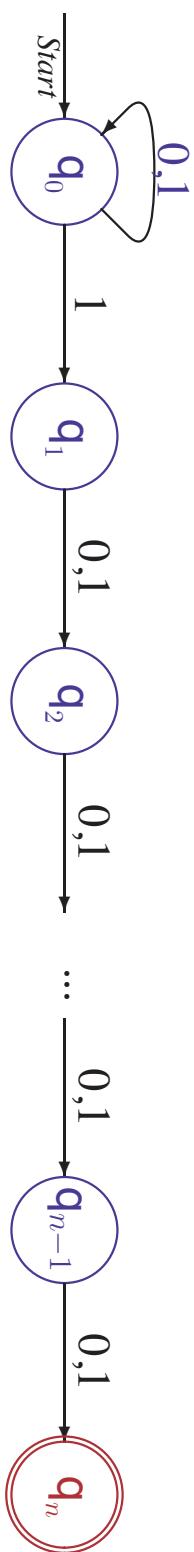
## Generierter DFA



## ANALYSE DER OPTIMIERTEN TEILMENGENKONSTRUKTION

- $A_D$  kann so klein sein wie  $A_N$ 
  - Nur wenige Teilmengen von  $Q_N$  werden wirklich erreicht

- $A_D$  kann exponentiell größer werden



–  $L(A_N) = \{w \in \{0, 1\}^* \mid \text{das } n\text{-te Zeichen vor dem Ende ist eine } 1\}$

– Jeder DEA  $A$  für  $L(A_N)$  benötigt mindestens  $2^n$  Zustände

– Beweis: Es gibt  $2^n$  Wörter der Länge  $n$  in  $\{0, 1\}^*$

Hat  $A$  weniger als  $2^n$  Zustände, so gibt es  $w = a_1..a_n$  und  $v = b_1..b_n$

mit  $w \neq v$  und  $\hat{\delta}_A(q_0, w) = \hat{\delta}_A(q_0, v)$

(**Schubfachprinzip**)

Sei  $a_i \neq b_i$ . Für  $q = \delta_A(q_0, w0^{i-1}) = \delta_A(q_0, v0^{i-1})$  folgt  $q \in F$  und  $q \notin F$

# ENDLICHE AUTOMATEN – ZUSAMMENFASSUNG

- **Deterministische Endliche Automaten (DEA)**

- Endliche Menge von **Zuständen**, endliche Menge von Eingabesymbolen
- Ein fester Startzustand, null oder mehr akzeptierende Zustände
- Überführungsfunktion bestimmt Änderung des Zustands bei Abarbeitung der Eingabe

- Erkannte Sprache: Eingaben, deren Abarbeitung in einem akzeptierenden Zustand endet

- **Automaten mit Ausgabe (Mealy/Moore-Automat)**

- Wie DEA, mit zusätzlicher Ausgabefunktion
- Gegenseitige Simulation möglich

- **Nichtdeterministische Automaten ( $\epsilon$ -NEA / NEA)**

- Wie DEA, aber mit mengenwertiger Überführungsfunktion und Zustandsüberführung bei leerer Eingabe
- Durch Teilmengenkonstruktion in äquivalenten DEA transformierbar

# Theoretische Informatik I

## Einheit 2.3

### Reguläre Ausdrücke



1. Anwendungen
2. Syntax und Semantik
3. Vereinfachungsregeln
4. Beziehung zu endlichen Automaten

# EINE ALGEBRAISCHE BESCHREIBUNG FÜR SPRACHEN

- Automaten beschreiben Abarbeitung von Sprachen
  - Operationale Semantik: Symbole führen zu Zustandsänderungen
  - Bestimmte Wörter bzw. Symbolketten werden durch Zustände akzeptiert
  - Für Automaten ist Sprache  $\hat{=}$  Menge der akzeptierten Wörter
- Wie beschreibt man Eigenschaften von Wörtern?
  - Deklarative Semantik: äußere Form von Zeichenreihen einer Sprache
    - z.B. *Wörter haben eine führende Null, dann beliebig viele Einsen*
    - Anwendungen brauchen präzise Beschreibungssprache für Wörter
      - Grundeinheiten von Programmiersprachen, Suchmuster für Browser, ...
- Reguläre Ausdrücke als formale Syntax
  - Kurze, prägnante Beschreibung des Aufbaus der Wörter einer Sprache
  - z.B. 01\*: “Zuerst eine Null, dann beliebig viele Einsen”

## ANWENDUNG: TEXTSUCHE

- **Suche nach Mustern in Texten**
  - Suche ob/wo/wie oft eine bestimmte Zeichenkette im Text erscheint
  - Textmuster kann Platzhalter enthalten
- **Beschreibe Textmuster durch reguläre Ausdrücke**
  - Zahl: Ziffernfolge dann evtl. Punkt und nichtleere Ziffernfolge
  - Formaler Ausdruck:
$$(0+1+\dots+9)^*(\epsilon+(.\,(0+1+\dots+9)\,(0+1+\dots+9)^*))$$
- **Vielfältige Anwendungen**
  - Google Suche nach einfachen Texten
  - Erweiterte Google Suche nach Textmustern
  - Unix Kommando **grep**: suche nach Textmustern in Dateien
  - Programmiersprachen wie **PERL** und **AWK**
  - Textsuche und Textersetzung in **Emacs**
  - Lexikalische Analyse in Compilern

# ANWENDUNG: LEXIKALISCHE ANALYSE

## Wichtigster Grundbestandteil von Compilern

- **Reguläre Ausdrücke beschreiben Token**

- Logische Grundeinheiten von Programmiersprachen
- z.B. Schlüsselwörter, Bezeichner, Dezimalzahlen, ...

- **“Lexer” transformieren reguläre Ausdrücke in Analyseprogramme**

- Analyse kann die Token der Programmiersprache identifizieren
- Zugrundeliegende Technik:

Umwandlung regulärer Ausdrücke in DEAs

# REGULÄRE AUSDRÜCKE PRÄZISIERT (SYNTAX)

- **Syntax:** Terme über  $\Sigma \cup \{\emptyset, \epsilon, +, \circ, *, (\ ), )\}$

Reguläre Ausdrücke sind induktiv wie folgt definiert

- $E = a$  ist ein regulärer Ausdruck für jedes  $a \in \Sigma$
- $E = \emptyset$  und  $F = \epsilon$  sind reguläre Ausdrücke
- Sind  $E$  und  $F$  reguläre Ausdrücke, dann sind auch
  - $E \circ F$ ,  $E^*$ ,  $E+F$  und  $(E)$  sind reguläre Ausdrücke
  - Mehr Ausdrücke möglich, aber nicht erforderlich
- **Konventionen zur Vereinfachung**
  - $E \circ F$  wird üblicherweise als  $EF$  abgekürzt
  - Definitorische Abkürzungen:  $E^+ \equiv EE^*$ ,  $[a_1 \dots a_n] \equiv a_1 + \dots + a_n$
  - Prioritätsregelungen ermöglichen, überflüssige Klammern wegzulassen
    - $*$  (“Sternoperator”) bindet stärker als  $\circ$ , und dies stärker als  $+$
    - Verkettung  $\circ$  und Alternative  $+$  sind assoziativ

# REGULÄRE AUSDRÜCKE PRÄZISIERT (SEMANTIK)

- Reguläre Ausdrücke beschreiben Sprachen über  $\Sigma$

- Die Sprache  $L(E)$  ist induktiv definiert

- Für für alle  $a \in \Sigma$  ist  $L(a) = \{a\}$  (einelementige Sprache, die nur  $a$  enthält)
- $L(\emptyset)$  ist die leere Sprache (üblicherweise geschrieben als  $\emptyset$  oder  $\{\}$ )
- $L(\epsilon) = \{\epsilon\}$  (einelementige Sprache, die nur das leere Wort enthält)
- $L(E \circ F) = L(E) \circ L(F) = \{vw \mid v \in L(E) \wedge w \in L(F)\}$ 
  - steht für die Verkettung (der Wörter) zweier Sprachen
- $L(E^*) = (L(E))^* = \{w_1 w_2 \dots w_n \mid n \in \mathbb{N} \wedge w_i \in L(E)\}$ 
  - \* steht für Verkettung beliebig vieler Wörter einer Sprache (Kleene'sche Hülle)
- $L(E+F) = L(E) \cup L(F) = \{w \in \Sigma^* \mid w \in L(E) \vee w \in L(F)\}$ 
  - + steht für die Vereinigung zweier Sprachen
- $L((E)) = L(E)$

# SPRACHEN VS. AUSDRÜCKE

- **Sprachen sind Mengen von Wörtern**

- Abstraktes semantisches Konzept: Ungeordnete Kollektion von Wörtern
- Beschreibung von Mengen (auf Folie, Tafel,...) benötigt textuelle Notation
- Notation benutzt Kurzschreibweisen wie  $\cup$ ,  $\circ$ ,  $*$  für Mengenoperationen
- ... aber ist selbst nur ein Hilfsmittel zur Kommunikation

- **Reguläre Ausdrücke sind Terme**

- Eine syntaktische Beschreibungsform, die ein Computer versteht
- Reguläre Ausdrücke werden zur Beschreibung von Sprachen benutzt und sind ähnlich zur Standardnotation von Mengen

- **Reguläre Ausdrücke sind selbst keine Sprachen**

- Unterscheide Ausdruck  $E$  von Sprache des Ausdrucks  $L(E)$
- Man verzichtet auf den Unterschied wenn der Kontext eindeutig ist

## BEISPIELE REGULÄRER AUSDRÜCKE

- $\mathbf{a}^* \mathbf{b} \mathbf{a}^*$ 
  - steht für die Menge aller Wörter über  $\Sigma = \{\mathbf{a}, \mathbf{b}\}$ , die genau ein  $b$  enthalten
  - $L(\mathbf{a}^* \mathbf{b} \mathbf{a}^*) = \{w \in \Sigma^* \mid w \text{ enthält genau ein } b\} = \{w \in \Sigma^* \mid \#_b(w) = 1\}$
- $\Sigma^* \mathbf{b} \Sigma^*$ 
  - steht für  $\{w \in \Sigma^* \mid w \text{ enthält mindestens ein } b\} = \{w \in \Sigma^* \mid \#_b(w) \geq 1\}$
- $\mathbf{a}^* (\mathbf{b} + \epsilon) \mathbf{a}^*$ 
  - steht für  $\{w \in \Sigma^* \mid w \text{ enthält maximal ein } b\} = \{w \in \Sigma^* \mid \#_b(w) \leq 1\}$
- $\mathbf{a} \emptyset$ 
  - steht für die leere Sprache, denn die Verkettung einer Sprache mit der leeren Sprache ist immer leer
- $\emptyset^*$ 
  - steht für die Menge  $\{\epsilon\}$ , denn die beliebige Verkettung von Wörtern einer Menge enthält immer das leere Wort

# ENTWICKLUNG REGULÄRER AUSDRÜCKE

Beschreibe Menge aller Wörter, in denen 0 und 1 abwechseln

## 1. Regulärer Ausdruck für die Sprache {01}

- **0** repräsentiert  $\{0\}$ , **1** repräsentiert  $\{1\}$
- Also ist  $L(\textcolor{red}{0}1) = L(\textcolor{red}{0}) \circ L(\textcolor{red}{1}) = \{0\} \circ \{1\} = \{01\}$

## 2. Erzeuge $\{01, 0101, 010101, \dots\}$ durch Sternbildung

$$- L((\textcolor{red}{0}1)^*) = L(\textcolor{red}{0}1)^* = \{01\}^* = \{\epsilon, 01, 0101, 010101, \dots\}$$

## 3. Manche Wörter nicht erfaßt

- Start mit Eins statt Null:
- Start und Ende mit Null:
- Start und Ende mit Eins:
  - Vollständiger Ausdruck:

$$\begin{aligned} & (\textcolor{red}{1}0)^* \\ & (\textcolor{red}{0}1)^* 0 \\ & (\textcolor{red}{1}0)^* 1 \end{aligned}$$

## 4. Es geht auch kürzer

- Optional **1** am Anfang oder **0** am Ende:

$$(\epsilon + 1)(01)^*(\epsilon + 0)$$

# BESTIMMUNG DER SEMANTIK VON $(\epsilon+1)(01)^*(\epsilon+0)$

$$\begin{aligned} & L((\epsilon+1)(01)^*(\epsilon+0)) \\ &= L((\epsilon+1)) \circ L((01)^*) \circ L((\epsilon+0)) \\ &= L(\epsilon) \cup L(1) \circ L((01))^* \circ L(\epsilon) \cup L(0) \\ &= \{\epsilon\} \cup \{1\} \circ (L(0) \circ L(1))^* \circ \{\epsilon\} \cup \{0\} \\ &= \{\epsilon, 1\} \circ \{01\}^* \circ \{\epsilon, 0\} \\ &= \{\epsilon, 1\} \circ \{w \mid \exists n \in \mathbb{N}. w = \underbrace{01\dots01}_{n-mal} \} \circ \{\epsilon, 0\} \\ &= \{w \mid \exists n \in \mathbb{N}. w = \underbrace{01\dots01}_{n-mal} \vee w = 1 \underbrace{01\dots01}_{n-mal} \} \\ &\quad \vee w = \underbrace{01\dots01}_{n-mal} 0 \vee w = 1 \underbrace{01\dots01}_{n-mal} 0 \} \end{aligned}$$

= **Die Menge aller Wörter**, in denen 0 und 1 abwechseln  
(Mühsamer Beweis durch Induktion)

# “RECHENREGELN” FÜR REGULÄRE AUSDRÜCKE

Wie zeigt man  $(01)^* + (10)^* + (01)^* 0 + (10)^* 1 \cong (\epsilon+1)(01)^*(\epsilon+0)$  ?

- Definiere Äquivalenz von Ausdrücken

- $E \cong F$ , falls  $L(E) = L(F)$

- Beweise algebraische Gesetze regulärer Ausdrücke

- Liefert Hilfsmittel zur Vereinfachung regulärer Ausdrücke

- Gesetze für Einheiten und Annihilatoren

- $\emptyset + E \cong E \cong E + \emptyset$ :  
$$L(\emptyset + E) = L(\emptyset) \cup L(E) = \emptyset \cup L(E) = L(E)$$

- $\epsilon \circ E \cong E \cong E \circ \epsilon$ :  
$$L(\epsilon \circ E) = L(\epsilon) \circ L(E) = \{\epsilon\} \circ L(E) = L(E)$$

- $\emptyset \circ E \cong \emptyset \cong E \circ \emptyset$ :  
$$L(\emptyset \circ E) = L(\emptyset) \circ L(E) = \emptyset \circ L(E) = \emptyset = L(\emptyset)$$

- Kommutativitätsgesetz für  $+$

- $E + F \cong F + E$ :  
$$L(E + F) = L(E) \cup L(F) = L(F) \cup L(E) = L(F + E)$$

- Kommutativität von  $\circ$  gilt nicht:  
$$= L(\textcolor{red}{01}) = \{01\} \neq \{10\} = L(\textcolor{red}{10})$$

# “RECHENREGELN” FÜR REGULÄRE AUSDRÜCKE II

- **Gesetze für Assoziativität von  $\circ$  und  $+$** 
  - $(E \circ F) \circ G \cong E \circ (F \circ G)$ :  
 $L((E \circ F) \circ G) = L(E \circ F) \circ L(G) = L(E) \circ L(F) \circ L(G) = L(E) \circ L(F \circ G) = L(E \circ (F \circ G))$
  - $(E + F) + G \cong E + (F + G)$ :  
 $L((E + F) + G) = L(E + F) \cup L(G) = L(E) \cup L(F) \cup L(G) = \dots = L(E + (F + G))$
- **Distributivgesetze**
  - $(E + F) \circ G \cong E \circ G + F \circ G$ :  
$$\begin{aligned} L((E + F) \circ G) &= (L(E) \cup L(F)) \circ L(G) \\ &= \{w \in \Sigma^* \mid \exists u \in L(E) \cup L(F). \exists v \in L(G). w = uv\} \\ &= \{w \in \Sigma^* \mid \exists u \in L(E). \exists v \in L(G). w = uv \vee \exists u \in L(F). \exists v \in L(G). w = uv\} \\ &= \{w \in \Sigma^* \mid \exists u \in L(E). \exists v \in L(G). w = uv\} \cup \{w \in \Sigma^* \mid \exists u \in L(F). \exists v \in L(G). w = uv\} \\ &= L(E) \circ L(G) \cup L(F) \circ L(G) = L(E \circ G + F \circ G) \end{aligned}$$
  - $G \circ (E + F) \cong G \circ E + G \circ F$
- **Idempotenz von  $+$ :**  $E + E \cong E$
- **Hüllengesetze:**
  - $\emptyset^* \cong \epsilon, \epsilon^* \cong \epsilon, (E^*)^* \cong E^*$
  - $E^+ \cong E \circ E^* \cong E^* \circ E, E^* \cong \epsilon + E^+$

# BEWEISMETHODIK FÜR WEITERE ÄQUIVALENZEN

- **Beispiel: Nachweis von  $(E+F)^* \cong (E^*F^*)^*$**

- Sei  $w \in L((E+F)^*)$
- Dann  $w = w_1..w_k$  mit  $w_i \in L(E)$  oder  $w_i \in L(F)$  für alle  $i$
- Dann  $w = w_1..w_k$  mit  $w_i \in L(E^*F^*)$  für alle  $i$  (semantisches Argument)
- Also  $w \in L((E^*F^*)^*)$

- **Beweis verwendet keine Information über  $E$  und  $F$**

- Man könnte genauso gut  $(a+b)^* \cong (a^*b^*)^*$  testen
- $(E+F)^* \cong (E^*F^*)^*$  gilt, weil  $(a+b)^* \cong (a^*b^*)^*$  gilt

- **Allgemeines Beweisverfahren**

- $E$  regulärer Ausdruck mit Metavariablen  $E_1,..,E_m$  für Sprachen  $L_1,..,L_m$
- Ersetze im Beweis für  $E \cong F$  alle Metavariablen durch Symbole  $a \in \Sigma$
- Teste Äquivalenz der konkreten Ausdrücke mit automatischem Prüfverfahren
  - Einheit 2.5

Korrektheitsbeweis: Induktion über Struktur regulärer Ausdrücke

# UMWANDLUNG REGULÄRER AUSDRÜCKE IN AUTOMATEN

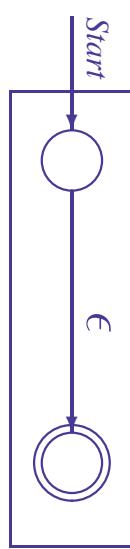
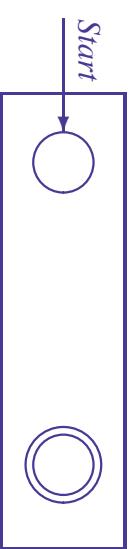
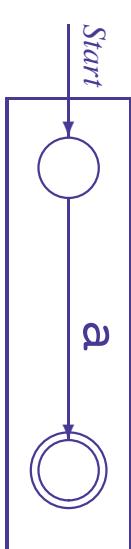
## Sprachen regulärer Ausdrücke sind endlich erkennbar

Für jeden regulären Ausdruck  $E$  gibt es einen  $\epsilon$ -NEA  $A$  mit

- $A$  hat genau einen akzeptierenden Zustand  $q_f$
- Der Startzustand von  $A$  ist in keinem  $\delta_A(q, a)$  enthalten
- Für alle  $a \in \Sigma$  ist  $\delta_A(q_f, a) = \emptyset$
- $L(E) = L(A)$

Beweis durch strukturelle Induktion über Aufbau regulärer Ausdrücke

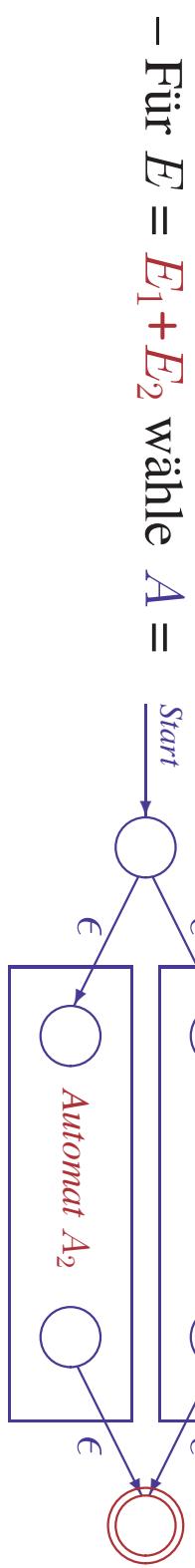
- Induktionsanfänge

- Für  $E = \epsilon$  wähle  $A =$   

- Für  $E = \emptyset$  wähle  $A =$   

- Für  $E = a$  wähle  $A =$   

- Für  $E = a \text{ } b$  wähle  $A$  aus  
– Korrektheit offensichtlich, da jeweils maximal ein Zustandsübergang

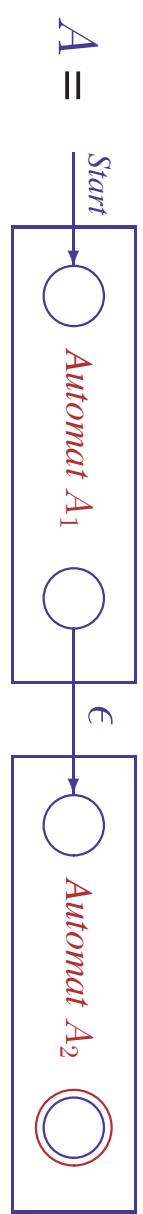
# UMWANDLUNG REGULÄRER AUSDRÜCKE IN AUTOMATEN

- Induktionsannahme: seien  $A_1$  und  $A_2$   $\epsilon$ -NEAs für  $E_1$  und  $E_2$

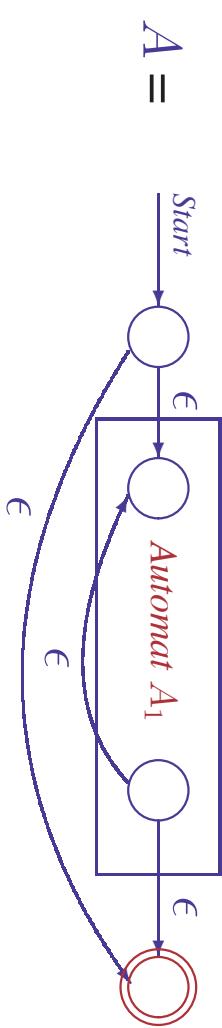
- Induktionssschritt



- Für  $E = E_1 \circ E_2$  wähle



- Für  $E = E_1^*$  wähle



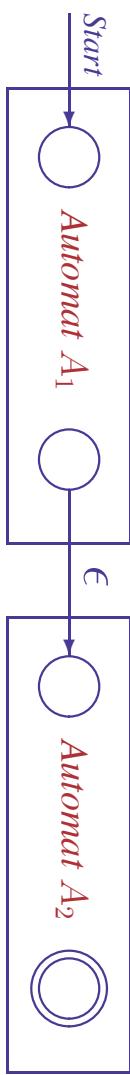
- Für  $E = (E_1)$  wähle  $A = A_1$

## KORREKTHEIT DER UMWANDLUNGEN

- Klammern ändern nichts

– Es ist  $L((E_1)) = L(E_1) = L(A_1) = L(A)$

- Verkettung ist Verschaltung von Automaten



Es gilt  $w \in L(E_1 \circ E_2)$

$$\Rightarrow w \in L(E_1) \circ L(E_2) = L(A_1) \circ L(A_2)$$

$$\Rightarrow \exists u \in L(A_1). \exists v \in L(A_2). w = uv$$

$$\Rightarrow \exists u, v \in \Sigma^*. w = uv \wedge q_{f,1} \in \hat{\delta}_1(q_{0,1}, u) \wedge q_{f,2} \in \hat{\delta}_2(q_{0,2}, v)$$

$$\Rightarrow \exists u, v \in \Sigma^*. w = uv \wedge q_{0,2} \in \hat{\delta}(q_{0,1}, u) \wedge q_{f,2} \in \hat{\delta}(q_{0,2}, v) \quad (q_{0,2} \in \text{ε-Hüllle}(q_{f,1}))$$

$$\Rightarrow q_{f,2} \in \hat{\delta}(q_{0,1}, w)$$

$$\Rightarrow w \in L(A)$$

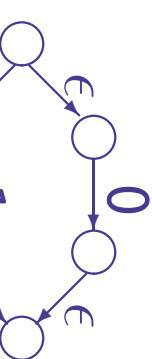
Argument ist umkehrbar, also  $w \in L(A) \Rightarrow w \in L(E_1 \circ E_2)$

- Sternbildung und Vereinigung ähnlich

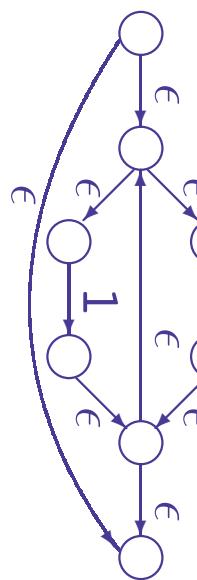
# UMWANDLUNG REGULÄRER AUSDRÜCKE AM BEISPIEL

## Konstruiere endlichen Automaten für $(0+1)^*1(0+1)^*$

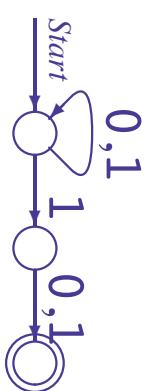
- Teilautomat für  $(0+1)$



- Automat für  $(0+1)^*1(0+1)^*$



- Elimination von ε-Ubergängen



# UMWANDLUNG VON ( $\epsilon$ -)NEAS IN REGULÄRE AUSDRÜCKE

## • Ursprünglich: Pfadanalyse im Übergangsdiagramm

- Spezialisierung eines allgemeinen Verfahrens für Pfadanalyse in Graphen
- Definiere reguläre Ausdrücke für Pfade durch Automaten
- Berechnung Ausdrücke iterativ und kombiniere alle relevanten Ausdrücke
- Kompliziertes und aufwendiges Verfahren

[Mehr dazu im Anhang](#)

## • Effizienterer Zugang: Elimination von Zuständen

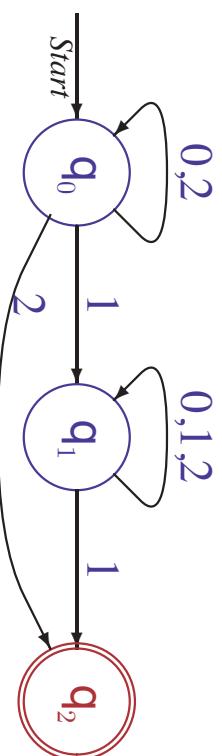
- Beschreibe Übergänge  $q_i \xrightarrow{a \in \Sigma} q_j$  durch reguläre Ausdrücke
- Beginne mit regulären Ausdrücken für direkte Übergänge
- Entferne einzelne Zustände und beschreibe die entstehenden Ausdrücke
- Liefert Ausdrücke für Übergänge zwischen Start- und Endzuständen

## • Technisches Hilfsmittel: verallgemeinerte NEAS (VNEAS)

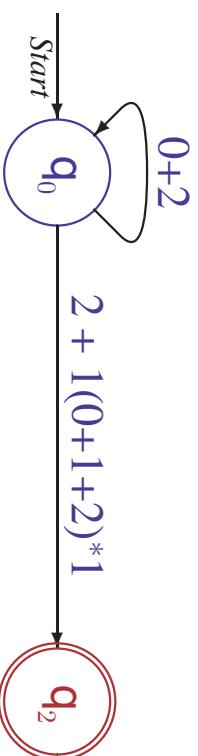
- NEA, dessen Überführungsfunktion  $\delta$  auf regulären Ausdrücken arbeitet
- **A akzeptiert  $w$** , wenn es einen **Pfad**  $w = v_1..v_m$  von  $q_0$  zu einem  $q \in F$  gibt und alle  $v_i$  in der Sprache des entsprechenden regulären Ausdrucks liegen
- Konsistente Formalisierung mühsam und ohne Erkenntnisgewinn

## ZUSTANDELIMINATION IN VNEAs

- Ursprünglicher NEA



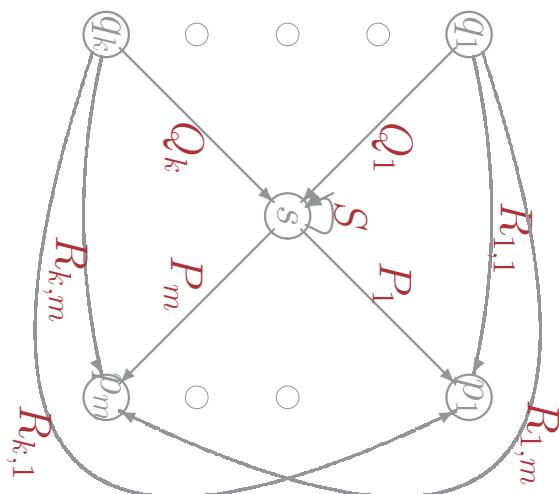
- Zugehöriger VNEA



- Nach Elimination von  $q_1$

- Ausdruck für Übergang von  $q_0$  nach  $q_2$  ergibt sich aus  
Übergang  $q_0$  nach  $q_1$ , Schleife bei  $q_1$ , Übergang  $q_1$  nach  $q_2$  und  
existierendem Ausdruck für direkten Übergang von  $q_0$  nach  $q_2$

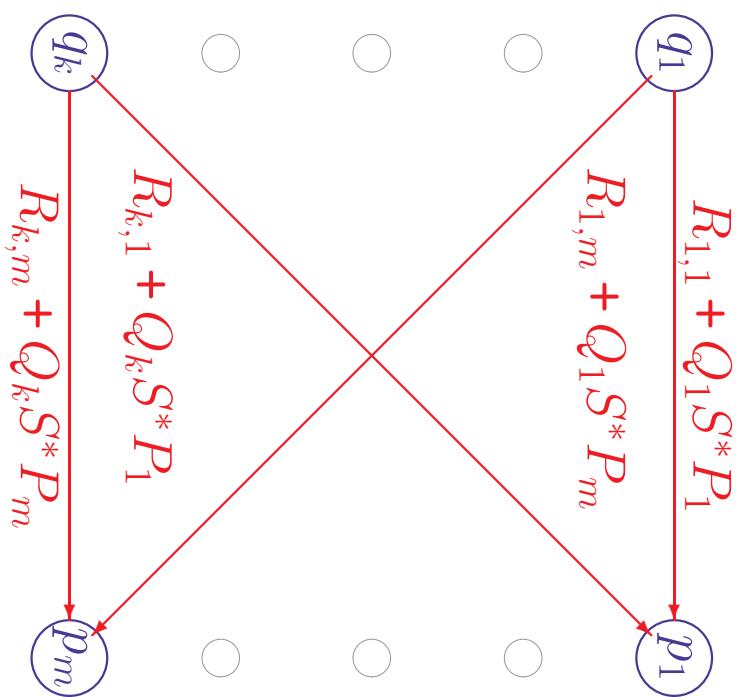
# ALLGEMEINE ZUSTANDSELEIMINATION IN VNEAS



**Eliminiere Zustand  $s$**

mit **Vorgängern**  $q_1, \dots, q_k$   
und **Nachfolgern**  $p_1, \dots, p_m$

- Eliminiere Pfad von  $q_1$  nach  $p_1$  über  $s$ :  $R_{1,1} + Q_1 S^* P_1$
- ⋮
- Eliminiere Pfad von  $q_k$  nach  $p_m$  über  $s$ :  $R_{k,m} + Q_k S^* P_m$
- ⋮
- Eliminiere Pfad von  $q_k$  nach  $p_m$  über  $s$ :  $R_{k,m} + Q_k S^* P_m$



## UMWANDLUNG DURCH ZUSTANDELIMINATION

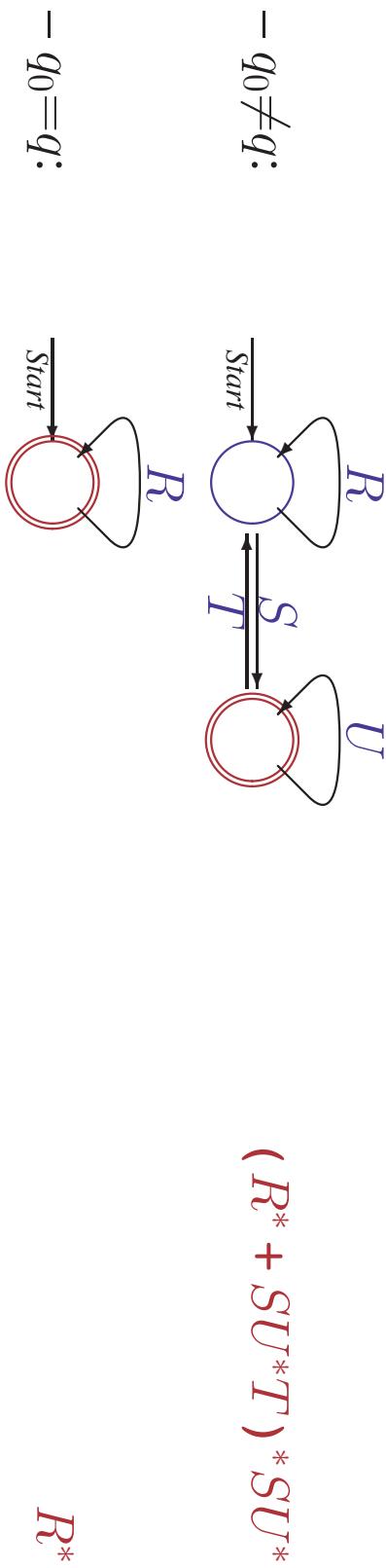
### 1. Transformiere endlichen Automaten in VNEA

– Ersetze Beschriftungen mit Symbolen  $a \in \Sigma$  durch reguläre Ausdrücke

### 2. Für $q \in F$ eliminiere alle Zustände außer $q_0$ und $q$

– Iterative Anwendung des Eliminationsverfahrens

### 3. Bilde regulären Ausdruck aus finalem Automaten



–  $q_0 \neq q$ :



–  $q_0 = q$ :



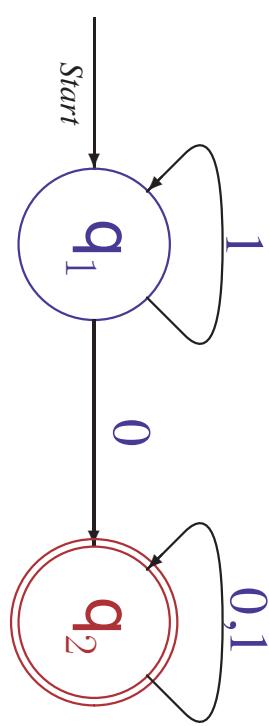
### 4. Vereinige Ausdrücke aller Endzustände

– Bilde Summe aller entstandenen regulären Ausdrücke

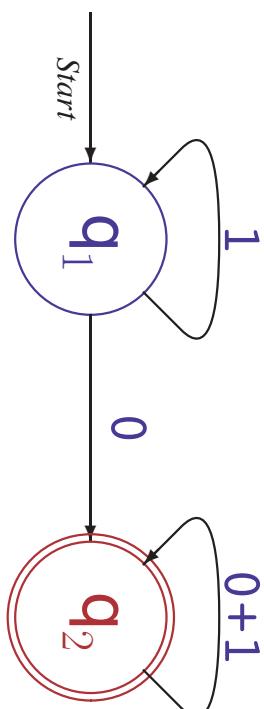
– Nicht erforderlich, wenn Automat nur einen Endzustand hat

→ Ergänze neuen Anfangs/Endzustand ohne ein/ausgehende Kanten

# UMWANDLUNG DURCH ZUSTANDELIMINATION: BEISPIEL



- Transformiere in VNEA



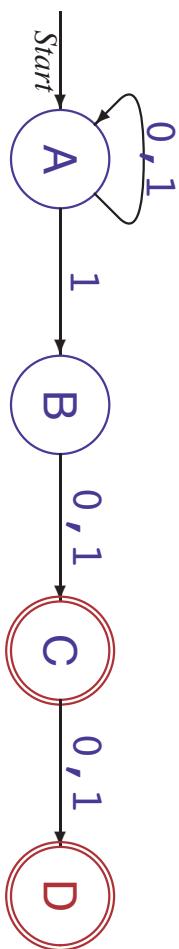
- Keine Zustände zu eliminieren

- Bilde regulären Ausdruck aus finalem Automaten

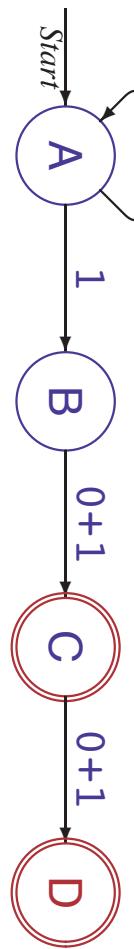
- Extrahierter Ausdruck:  $(1^* + 0(0+1)^*\emptyset)^*0(0+1)^*$
- Nach Vereinfachung:  
$$\boxed{1^*0(0+1)^*}$$

Umwandlung mit Pfadanalyseverfahren würde 12 aufwendige Schritte erfordern

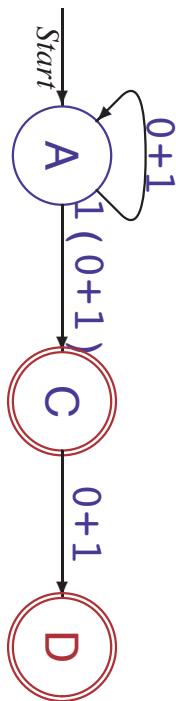
## UMWANDLUNG DURCH ZUSTANDELEMINATION II



- Transformiere in VNEA



- Elimination von Zustand **B**



- Elimination von Zustand **C** für Endzustand **D**



- Elimination von Zustand **D** für Endzustand **C**



- Gesamter Ausdruck:

$(0+1)^*1(0+1) + (0+1)^*1(0+1)(0+1)$

- **Algebraische Notation für Sprachen**

- $\epsilon, \emptyset$ , Symbole des Alphabets, Vereinigung, Verkettung, Sternoperator
- Äquivalent zu endlichen Automaten
- Gut zum Nachweis algebraischer Gesetze von Sprachen
- Anwendung in Programmiersprachen und Suchmaschinen

- **Transformation in endliche Automaten**

- Iterative Konstruktion von  $\epsilon$ -NEAs
- Nachträgliche Optimierung durch Elimination von  $\epsilon$ -Übergängen

- **Transformation von Automaten in Ausdrücke**

- Konstruktion durch Elimination von Zuständen in VNEAs
- Historisch: Konstruktion von Ausdrücken für Abarbeitungspfade
- Nachträgliche Optimierungen durch Anwendung algebraischer Gesetze

# ANNEHÄNG

## Originalmethode: allgemeines Graphanalyseverfahren

- Gegeben  $\text{DEA } A = (\{q_1, \dots, q_n\}, \Sigma, \delta, q_1, \{q_{f_1}, \dots, q_{f_m}\})$
- Definiere Ausdrücke für Pfade durch  $A$ 
  - $\textcolor{red}{R}_{ij}^k$ : Regulärer Ausdruck für Menge der Wörter  $w$  mit  $\hat{\delta}(q_i, w) = q_j$ ,  
so dass für alle  $\epsilon \neq v \sqsubseteq w$  ( $v \neq w$ ) gilt:  $\hat{\delta}(q_i, v) = q_m \Rightarrow m \leq k$   
(Abarbeitung von  $w$  berührt keinen Zustand größer als  $k$ )
- Setze die  $R_{ij}^k$  zu Ausdruck für  $L(A)$  zusammen
  - Per Definition ist  $R_{ij}^n$  ein Ausdruck für Wörter  $w$  mit  $\hat{\delta}(q_i, w) = q_j$
  - Setze  $\textcolor{red}{R} = R_{1f_1}^n + \dots + R_{1f_m}^n$
  - Dann gilt  $L(\textcolor{blue}{R}) = \bigcup_{j=1}^m \{w \in \Sigma^* \mid \hat{\delta}(q_1, w) = q_{f_j}\}$   
 $= \{w \in \Sigma^* \mid \exists q \in \{q_{f_1}, \dots, q_{f_m}\}. \hat{\delta}(q_1, w) = q\} = L(A)$

# ITERATIVE BESTIMMUNG DER AUSDRÜCKE $R_{ij}^k$

- **Basisfall  $R_{ij}^0$ :** Pfad darf zwischendurch keine Zustände berühren
  - Pfadlänge 0 (nur für  $i=j$ ):  $\epsilon \in L(R_{ii}^0)$
  - Pfadlänge 1:  $\{a \in \Sigma \mid \delta(q_i, a) = q_j\} \subseteq L(R_{ij}^0)$
  - Ergebnis:  $R_{ii}^0 = \epsilon + \mathbf{a}_1 + \dots + \mathbf{a}_k$ , wobei  $\{a \in \Sigma \mid \delta(q_i, a) = q_j\} = \{a \in \Sigma \mid \delta(q_i, a) = q_j\}$
  - $R_{ij}^0 = \emptyset + \mathbf{a}_1 + \dots + \mathbf{a}_k \quad (i \neq j)$
- **Schrittfall  $R_{ij}^k$  ( $0 < k \leq n$ ):** zwei Alternativen
  - Wörter  $w \in L(R_{ij}^k)$ , deren Pfad  $q_k$  nicht enthält, gehören zu  $L(R_{ij}^{k-1})$
  - Wörter  $w \in L(R_{ij}^k)$ , deren Pfad  $q_k$  enthält:
 

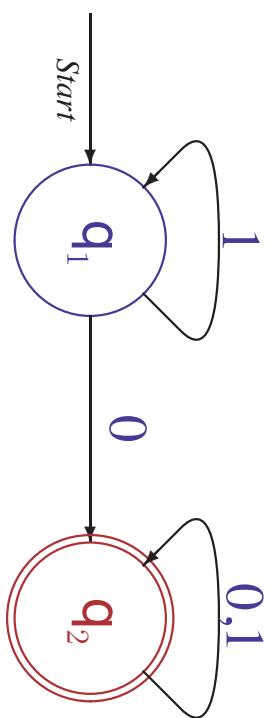
Zerlege  $w$  in  $uz_1..z_pv$  mit  $\hat{\delta}(q_i, u) = q_k \wedge \forall l \leq p. \hat{\delta}(q_k, z_l) = q_k \wedge \hat{\delta}(q_k, v) = q_j$

$R_{ik}^{k-1}$       Null oder mehr Wörter in  $R_{ik}^{k-1}$        $R_{kj}^{k-1}$
- Ergebnis:  $R_{ij}^k = R_{ij}^{k-1} + R_{ik}^{k-1} \circ (R_{kk}^{k-1})^* \circ R_{kj}^{k-1}$

# UMWANDLUNG VON AUTOMATEN AM BEISPIEL

- Basisfall

$$\begin{aligned}
 R_{11}^0 &= \epsilon + 1 \\
 R_{12}^0 &= 0 \\
 R_{21}^0 &= \emptyset \\
 R_{22}^0 &= \epsilon + 0 + 1
 \end{aligned}$$



- Stufe 1

$$\begin{aligned}
 R_{11}^1 &= R_{11}^0 + R_{11}^0(R_{11}^0)^*R_{11}^0 = \epsilon + 1 + (\epsilon + 1)(\epsilon + 1)^*(\epsilon + 1) && \mapsto 1^* \\
 R_{12}^1 &= R_{12}^0 + R_{11}^0(R_{11}^0)^*R_{12}^0 = 0 + (\epsilon + 1)(\epsilon + 1)^*0 && \mapsto 1^*0 \\
 R_{21}^1 &= R_{21}^0 + R_{21}^0(R_{11}^0)^*R_{11}^0 = \emptyset + \emptyset(\epsilon + 1)^*(\epsilon + 1) && \mapsto \emptyset \\
 R_{22}^1 &= R_{22}^0 + R_{21}^0(R_{11}^0)^*R_{12}^0 = \epsilon + 0 + 1 + \emptyset(\epsilon + 1)^*0 && \mapsto \epsilon + 0 + 1
 \end{aligned}$$

- Stufe 2

$$\begin{aligned}
 R_{11}^2 &= R_{11}^1 + R_{12}^1(R_{22}^1)^*R_{21}^1 = 1^* + 1^*0(\epsilon + 0 + 1)^*\emptyset && \mapsto 1^* \\
 R_{12}^2 &= R_{12}^1 + R_{12}^1(R_{22}^1)^*R_{22}^1 = 1^*0 + 1^*0(\epsilon + 0 + 1)^*(\epsilon + 0 + 1) && \mapsto 1^*0(0+1)^* \\
 R_{21}^2 &= R_{21}^1 + R_{22}^1(R_{22}^1)^*R_{21}^1 = \emptyset + (\epsilon + 0 + 1)(\epsilon + 0 + 1)^*\emptyset && \mapsto \emptyset \\
 R_{22}^2 &= R_{22}^1 + R_{22}^1(R_{22}^1)^*R_{22}^1 = (\epsilon + 0 + 1) + (\epsilon + 0 + 1)(\epsilon + 0 + 1)^*(\epsilon + 0 + 1) && \mapsto (0+1)^*
 \end{aligned}$$

**Regulärer Ausdruck des Automaten ist:  $1^*0(0+1)^*$**

# DAS PFADANALYSEVERFAHREN IST ZU KOMPLIZIERT

- Konstruktion aller  $R_{ij}^k$  ist aufwendig
  - Es müssen mehr als  $n^3$  Ausdrücke  $R_{ij}^k$  erzeugt werden
  - Ausdrücke  $R_{ij}^k$  können viermal so groß wie die  $R_{ij}^{k-1}$  werden
  - Ohne Vereinfachung der  $R_{ij}^k$  sind bis zu  $n^3 * 4^n$  Symbole zu erzeugen
- Optimierungen des Verfahrens sind möglich
  - Vermeide Vielfachkopien der  $R_{ij}^{k-1}$
  - Vereinfache Ausdrücke  $R_{ij}^k$  direkt nach Erzeugung
  - Liefert keine grundsätzliche Verbesserung

Zustandselimination ist erheblich effizienter

# Theoretische Informatik I

## Einheit 2.4

### Grammatiken



1. Arbeitsweise
2. Klassifizierung
3. Beziehung zu Automaten



# BESCHREIBUNGSFORMEN FÜR SPRACHEN

- **Mathematische Mengennotation**

- Prädikate beschreiben **Eigenschaften** der Wörter
- Extrem flexibel, nicht notwendig “berechenbar”

- **Endliche Automaten**

- Beschreibung der Verarbeitung von Sprachen
- Schwerpunkt ist Erkennen korrekter Wörter

- **Reguläre Ausdrücke**

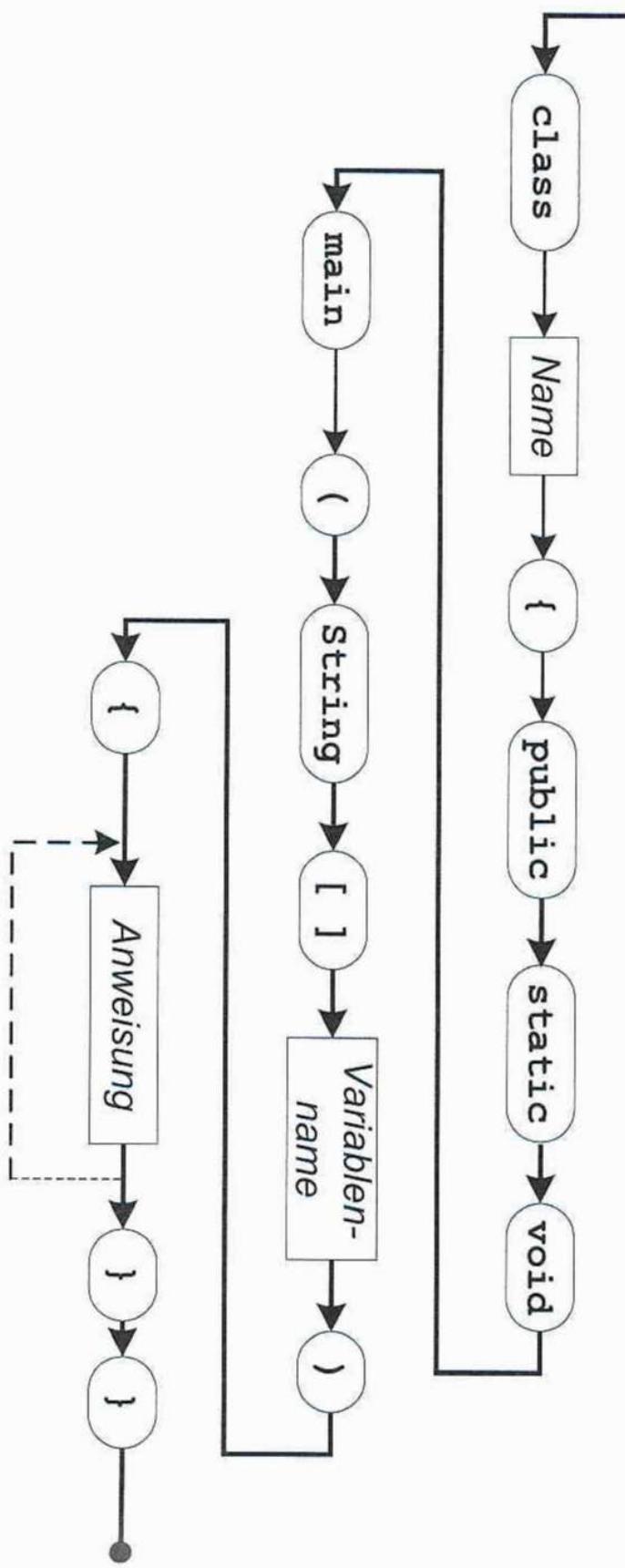
- Beschreibung der Struktur der Sprache

- **Grammatiken**

- Produktionsregeln beschreiben Aufbau der Wörter
- Auch für komplexere Strukturen als reguläre Sprachen
- Gängig für die Beschreibung von Programmiersprachen

# BEISPIEL: AUSZUG DER GRAMMATIK VON JAVA

Java-Programm  
Applikation



# KOMPONENTEN VON GRAMATIKEN

- **Terminalsymbole: Alphabet der Sprache**

- Symbole, aus denen die erzeugten Wörter bestehen sollen
- Bei Programmiersprachen meist ASCII-Symbole ohne Kontrollzeichen

- **Variablen: Hilfsalphabet für Verarbeitung**

- Beschreiben die syntaktischen Kategorien der Sprache
- Bei JAVA z.B. **Applikation, Name, Variablename, Anweisung, ...**
- Andere Bezeichnung: **Nichtterminale Symbole**

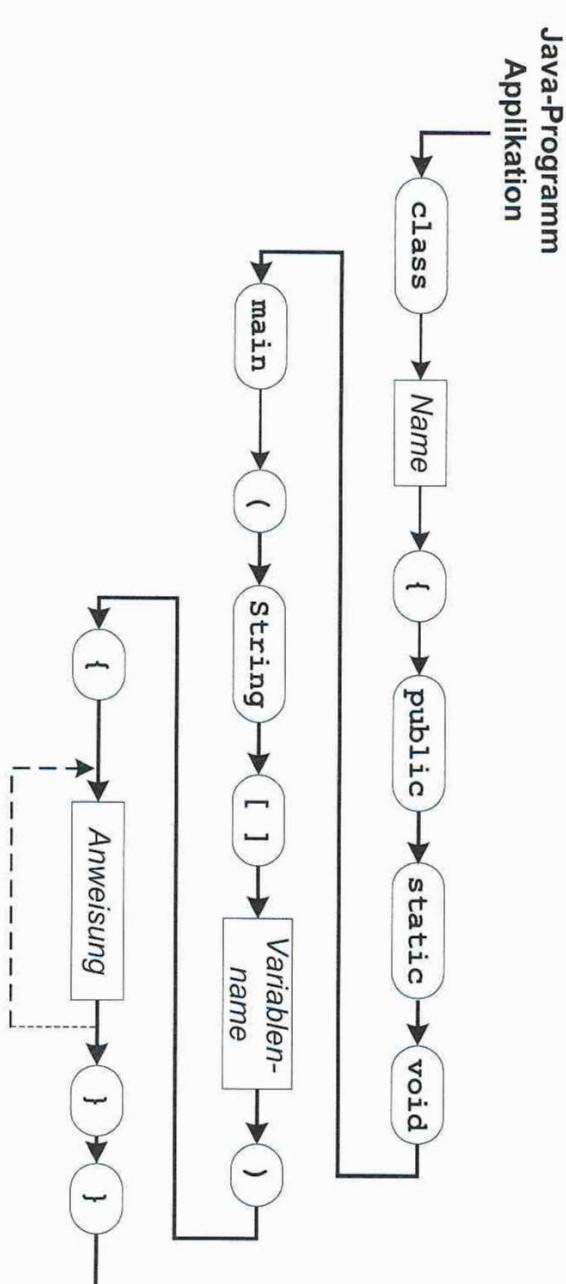
- **Produktionen: Regeln zur Erzeugung von Wörtern**

- Erklären wie syntaktischen Kategorien aufgebaut sind
- Erklären Erzeugung von Wörtern der Sprache in den einzelnen Kategorien
- z.B. “Eine Applikation beginnt mit `class` gefolgt von einem Namen, ...”

- **Startsymbol**

- Erklärt welche syntaktische Kategorie beschrieben werden soll

# GRAMMATIKEN – MATHEMATISCH PRÄZISIERT



## Eine Grammatik ist ein 4-Tupel $G = (V, T, P, S)$ mit

- $V$  endliches Hilfsalphabet
- $T$  endliches Terminalalphabet mit  $V \cap T = \emptyset$
- $P \subseteq \Gamma^+ \times \Gamma^*$  endliche Menge der Produktionen (wobei  $\Gamma = V \cup T$ )
- Schreibweise für Produktionen:  $l \rightarrow r \in P$  statt  $(l, r) \in P$
- $S \in V$  Startsymbol

## ARBEITSWEISE: PRODUKTION VON WÖRTERN DER ZIELSPRACHE

- $G_1 = (\{S\}, \{0, 1\}, P, S)$  mit  $P = \{S \rightarrow S1, S \rightarrow S0, S \rightarrow \epsilon\}$

Erzeugung von Wörtern:

$$S \rightarrow \epsilon$$

$$S \rightarrow S0 \rightarrow 0$$

$$S \rightarrow S0 \rightarrow S10 \rightarrow S010 \rightarrow S0010 \rightarrow 0010$$

- Nur Wörter über dem Terminalalphabet sind von Interesse
- $\epsilon, 0, 0010$  gehören zur erzeugten Sprache
- $S, S0, S10, S010, S0010$  sind nur “Zwischenschritte”

- $G_2 = (\{S, A, B, C\}, \{0, 1\}, P, S)$  mit

$$P = \{S \rightarrow B, S \rightarrow CA0, A \rightarrow BBB, B \rightarrow C1, B \rightarrow 0, CC1 \rightarrow \epsilon\}$$

Ableitungen:

$$S \rightarrow B \rightarrow 0$$

$$S \rightarrow B \rightarrow C1$$

Erfolglos, kein Wort der Zielsprache erreichbar

$$S \rightarrow CA0 \rightarrow CABBB0 \rightarrow CC1BB0 \rightarrow BB0 \rightarrow 0B0 \rightarrow 000$$

✓

# ARBEITSWEISE VON GRAMMATIKEN – PRÄZISIERT

- **Ableitungsrelation**  $\longrightarrow \subseteq \Gamma^+ \times \Gamma^*$

- $w \longrightarrow z \equiv \exists x, y \in \Gamma^*. \exists l \rightarrow r \in P. w = x \textcolor{red}{l} y \wedge z = x \textcolor{red}{r} y$

Anwendung von Produktionen auf Wörter

- **Erweiterte Ableitungsrelation**  $\xrightarrow{*} \subseteq \Gamma^+ \times \Gamma^*$

- $w \xrightarrow{0} z \equiv w = z$
- $w \xrightarrow{n+1} z \equiv \exists u \in \Gamma^*. w \longrightarrow u \wedge u \xrightarrow{n} z$
- $w \xrightarrow{*} z \equiv \exists n \in \mathbb{N}. w \xrightarrow{n} z$
- Grammatik durch optionalen Index  $G$  ( $\xrightarrow{*} G$ ) spezifizierbar

- **Von  $G$  erzeugte Sprache**

- Menge der Terminalwörter, die aus  $S$  abgeleitet werden können

$$L(G) \equiv \{w \in T^* \mid S \xrightarrow{*} G w\}$$

# GRAMMATIK FÜR $L = \{0^k 1^l \mid k \leq l\}$

- $G_3 = (\{S\}, \{\mathbf{0}, \mathbf{1}\}, P, S)$  mit  $P = \{S \rightarrow S\mathbf{1}, S \rightarrow 0S\mathbf{1}, S \rightarrow \epsilon\}$
- Zeige  $L(G_3) = L$  per Induktion über Länge der Ableitung
  - Ableitungen der Länge 0 liefern keine Terminalwörter
  - Zeige:  $\forall l \in \mathbb{N}. \forall w \in \{\mathbf{0}, \mathbf{1}\}^*. S \xrightarrow{l+1} w \Leftrightarrow (\exists k \leq l. w = 0^k 1^l)$
- Basisfall
  - $S \xrightarrow{1} w \Leftrightarrow (S \rightarrow w) \in P \Leftrightarrow w = \epsilon \Leftrightarrow \exists k \leq 0. w = 0^k 1^0$  ✓
- Induktionssschritt
  - Es gelte  $\forall w \in \{0, 1\}^*. S \xrightarrow{l+1} w \Leftrightarrow (\exists k \leq l. w = 0^k 1^l)$
  - $$\begin{aligned} & S \xrightarrow{l+2} v \\ \Leftrightarrow & S \rightarrow S\mathbf{1} \xrightarrow{l+1} v \vee S \rightarrow 0S\mathbf{1} \xrightarrow{l+1} v \\ \Leftrightarrow & \exists w \in \{0, 1\}^*. S \xrightarrow{l+1} w \wedge (v = w\mathbf{1} \vee v = 0w\mathbf{1}) \\ \Leftrightarrow & \exists w \in \{0, 1\}^*. \exists k \leq l. w = 0^k 1^l \wedge (v = w\mathbf{1} \vee v = 0w\mathbf{1}) \quad (\text{Annahme}) \\ \Leftrightarrow & \exists k \leq l. v = 0^k 1^{l+1} \vee v = 0^{k+1} 1^{l+1} \\ \Leftrightarrow & \exists k \leq (l+1). v = 0^k 1^{l+1} \end{aligned}$$
 ✓

# KLASSIFIZIERUNG VON GRAMMATIKEN

- **allgemein (Typ 0):** keine Einschränkung an die Produktionen
- **kontextsensitiv (Typ 1)**
  - nur Regeln der Form  $x A y \rightarrow x z y$  oder  $S \rightarrow \epsilon$  ( $x, y, z \in \Gamma^*, A \in V, z \neq \epsilon$ )  
( $S \rightarrow \epsilon$  nur erlaubt, wenn  $S$  nicht rechts in einer anderen Regel auftaucht)
- **expansiv**
  - nur Regeln der Form  $x \rightarrow z$  mit  $|x| \leq |z|$ , oder  $S \rightarrow \epsilon$  ( $x \in \Gamma^+, z \in (\Gamma - \{S\})^+$ )
- **kontextfrei (Typ 2)**
  - nur Regeln der Form  $A \rightarrow z$  ( $z \in \Gamma^*, A \in V$ )
- **linear**
  - nur Regeln der Form  $A \rightarrow \epsilon$  oder  $A \rightarrow u B v$  ( $A, B \in V, u, v \in T^*$ )
- **rechtslinear (Typ 3)**
  - nur Regeln der Form  $A \rightarrow \epsilon$  oder  $A \rightarrow a B$   
Manche Bücher: nur Regeln der Form  $A \rightarrow \epsilon$  oder  $A \rightarrow v B$  ( $A, B \in V, a \in T$ )
  - **linkslinear**
    - nur Regeln der Form  $A \rightarrow \epsilon$  oder  $A \rightarrow B a$  ( $A, B \in V, a \in T$ )

## BEISPIELE FÜR GRAMMATIKKLASSEN

- **kontextsensitiv:** Regeln  $x A y \rightarrow x z y$  oder  $S \rightarrow \epsilon$
- **expansiv:** Regeln  $x \rightarrow z$  mit  $|x| \leq |z|$ , oder  $S \rightarrow \epsilon$
- **kontextfrei:** Regeln  $A \rightarrow z$
- **linear:** Regeln  $A \rightarrow \epsilon$  oder  $A \rightarrow u B v$
- **rechtslinear:** Regeln  $A \rightarrow \epsilon$  oder  $A \rightarrow a B$
- **linkslinear:** Regeln  $A \rightarrow \epsilon$  oder  $A \rightarrow B a$

- $G_1 = (\{S\}, \{\mathbf{0}, \mathbf{1}\}, P, S)$  mit  $P = \{S \rightarrow S\mathbf{1}, S \rightarrow S\mathbf{0}, S \rightarrow \epsilon\}$ 
  - linkslinear, kontextfrei, nicht expansiv, nicht kontextsensitiv ( $S$  rechts,  $S \rightarrow \epsilon$ )
- $G_2 = (\{S, A, B, C\}, \{\mathbf{0}, \mathbf{1}\}, P, S)$  mit  
 $P = \{S \rightarrow B, S \rightarrow CA\mathbf{0}, A \rightarrow BBB, B \rightarrow C\mathbf{1}, B \rightarrow \mathbf{0}, CC\mathbf{1} \rightarrow \epsilon\}$ 
  - allgemein (keine anderen Bedingungen erfüllt)
- $G_3 = (\{S\}, \{\mathbf{0}, \mathbf{1}\}, P, S)$  mit  $P = \{S \rightarrow S\mathbf{1}, S \rightarrow \mathbf{0}S\mathbf{1}, S \rightarrow \epsilon\}$ 
  - linear, kontextfrei, nicht expansiv, nicht kontextsensitiv
- $G_4 = (\{S, A, B, C\}, \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}, P, S)$  mit  $P = \{S \rightarrow aCBC, S \rightarrow aBC, CB \rightarrow BC, aB \rightarrow ab, bB \rightarrow bb, bC \rightarrow bc, cC \rightarrow cc\}$ 
  - expansiv, nicht kontextfrei, nicht kontextsensitiv

## SPRACHKLASSEN

- **Typ-0 Sprachen**
    - Sprachen der Form  $L = L(G)$  für eine beliebige Grammatik  $G$
  - **Typ-1 Sprachen (kontextsensitive Sprachen)**
    - Sprachen der Form  $L = L(G)$  für eine kontextsensitive Grammatik  $G$
    - $L$  ist kontextsensitiv g.d.w.  $L = L(G)$  für eine expansive Grammatik  $G$
  - **Typ-2 Sprachen (kontextfreie Sprachen)**
    - Sprachen der Form  $L = L(G)$  für eine kontextfreie Grammatik  $G$
  - **Lineare Sprachen**
    - Sprachen der Form  $L = L(G)$  für eine lineare Grammatik  $G$
  - **Typ-3 Sprachen (reguläre Sprachen)**
    - Sprachen der Form  $L = L(G)$  für eine rechtslineare Grammatik  $G$
    - $L$  ist regulär g.d.w.  $L = L(G)$  für eine linkslineare Grammatik  $G$
- $\mathcal{L}_i \equiv \{ L \mid L \text{ ist Sprache vom Typ } i \}$

# Typ-3 SPRACHEN VS. REGULÄRE SPRACHEN

## Wie hängen Grammatiken und Automaten zusammen?

- Automaten verarbeiten Eingabewörter
    - Jedes Symbol wird in einem Schritt abgearbeitet
    - Symbol bestimmt, ob Automat im Zustand bleibt oder wechselt
  - Grammatiken erzeugen Wörter
    - Hilfsymbole werden im Endeffekt in Terminalwörter umgewandelt
    - Nichtlineare Grammatiken erzeugen mehrere Symbole gleichzeitig
    - Ableitungen in rechts-/linkslinearen Grammatiken erzeugen pro Schritt ein Terminalsymbol und verwenden jeweils nur ein Hilfsymbol
  - Wie kann man umwandeln?
    - Konstruiere zu jedem DEA eine äquivalente rechtslineare Grammatik
    - Konstruiere zu jeder rechtslinearen Grammatik einen äquivalenten DEA
- $$\hookrightarrow \mathcal{L}_3 = \{ L \mid L \text{ ist regulär} \}$$

# UMWANDLUNG VON DEAS IN TYP-3 GRAMMATIKEN

Für jeden DEA  $A$  gibt es eine  
Typ-3 Grammatik  $G$  mit  $L(G) = L(A)$

- **Gegeben** DEA  $A = (Q, \Sigma, \delta, q_0, F)$

- Wandle Abarbeitung von Symbolen in Erzeugung durch Grammatik um
- Setze  $G := (Q, \Sigma, P, q_0)$  mit  $P = \{q \xrightarrow{a} q' \mid \delta(q, a) = q'\} \cup \{q \xrightarrow{\epsilon} \mid q \in F\}$
- $G$  ist per Konstruktion rechtslinear, also vom Typ 3

- **Zeige**  $L(G) = L(A)$

$$\begin{aligned} w &= w_1..w_n \in L(G) \\ \Leftrightarrow & q_0 \xrightarrow{*} w_1..w_n \\ \Leftrightarrow & \exists q_1, .., q_n \in Q. \quad q_0 \longrightarrow w_1 q_1 \longrightarrow w_1 w_2 q_2 \longrightarrow ... \longrightarrow w_1..w_n q_n \longrightarrow w_1..w_n \\ \Leftrightarrow & \exists q_1, .., q_n \in Q. \quad q_0, w_1..w_n \vdash q_1, w_2..w_n \vdash ... \vdash q_{n-1}, w_n \vdash q_n, \epsilon \wedge q_n \in F \\ \Leftrightarrow & \exists q_n \in F. \quad q_0, w_1..w_n \vdash^* q_n, \epsilon \\ \Leftrightarrow & w \in L(A) \end{aligned}$$

✓

# UMWANDLUNG VON TYP-3 GRAMMATIKEN IN NEAS

Für jede Typ-3 Grammatik  $G$  gibt es einen  
NEA  $A$  mit  $L(A) = L(G)$

- **Gegeben Grammatik  $G = (V, T, P, S)$**

- Wandle Erzeugung von Symbolen in Abarbeitung durch NEA um
- Setze  $A := (V, T, \delta, S, F)$  mit  $\delta(X, a) = \{X' \mid X \xrightarrow{a} X' \in P\}$   
und  $F = \{X \in V \mid X \xrightarrow{} \epsilon \in P\}$

- **Zeige  $L(A) = L(G)$**

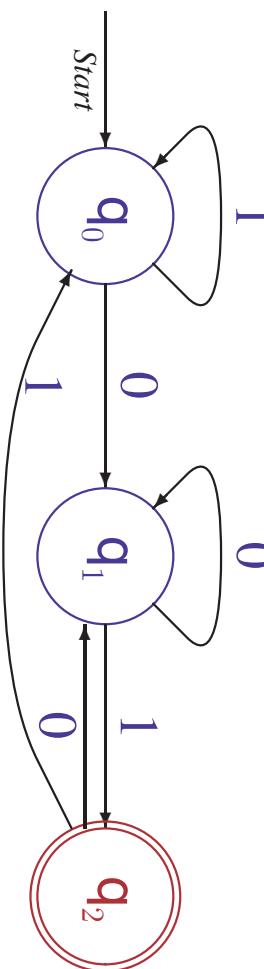
$$\begin{aligned} w &= w_1..w_n \in L(A) \\ \Leftrightarrow \exists X_n \in F. \quad S, w_1..w_n &\vdash^* X_n, \epsilon \\ \Leftrightarrow \exists X_1,..,X_n \in V. \quad S, w_1..w_n &\vdash \dots \vdash X_n, \epsilon \wedge X_n \in F \\ \Leftrightarrow \exists X_1,..,X_n \in V. \quad S &\longrightarrow w_1 X_1 \longrightarrow \dots \longrightarrow w_1..w_n X_n \longrightarrow w_1..w_n \\ \Leftrightarrow S &\xrightarrow{*} w \\ \Leftrightarrow w &\in L(G) \end{aligned}$$

✓

## UMWANDLUNGEN AM BEISPIEL

- Konvertiere DEA für  $(0+1)^*01$

–  $A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$  mit



- Erzeuge Grammatik

–  $G = (\{q_0, q_1, q_2\}, \{0, 1\}, P, q_0)$  mit

$$P = \{q_0 \rightarrow 1q_0, q_0 \rightarrow 0q_1, q_1 \rightarrow 1q_2, q_1 \rightarrow 0q_1, q_2 \rightarrow 1q_0, q_2 \rightarrow 0q_1, q_2 \rightarrow \epsilon\}$$

- Umwandlung von  $G$  in einen NEA

– Transformation erzeugt ursprünglichen Automaten

# DIE CHOMSKY HIERARCHIE

$$\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0$$

- **Wichtige Vertreter der Klassen**

- $\mathcal{L}_2 - \mathcal{L}_3$ :  $\{0^n 1^n \mid n \in \mathbb{N}\}$
- $\mathcal{L}_1 - \mathcal{L}_2$ :  $\{0^n 1^n 2^n \mid n \in \mathbb{N}\}$
- $\mathcal{L}_0 - \mathcal{L}_1$ :  $\{w_i \in \{0, 1\}^* \mid \text{Das Programm mit Codierung } w_i \text{ hält bei Eingabe } w_i\}$

- **Zugehörige Automatenmodelle**

- $\mathcal{L}_0$ : Turingmaschine
- $\mathcal{L}_1$ : linear platzbeschränkte nichtdeterministische Turingmaschine
- $\mathcal{L}_2$ : nichtdeterministischer endlicher Automat mit Kellerspeicher
- $\mathcal{L}_3$ : endlicher Automat

## Mehr in zukünftigen Vorlesungen

# Theoretische Informatik I

## Einheit 2.5

### Eigenschaften regulärer Sprachen



1. Abschlusseigenschaften
2. Prüfen von Eigenschaften
3. Wann sind Sprachen nicht regulär?

# WICHTIGE EIGENSCHAFTEN FORMALER SPRACHEN

- **Abschlusseigenschaften**

- Wie können Sprachen elegant zusammengesetzt werden?
- Erlaubt schematische Komposition von Sprachbausteinen

- **Entscheidbarkeitsfragen**

- Kann man bestimmte Eigenschaften automatisch testen?
- Wortproblem (Zugehörigkeit eines Wortes zur Sprache)
- Vergleiche zwischen Sprachen (nichtleer, Teilmenge, gleich, ...)

- **Grenzen einer Sprachklasse**

- Wie einfach strukturiert müssen die Sprachen der Klasse sein?
- Welche Sprachen gehören nicht zur Klasse?

**Aus theoretischer Sicht sind das  
die wirklich interessanten Fragen**

# ABSCHLUSSEIGENSCHAFTEN, WOZU?

## Zeige, dass bestimmte Operationen auf regulären Sprachen wieder zu regulären Sprachen führen

- **Wiederverwendung von “Sprachmodulen”**
  - Schematische Komposition von
    - Grammatiken zur Erzeugung von Sprachen
    - Automaten zur Erkennung von Sprachen
    - Regulären Ausdrücken
- **Schematische Konstruktion ist effektiver**
  - Fehlerfreier Aufbau sehr komplexer Grammatiken / Automaten
  - + Schematische Optimierung / Minimierung
  - Konstruktion “von Hand” oft fehleranfällig
- **Beispiel: Literale einer Programmiersprache**
  - Bilde Automaten für **Tokenklassen**: Zahlen, Bezeichner, Schlüsselwörter, ...
  - Konstruktion liefert Automaten für alle Arten von Literalen

## ABSCHLUSSEIGENSCHAFTEN, PRÄZISIERT

**Zeige:  $L_1, L_2$  regulär  $\Rightarrow L_1 \text{ op } L_2$  regulär**

- Es gilt **Abgeschlossenheit unter neun Operationen**

- Die Vereinigung zweier regulärer Sprachen ist regulär  $L_1 \cup L_2$
  - Das Komplement einer regulären Sprache ist regulär  $\bar{L}$
  - Der Durchschnitt zweier regulärer Sprachen ist regulär  $L_1 \cap L_2$
  - Die Differenz zweier regulärer Sprachen ist regulär  $L_1 - L_2$
  - Die Spiegelung einer regulären Sprache ist regulär  $L^R$
  - Die Hülle einer regulären Sprache ist regulär  $L^*$
  - Die Verkettung zweier regulärer Sprachen ist regulär  $L_1 \circ L_2$
  - Das Bild einer regulären Sprache unter Homomorphismen ist regulär  $h(L)$   $h^{-1}(L)$
  - Das Urbild ... " " ... unter Homomorphismen ist regulär
- Nachweis durch Verwendung aller Modelle
    - **DEA**,  $(\epsilon)$ **NEA**, reguläre Ausdrücke, Typ-3 Grammatiken
    - Modelle sind ineinander umwandelbar – **wähle das passendste**

# ABSCHLUSS UNTER VEREINIGUNG, VERKETTUNG, HÜLLE

## Beweisführung mit regulären Ausdrücken

- $L_1, L_2$  regulär  $\Rightarrow L_1 \cup L_2$  regulär  
 $L_1, L_2$  regulär
  - $\Rightarrow$  Es gibt reguläre Ausdrücke  $E_1, E_2$  mit  $L_1 = L(E_1), L_2 = L(E_2)$
  - $\Rightarrow L_1 \cup L_2 = L(E_1) \cup L(E_2) = L(E_1 + E_2)$  regulär
- $L_1, L_2$  regulär  $\Rightarrow L_1 \circ L_2$  regulär  
 $L_1, L_2$  regulär
  - $\Rightarrow$  Es gibt reguläre Ausdrücke  $E_1, E_2$  mit  $L_1 = L(E_1), L_2 = L(E_2)$
  - $\Rightarrow L_1 \circ L_2 = L(E_1) \circ L(E_2) = L(E_1 \circ E_2)$  regulär
- $L$  regulär  $\Rightarrow L^*$  regulär  
 $L$  regulär
  - $\Rightarrow$  Es gibt einen regulären Ausdruck  $E$  mit  $L = L(E)$
  - $\Rightarrow L^* = (L(E))^* = L(E^*)$  regulär

# ABSCHLUSS UNTER KOMPLEMENTBILDUNG

## Beweisführung mit endlichen Automaten

- **$L$  regulär  $\Rightarrow \overline{L}$  regulär**

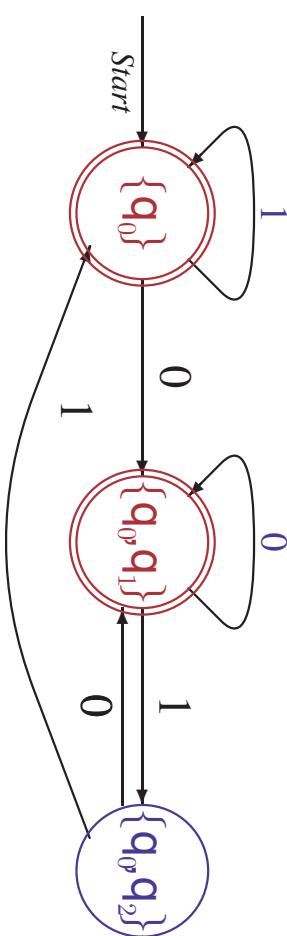
Komplementiere akzeptierende Zustände des erkennenden Automaten

### **$L$ regulär**

$\Rightarrow$  Es gibt einen DEA  $A = (Q, \Sigma, \delta, q_0, F)$  mit  $L = L(A)$   
 $\Rightarrow \overline{L} = \overline{L(A)} = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \notin F\} = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in Q - F\}$   
 $= L((Q, \Sigma, \delta, q_0, Q - F))$  regulär

- **Beispiel: Komplementierung von  $(0+1)^*01$**

- Zugehöriger DEA
- Komplementautomat erkennt Wörter die nicht mit 01 enden
- Regulärer Ausdruck durch Zustandseliminationsverfahren erzeugbar



# ABSCHLUSS UNTER DURCHSCHNITT UND DIFFERENZ

- **Einfache mathematische Beweise**

$L_1, L_2$  regulär  $\Rightarrow \overline{L_1 \cap L_2} = \overline{L_1} \cup \overline{L_2}$  regulär

$L_1, L_2$  regulär  $\Rightarrow L_1 - L_2 = L_1 \cap \overline{L_2}$  regulär

- **Produktkonstruktion auf endlichen Automaten**

Simultane Abarbeitung von Wörtern in beiden Automaten

$L_1, L_2$  regulär

$\Rightarrow$  Es gibt DEAs  $A_1 = (Q_1, \Sigma, \delta_1, q_{0,1}, F_1)$

und  $A_2 = (Q_2, \Sigma, \delta_2, q_{0,2}, F_2)$

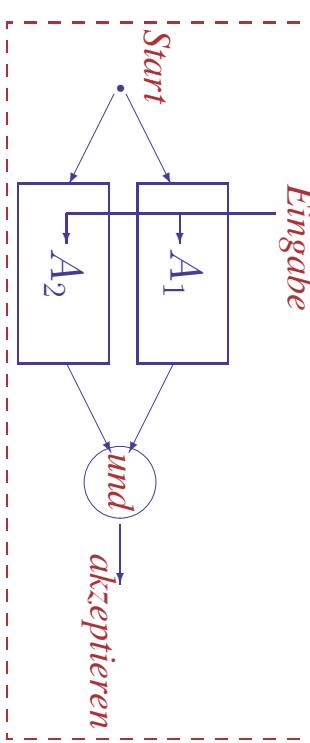
mit  $L_1 = L(A_1), L_2 = L(A_2)$

$\Rightarrow L_1 \cap L_2 = \{w \in \Sigma^* \mid \hat{\delta}_1(q_{0,1}, w) \in F_1 \wedge \hat{\delta}_2(q_{0,2}, w) \in F_2\}$   
 $= \{w \in \Sigma^* \mid (\hat{\delta}_1(q_{0,1}, w), \hat{\delta}_2(q_{0,2}, w)) \in F_1 \times F_2\}$

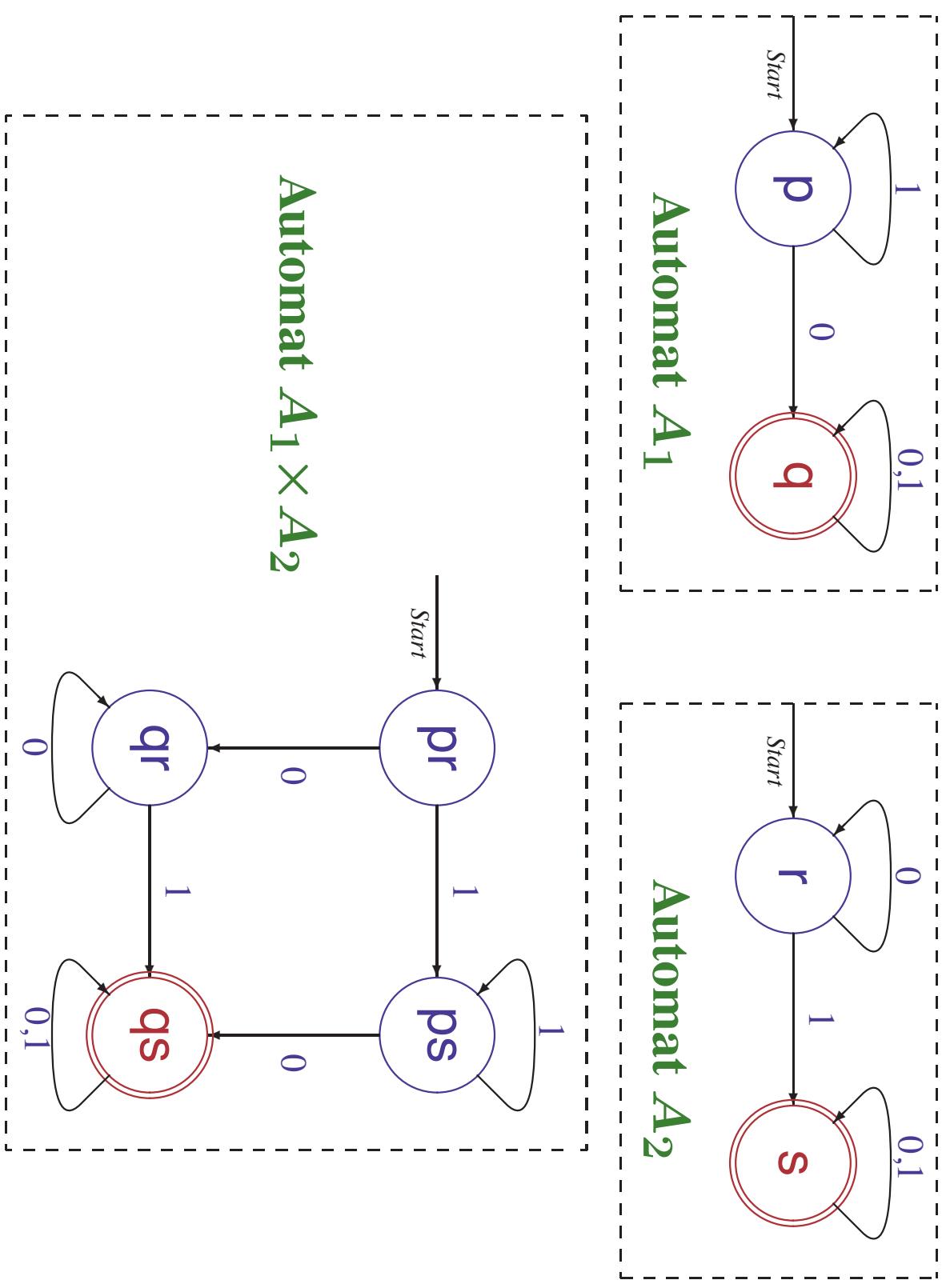
Konstruiere  $A = (Q_1 \times Q_2, \Sigma, \delta, (q_{0,1}, q_{0,2}), F_1 \times F_2)$

mit  $\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$  für  $p \in Q_1, q \in Q_2, a \in \Sigma$

$\Rightarrow L_1 \cap L_2 = L(A)$  regulär



# PRODUKTKONSTRUKTION AM BEISPIEL



## ABSCHLUSS UNTER SPIEGELUNG

$L$  regulär  $\Rightarrow L^R = \{w_n..w_1 \mid w_1..w_n \in L\}$  regulär

- **Beweisführung mit Automaten**

- Bilde Umkehrautomaten zu  $A = (Q, \Sigma, \delta, q_0, F)$  mit  $L = L(A)$ 
  - Umkehrung der Pfeile im Diagramm:  $\delta^R(q, a) = \{q' \mid \delta(q', a) = q\}$
  - $q_0$  wird zum akzeptierenden Zustand:  $F^R = \{q_0\}$
  - Neuer Startzustand  $q_0^R$  mit  $\epsilon$ -Übergängen zu allen  $q \in F$

- **Induktiver Beweis mit regulären Ausdrücken**

- Sei  $L = L(E)$  für einen regulären Ausdruck
  - Für  $E \in \{\emptyset, \epsilon, a\}$  ist  $L^R = L = L(E)$  regulär
  - Für  $E \in \{E_1 + E_2\}$  ist  $L^R = (L(E_1) \cup L(E_2))^R = L(E_1)^R \cup L(E_2)^R$  regulär
  - Für  $E = E_1 \circ E_2$  ist  $L^R = (L(E_1) \circ L(E_2))^R = L(E_2)^R \circ L(E_1)^R$  regulär
  - Für  $E = E_1^*$  ist  $L^R = L(E_1^*)^R = (L(E_1)^R)^*$  regulär
- **Beispiel: Spiegelung von  $L((0+1)^* 0^*)$** 
  - $L^R = L((0^*)^R (0+1)^R) = L((0^R)^* (0^R+1^R)) = L(0^* (0+1))$

# BILD UND URBILD UNTER HOMOMORPHISMEN

- **Homomorphismus**  $h: \Sigma^* \rightarrow \Sigma'^*$

- Funktion, die eindeutig durch Verhalten auf Symbolen definiert ist
- $h: \Sigma^* \rightarrow \Sigma'^*$  ist Homomorphismus, wenn  $h(v_1..v_n) = h(v_1)..h(v_n)$  für  $v_i \in \Sigma$   
Für  $h_1(a) = 01$ ,  $h_1(b) = 0101$  muß  $h_1(aa) = 0101$ ,  $h_1(ab) = 010101$  sein
- Homomorphismen sind mit endlichen (Ein-/Ausgabe) Automaten berechenbar

- **Bild einer Menge  $L$  unter einer Funktion  $h$**

- Menge aller Funktionswerte von  $h$  bei Eingaben aus  $L$
- $h(L) = \{h(w) \mid w \in L\} \subseteq \Sigma'^*$

– z.B.  $h_1(\{ab, aa, b\}) = \{0101, 010101\}$ ,  $h_1(\{a, b\}^*) = \{01\}^*$

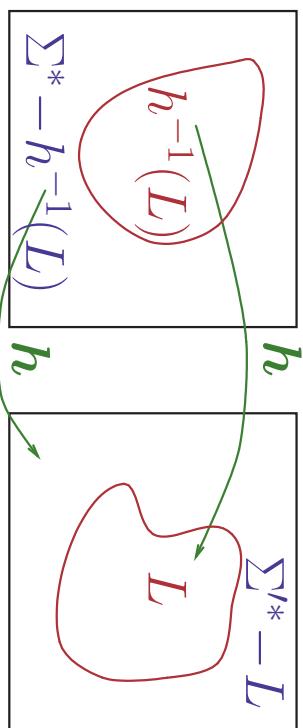
- **Urbild einer Menge  $L$  unter einer beliebigen (!) Funktion  $h$**

- Menge aller Eingaben, deren

Funktionswerte in  $L$  liegen

$$- h^{-1}(L) = \{w \in \Sigma^* \mid h(w) \in L\}$$

- z.B.  $h_1^{-1}(\{0101\}) = \{aa, b\}$ ,
- $h_1^{-1}(\{0101, 11, 1\}) = \{aa, b\}$ ,
- $h_1^{-1}(L((0101)^*)) = \{w \in \{a, b\}^* \mid \#_a(w) \text{ gerade}\}$



# ABSCHLUSSEIGENSCHAFT: BILD UNTER HOMOMORPHISMEN

$L \subseteq \Sigma^*$  regulär,  $h: \Sigma^* \rightarrow \Sigma'^*$  Homomorphismus  $\Rightarrow h(L)$  regulär

## Beweis mit Grammatiken

### $L$ regulär

$\Rightarrow$  Es gibt eine Typ-3 Grammatik  $G = (V, \Sigma, P, S)$  mit  $L = L(G)$   
 $\Rightarrow h(L) = h(L(G)) = \{h(v_1) \dots h(v_n) \in \Sigma'^* \mid S \xrightarrow{*} v_1 \dots v_n\}$

Für  $A \rightarrow a B \in P$  erzeuge Regeln  $A \rightarrow a_1 B_1, B_1 \rightarrow a_2 B_2, \dots, B_{k-1} \rightarrow a_k B$ ,  
wobei  $h(a) = a_1 \dots a_k$  und alle  $B_i$  neue Hilfsvariablen

Sei  $P_h$  die Menge dieser Regeln,  $V_h$  die Menge ihrer Hilfsvariablen

Für  $G_h = (V_h, \Sigma', P_h, S)$  gilt  $A \rightarrow a B \in P \Leftrightarrow A \xrightarrow{*}_{G_h} h(a) B$   
und  $S \xrightarrow{*} G_h v_1 \dots v_n \Leftrightarrow S \xrightarrow{*}_{G_h} h(v_1) \dots h(v_n)$   
 $\Rightarrow h(L) = \{h(v_1) \dots h(v_n) \in \Sigma'^* \mid S \xrightarrow{*} G_h h(v_1) \dots h(v_n)\} = L(G_h)$  **regulär**  
– Sonderbehandlung für  $h(a) = \epsilon$  erforderlich, da Regel  $A \rightarrow B$  unzulässig

---

Beweis mit regulären Ausdrücken in Hopcroft, Motwani, Ullman §4.2.3

# ABSCHLUSS: URBILD UNTER HOMOMORPHISMEN

$L \subseteq \Sigma'^*$  regulär,  $h: \Sigma^* \rightarrow \Sigma'^*$  Homomorphismus  $\Rightarrow h^{-1}(L)$  regulär

## Beweis mit endlichen Automaten

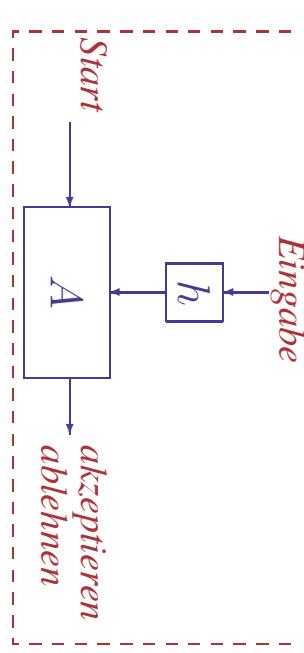
Berechnung von  $h$  vor Abarbeitung der Wörter im Automaten

$L$  regulär

$\Rightarrow$  Es gibt einen DEA  $A = (Q, \Sigma', \delta, q_0, F)$

mit  $L = L(A) = \{v \in \Sigma'^* \mid \hat{\delta}(q_0, v) \in F\}$

$\Rightarrow h^{-1}(L) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, h(w)) \in F\}$



Konstruiere  $A_h = (Q, \Sigma, \delta_h, q_0, F)$  mit  $\delta_h(q, a) = \hat{\delta}(q, h(a))$

Dann gilt  $\hat{\delta}_h(q, w) = \hat{\delta}(q, h(w))$  für alle  $q \in Q$  und  $w \in \Sigma^*$

$\Rightarrow h^{-1}(L) = \{w \in \Sigma^* \mid \hat{\delta}_h(q_0, w) \in F\} = L(A_h)$  regulär

# TESTS FÜR EIGENSCHAFTEN REGULÄRER SPRACHEN

- **Welche Eigenschaften sind automatisch prüfbar?**

- Ist die Sprache eines Automaten leer?
- **Zugehörigkeit:** Ist ein Wort  $w$  Element der Sprache eines Automaten?
- **Äquivalenz:** Beschreiben zwei Automaten dieselbe Sprache?  
Gleiche Fragestellung für Grammatiken und reguläre Ausdrücke

- **Wechsel der Repräsentation ist effektiv**

- NEA  $\rightarrow$  DEA: Teilmengenkonstruktion (exponentielle Aufblähung möglich)
- $\epsilon$ -NEA  $\rightarrow$  DEA: Hüllenbildung + Teilmengenkonstruktion
- DEA  $\rightarrow$   $\epsilon$ -NEA/NEA: Modifikation der Präsentation (Mengenklammern)
- DEA  $\rightarrow$  RA: Zustandselimination (oder  $R_{ij}^k$ -Methode)
- RA  $\rightarrow$   $\epsilon$ -NEA: induktive Konstruktion von Automaten
- DEA  $\rightarrow$  Typ-3 Grammatik: Regeln für Überführungsschritte einführen
- Typ-3 Grammatik  $\rightarrow$  NEA: Überführungstabelle codiert Regeln

**Es reicht, Tests für ein Modell zu beschreiben**

PRÜFE, OB EINE REGULÄRE SPRACHE LEER IST

- Nichttriviales Problem

- Automaten: Gibt es überhaupt einen akzeptierenden Pfad?
  - Reguläre Ausdrücke: Wird mindestens ein einziges Wort charakterisiert?
  - Grammatiken: Wird überhaupt ein Wort aus dem Startzustand erzeugt?

## • Erreichbarkeitstest für DEA $A = (Q, \Sigma, \delta, q_0, F)$

- Wegen  $\delta(q_0, \epsilon) = q_0$  ist  $q_0$  in 0 Schritten erreichbar
  - $q$  in  $k$  Schritten erreichbar,  $\delta(q, a) = q' \Rightarrow q'$  in  $k+1$  Schritten erreichbar
  - $L(A) = \emptyset \Leftrightarrow$  kein  $q \in F$  in  $|Q|$  Schritten erreichbar**

## • Induktive Analyse für reguläre Ausdrücke

- $L(\emptyset) = \emptyset$ ,  $L(\epsilon) \neq \emptyset$ ,  $L(a) \neq \emptyset$
  - $L((E)) = \emptyset \Leftrightarrow L(E) = \emptyset$  keine Änderung
  - $L(E+F) = \emptyset \Leftrightarrow L(E) = \emptyset \wedge L(F) = \emptyset$  Vereinigung von Elementen
  - $L(E \circ F) = \emptyset \Leftrightarrow L(E) = \emptyset \vee L(F) = \emptyset$  Elemente beider Sprachen nötig
  - $L(E^*) \neq \emptyset$ ,  $\epsilon$  gehört immer zu  $L(E^*)$

# TEST AUF ZUGEHÖRIGKEIT (WORTPROBLEM)

- Unterschiedlich schwierig je nach Repräsentation

- Automaten: Gibt es einen akzeptierenden Pfad für das Wort  $w$ ?
- Reguläre Ausdrücke: Wird  $w$  von der Charakterisierung erfasst?
- Grammatiken: Kann  $w$  aus dem Startzustand erzeugt werden?

- Abarbeitung durch DEA  $A = (Q, \Sigma, \delta, q_0, F)$

- Bestimme  $\hat{q} := \hat{\delta}(q_0, w)$  und teste  $\hat{q} \in F$
- Maximal  $|w| + |F|$  Arbeitsschritte

Test für andere Repräsentationen  
durch Umwandlung in DEA

# TEST AUF ÄQUIVALENZ VON SPRACHEN

- **Wann sind zwei reguläre Sprachen gleich?**

- Nichttrivial, da Beschreibungsformen sehr verschieden sein können
  - Verschiedene Automaten, Grammatiken, Ausdrücke, Mischformen, ...

- **Gibt es eine “kanonische” Repräsentation?**

- z.B.
  - Transformiere alles in deterministische endliche Automaten
    - Erzeuge Standardversion mit kleinstmöglicher Anzahl von Zuständen
  - Äquivalenztest prüft dann, ob der gleiche Standardautomat erzeugt wird

- **Wie standardisiert man Automaten?**

- Entferne Zustände, die vom Startzustand unerreichbar sind
- Fasse Zustände zusammen, die für alle Wörter “äquivalent” sind
  - Es führen exakt dieselben Wörter zu akzeptierenden Zuständen
- Ergibt **minimalen äquivalenten Automaten**

# ÄQUIVALENZTEST FÜR ZUSTÄNDE

- **Äquivalenz der Zustände  $p$  und  $q$  ( $\textcolor{red}{p} \cong q$ )**

- Für alle Wörter  $w \in \Sigma^*$  gilt  $\hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$
- Die Wörter müssen nicht zum gleichen Zustand führen

- **Positives Prüfverfahren schwierig**

- Man muss alle Wörter überprüfen, die von einem Zustand ausgehen
- Man kann sich auf Wörter der maximalen Länge  $|Q|$  beschränken
- Besser: Nichtäquivalente (**unterscheidbare**) Zustände identifizieren

- **Methodik: Table-Filling Algorithmus**

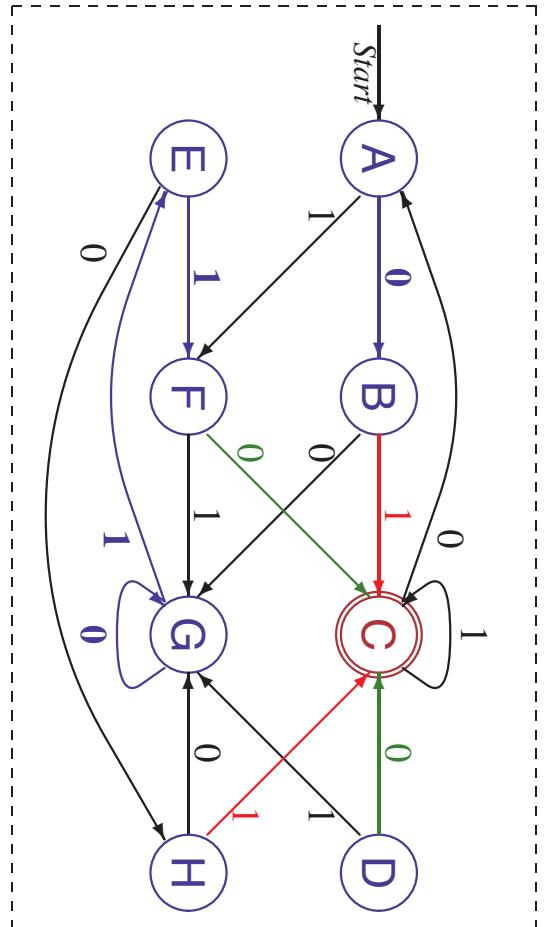
Markiere Unterscheidbarkeit von Zuständen in Tabelle

- Start:  $p \not\cong q$ , falls  $p \in F$  und  $q \notin F$
- Iteration:  $p \not\cong q$ , falls  $\delta(p, a) \not\cong \delta(q, a)$  für ein  $a \in \Sigma$

In jeder Iteration werden nur noch ungeklärte Paare überprüft

Nach maximal  $|Q|$  Iterationen sind alle Unterschiede bestimmt

# ÄQUIVALENZTEST AM BEISPIEL



*Tabelle der Unterschiede*

	A	B	C	D	E	F	G	H
A	■	✗	✗	✗	✗	✗	✗	✗
B	■	✗	✗	✗	✗	✗	✗	✗
C	■	■	✗	✗	✗	✗	✗	✗
D	■	■	✗	✗	✗	✗	✗	✗
E	■	■	■	✗	✗	✗	✗	✗
F	■	■	■	■	✗	✗	✗	✗
G	■	■	■	■	■	✗	✗	✗
H	■	■	■	■	■	■	✗	✗

1. Unterscheide akzeptierende Zustände (**C**) von allen anderen

2a. Eingabesymbol 0: Nur **D** und **F** führen zu akzeptierenden Zuständen

2b. Eingabesymbol 1: Nur **B** und **H** führen zu akzeptierenden Zuständen

3. Überprüfe Nachfolger von  $\{A, E\}$ ,  $\{A, G\}$ ,  $\{B, H\}$ ,  $\{D, F\}$  und  $\{E, G\}$ .

4. Überprüfung von  $\{A, E\}$ ,  $\{B, H\}$  und  $\{D, F\}$  gibt keine Unterschiede

Äquivalenzklassen sind  $\{A, E\}$ ,  $\{B, H\}$ ,  $\{D, F\}$ ,  $\{C\}$  und  $\{G\}$

# ÄQUIVALENZTEST FÜR SPRACHEN

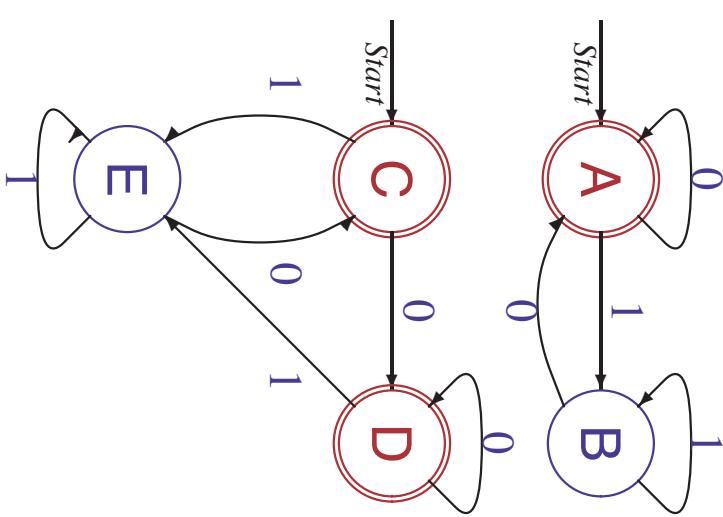
- Prüfverfahren

- Standardisiere Beschreibungsform in zwei disjunkte DEAs  $A_1$  und  $A_2$
- Vereinige Automaten zu  $A = (Q_1 \cup Q_2 \cup \{q'\}, \Sigma, \delta_1 \cup \delta_2, q', F_1 \cup F_2)$
- $A$  enthält  $A_1$  und  $A_2$  als unabhängige Teile
- Bilde Äquivalenzklassen von  $A$

und teste ob  $q_{0,1}$  und  $q_{0,2}$  äquivalent sind

- Zwei DEAs für  $L(\epsilon + (0 + 1)^*0)$

- Äquivalenzklassen sind  $\{A, C, D\}$  (alle Endzustände) und  $\{B, E\}$  (alle Nicht-Endzustände)
- Da  $A$  und  $C$  äquivalent sind, sind die Automaten äquivalent



# MINIMIERUNG ENDLICHER AUTOMATEN

## Konstruiere äquivalenten DEA mit minimaler Menge von Zuständen

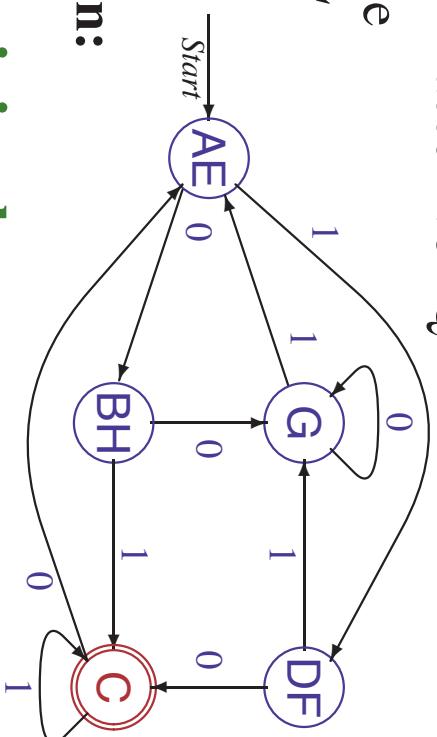
- **Entferne überflüssige Zustände**

- $q$  ist überflüssig, wenn  $\hat{\delta}(q_0, w) \neq q$  für alle Wörter  $w \in \Sigma^*$
- Reduziere  $Q$  zu Menge der erreichbaren Zustände (Verfahren auf Folie 13)

- **Fasse äquivalente Zustände zusammen**

- Bestimme Menge der Äquivalenzklassen von  $Q$
- Setze  $Q'$  als Menge der Äquivalenzklassen von  $Q$

von  $\delta'(q, a)$  für ein beliebiges  $q \in S$   
Wohldefiniert, da alle Nachfolger äquivalenter Zustände äquivalent sind



Anwendung auf Beispielautomaten:

- **Resultierender Automat ist minimal**

- Automaten teilen Sprachen in Äquivalenzklassen

- Wörter, die zum gleichen Zustand führen, sind ununterscheidbar
- Wörter, die zu äquivalenten Zuständen führen, sind ununterscheidbar

Jede Fortsetzung der Wörter führt zum “gleichen” Ergebnis

$\hat{\delta}(q_0, u) \hat{=} \hat{\delta}(q_0, v)$  bedeutet  $\hat{\delta}(q_0, uw) \in F \Leftrightarrow \hat{\delta}(q_0, vw) \in F$  für alle  $w \in \Sigma^*$

- Äquivalenzklassen hängen nur von der Sprache ab

- Für  $L \subseteq \Sigma^*$  definiere Äquivalenzrelation  $\sim_L$  auf  $\Sigma^*$ :
  - $u \sim_L v \equiv uw \in L \Leftrightarrow vw \in L$  gilt für alle  $w \in \Sigma^*$
  - $\sim_L$  ist eine Äquivalenzrelation
  - Die Äquivalenzklasse eines Wortes  $v$  ist  $[v]_L = \{u \in \Sigma^* \mid u \sim_L v\}$
  - Äquivalenzklassen sind disjunkt oder identisch
- Gibt es  $w \in [u]_L \cap [v]_L$  dann ist  $u \sim_L v$ , also  $z \in [u]_L \Leftrightarrow z \in [v]_L$  für alle  $z$
- $\Sigma^*/L$  bezeichnet die Menge der Äquivalenzklassen modulo  $\sim_L$

# ÄQUIVALENZKLASSEN DER SPRACHE $L = \{0^n 1^m \mid n, m \in \mathbb{N}\}$

- $[\epsilon]_L = \{u \in \{0, 1\}^* \mid u \sim_L \epsilon\} = \{u \mid \forall w. uw \in L \Leftrightarrow w \in L\} = \{0^k \mid k \in \mathbb{N}\}$
- $[0]_L = [00]_L = [000]_L = \dots = [\epsilon]_L$ , weil  $0 \in [\epsilon]_L$ ,  $00 \in [\epsilon]_L$ ,  $000 \in [\epsilon]_L$ , ...  
 $[1]_L = \{u \mid \forall w. uw \in L \Leftrightarrow 1w \in L\} = \{u \mid \forall w. uw \in L \Leftrightarrow \exists j. w = 1^j\}$   
=  $\{0^k 1^i \mid k \in \mathbb{N}, i > 0\}$
- $[01]_L = [1]_L$ , weil  $01 \in [1]_L$
- $[10]_L = \{u \mid \forall w. uw \in L \Leftrightarrow 10w \in L\} = \{u \mid \forall w. uw \notin L\} = \{0, 1\}^* - L$   
Grund: für  $L = \{0^n 1^m \mid n, m \in \mathbb{N}\}$  gilt  $u \notin L \Rightarrow \forall w. uw \notin L$  (Umkehrung gilt immer)
- Wegen  $[\epsilon]_L \cup [1]_L = L$  folgt:  $\{0, 1\}^*/L = \{[\epsilon]_L, [1]_L, [10]_L\}$

**Reguläre Sprachen haben nur endlich viele Äquivalenzklassen**

# ÄQUIVALENZKLASSEN DER SPRACHE $L = \{0^n 1^n \mid n \in \mathbb{N}\}$

- $[\epsilon]_L = \{u \mid \forall w. u w \in L \Leftrightarrow w \in L\} = \{\epsilon\}$
- $[0]_L = \{u \mid \forall w. u w \in L \Leftrightarrow 0w \in L\} = \{u \mid \forall w. u w \in L \Leftrightarrow \exists n. w = 0^n 1^{n+1}\} = \{0\}$
- $[1]_L = \{u \mid \forall w. u w \in L \Leftrightarrow 1w \in L\} = \{u \mid \forall w. u w \notin L\} = \{0, 1\}^* - \{0^n 1^m \mid n \geq m\}$
- $[00]_L = \dots = \{u \mid \forall w. u w \in L \Leftrightarrow \exists n. w = 0^n 1^{n+2}\} = \{00\}$
- $[01]_L = \dots = \{u \mid \forall w. u w \in L \Leftrightarrow w = \epsilon\} = L - \{\epsilon\}$
- $[10]_L = [11]_L = [1]_L$ , weil  $10 \in [1]_L$ ,  $11 \in [1]_L$
- $[000]_L = \dots = \{u \mid \forall w. u w \in L \Leftrightarrow \exists n. w = 0^n 1^{n+3}\} = \{000\}$
- **Es gibt unendlich viele Klassen in  $\{0, 1\}^*/L$** 
  - z.B. sind alle Klassen  $[0^k]_L$  ( $= \{0^k\}$ ) verschieden

Für den Beweis dieser Aussage muß man die Klassen nicht exakt bestimmen. Es reicht:

„Für  $k \neq j$  und  $w = 1^k$  ist  $0^k w \in L$ , aber  $0^j w \notin L$ , also  $0^k \not\in_L 0^j$  also  $[0^k]_L \neq [0^j]_L$ “

## Nichtreguläre Sprachen haben unendlich viele Äquivalenzklassen

# DER SATZ VON MYHILL/NERODE

## Eine Sprache $L$ ist regulär, g.d.w $\Sigma^*/L$ endlich ist

### Beweis

$\Rightarrow$  : Es sei  $L$  eine reguläre Sprache

Dann gibt es einen minimalen DEA  $A = (Q, \Sigma, \delta, q_0, F)$  mit  $L = L(A)$

Da  $A$  minimal ist, gilt für beliebige Wörter  $u, v \in \Sigma^*$

$\hat{\delta}(q_0, u) = \hat{\delta}(q_0, v) \Leftrightarrow (\hat{\delta}(q_0, uw) \in F \Leftrightarrow \hat{\delta}(q_0, vw) \in F)$  für alle  $w \in \Sigma^*$

$\Leftrightarrow (u w \in L \Leftrightarrow v w \in L)$  für alle  $w \in \Sigma^* \Leftrightarrow u \sim_L v$

Damit ist  $|\Sigma^*/L|$  (der Index von  $L$ ) gleich der Anzahl der Zustände in  $A$

$\Leftarrow$  : Es sei  $\Sigma^*/L$  endlich.

Konstruiere einen DEA  $A = (\Sigma^*/L, \Sigma, \delta, [\epsilon]_L, F)$

mit  $\delta([u]_L, a) = [ua]_L$  für alle  $a \in \Sigma$  und  $F = \{[v]_L \mid v \in L\}$

$\delta$  ist wohldefiniert, weil  $ua \sim_L va$  für alle  $a \in \Sigma$  gilt, wenn  $u \sim_L v$

und es gilt  $w \in L(A) \Leftrightarrow \hat{\delta}([\epsilon]_L, w) \in F \Leftrightarrow [w]_L \in F \Leftrightarrow w \in L$

## Wie zeigt man, dass eine Sprache $L$ nicht regulär ist?

- **Direkter Nachweis**

- Zeige, dass kein endlicher Automat genau die Wörter von  $L$  erkennt
- Sprache muss unendlich sein und komplizierte Struktur haben  
(Anzahl der Äquivalenzklassen muss unendlich sein)
- Technisches Hilfsmittel: Pumping Lemma

- **Verwendung der Abschlusseigenschaften**

- Zeige, dass Regularität von  $L$  dazu führen würde, dass eine als nichtregulär bekannte Sprache regulär sein müsste
- Häufige Technik: (Ur-)bild unter Homomorphismen

# DAS PUMPING LEMMA FÜR REGULÄRE SPRACHEN

- Warum ist  $\{0^n 1^n \mid n \in \mathbb{N}\}$  nicht regulär?

- Ein DEA muss alle Nullen beim Abarbeiten zählen und dann vergleichen
- Für  $n > |Q|$  muss ein Zustand von  $A$  doppelt benutzt worden sein
- Eine  $\delta$ -Schleife mit  $k$  Zuständen bedeutet, dass  $A$  auch  $0^{n+k} 1^n$  akzeptiert

- Allgemeine Version: **Pumping Lemma**

Für jede reguläre Sprache  $L \in \mathcal{L}_3$  gibt es eine Zahl  $n \in \mathbb{N}$ , so dass jedes Wort  $w \in L$  mit Länge  $|w| \geq n$  zerlegt werden kann in  $w = xyz$  mit den Eigenschaften

- (1)  $y \neq \epsilon$ ,
- (2)  $|xy| \leq n$  und
- (3) für alle  $k \in \mathbb{N}$  ist  $xyz^k \in L$

- Aussage ist wechselseitig konstruktiv

- Die Zahl  $n$  kann zu jeder regulären Sprache  $L$  bestimmt werden
- Die Zerlegung  $w = xyz$  kann zu jedem Wort  $w \in L$  bestimmt werden

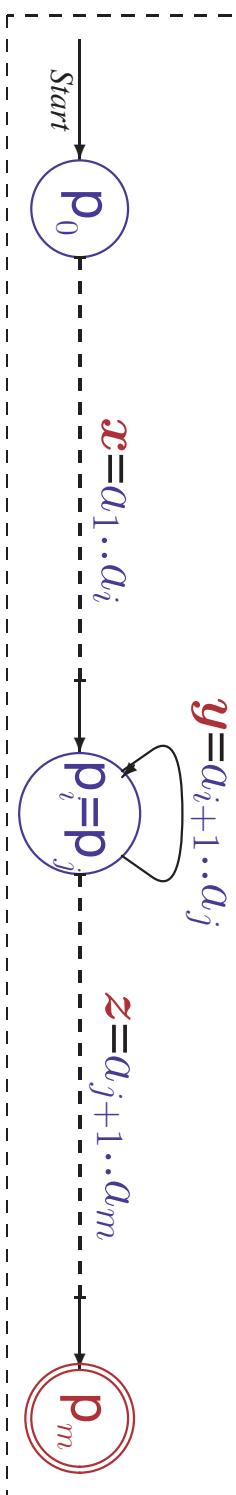
## BEWEIS DES PUMPING LEMMAS

Für jede Sprache  $L \in \mathcal{L}_3$  gibt es ein  $n \in \mathbb{N}$ , so dass jedes  $w \in L$  mit  $|w| \geq n$  zerlegbar ist in  $w = xyz$  mit den Eigenschaften

(1)  $y \neq \epsilon$ , (2)  $|xy| \leq n$  und (3) für alle  $k \in \mathbb{N}$  ist  $xyz^k \in L$

### • Beweis mit Automaten

- Sei  $L$  regulär und  $A = (Q, \Sigma, \delta, q_0, F)$  ein DEA mit  $L = L(A)$
- Wähle  $n = |Q|$ . Betrachte  $w = a_1..a_m$  mit  $|w| \geq n$  und  $p_i := \hat{\delta}(q_0, a_1..a_i)$
- Dann gibt es  $i, j$  mit  $0 \leq i < j \leq n$  und  $p_i = p_j$  (Schubfachprinzip)
- Zerlege  $w$  in  $w = xyz$  mit  $x = a_1..a_i$ ,  $y = a_{i+1}..a_j$  und  $z = a_{j+1}..a_m$



- Per Konstruktion gilt  $y \neq \epsilon$ ,  $|xy| \leq n$  und  $\hat{\delta}(p_i, y^k) = p_i$  für alle  $k \in \mathbb{N}$
- Also  $\hat{\delta}(q_0, xyz) = \hat{\delta}(p_i, y^k z) = \hat{\delta}(p_i, yz) = \hat{\delta}(q_0, xy) = \hat{\delta}(q_0, w) \in F$

## ANWENDUNGEN DES PUMPING LEMMAS

$L_1 = \{0^m 1^m \mid m \in \mathbb{N}\}$  ist nicht regulär

### • Verwende Kontraposition des Pumping Lemmas

Eine Sprache  $L$  ist nicht regulär, wenn es kein  $n \in \mathbb{N}$  gibt, so dass jedes  $w \in L$  mit  $|w| \geq n$  zerlegbar ist in  $w = xyz$  mit den Eigenschaften

- (1)  $y \neq \epsilon$ , (2)  $|xy| \leq n$  und (3) für alle  $k \in \mathbb{N}$  ist  $xy^k z \in L$

### Umformulierung: Ziehe Negation in die Bedingungen hinein

$L$  ist nicht regulär, wenn es für jedes  $n \in \mathbb{N}$  ein  $w \in L$  mit  $|w| \geq n$  gibt so dass für jede Zerlegung  $w = xyz$  mit den Eigenschaften

- (1)  $y \neq \epsilon$  und (2)  $|xy| \leq n$  ein  $k \in \mathbb{N}$  existiert mit  $xy^k z \notin L$

### • Kontrapositionsbeweis für $L_1 \notin \mathcal{L}_3$

- Sei  $n \in \mathbb{N}$  beliebig. Wir wählen  $w = 0^m 1^m$  für ein  $m > n$
- Sei  $w = xyz$  eine beliebige Zerlegung mit  $y \neq \epsilon$  und  $|xy| \leq n$ .
  - Dann gilt  $x = 0^i$ ,  $y = 0^j$ ,  $z = 0^{m-i-j} 1^m$  für ein  $j \neq 0$  und  $i+j \leq n$ .
  - Wir wählen  $k=0$ . Dann ist  $xy^0 z = 0^{m-j} 1^m \notin L_1$

**Aufgrund des Pumping Lemmas kann  $L_1$  also nicht regulär sein.**

## ANWENDUNGEN DES PUMPING LEMMAS II

$$L_2 = \{w \in \{1\}^* \mid |w| \text{ ist Primzahl}\} \notin \mathcal{L}_3$$

- **Beweis folgt dem gleichen Schema**

- Sei  $n \in \mathbb{N}$  beliebig.
- Wir wählen  $w = 1^p$  für eine Primzahl  $p > n + 1$
- Sei  $w = xyz$  eine beliebige Zerlegung mit  $y \neq \epsilon$  und  $|xy| \leq n$   
Dann gilt  $x=1^i$ ,  $y=1^j$   $z=1^{p-i-j}$  für ein  $j \neq 0$  und  $i+j \leq n$ .
- Wir wählen  $k=p-j$ .  
Dann ist  $xyz^k = 1^i 1^j (p-j) 1^{p-i-j} = 1^{i+j(p-j)+p-i-j} = 1^{(j+1)(p-j)} \notin L_2$

**Aufgrund des Pumping Lemmas kann  $L_2$  also nicht regulär sein.**

# NACHWEIS VON $L \notin \mathcal{L}_3$ MIT ABSCHLUSSEIGENSCHAFTEN

- Anwendung des Pumping Lemmas ist oft mühsam

- Beweis für  $L_3 = \{(m)^m \mid m \in \mathbb{N}\} \notin \mathcal{L}_3$  identisch mit dem von  $L_1$
- Beweis für  $L_4 = \{w \in \{0, 1\}^* \mid \#_0(w) = \#_1(w)\} \notin \mathcal{L}_3$  ähnlich  
 $(\#_1(w)$  ist die Anzahl der Einsen in  $w$ )

- Verwende Umkehrung der Abschlusseigenschaften

$$\begin{array}{ll} \overline{L} \notin \mathcal{L}_3 \Rightarrow L \notin \mathcal{L}_3 & h^R(L) \notin \mathcal{L}_3 \Rightarrow L \notin \mathcal{L}_3 \\ h(L) \notin \mathcal{L}_3 \Rightarrow L \notin \mathcal{L}_3 & h^{-1}(L) \notin \mathcal{L}_3 \Rightarrow L \notin \mathcal{L}_3 \\ L \cup L' \notin \mathcal{L}_3 \wedge L' \in \mathcal{L}_3 \Rightarrow L \notin \mathcal{L}_3 & L \cap L' \notin \mathcal{L}_3 \wedge L' \in \mathcal{L}_3 \Rightarrow L \notin \mathcal{L}_3 \\ L \circ L' \notin \mathcal{L}_3 \wedge L' \in \mathcal{L}_3 \Rightarrow L \notin \mathcal{L}_3 & L' \circ L \notin \mathcal{L}_3 \wedge L' \in \mathcal{L}_3 \Rightarrow L \notin \mathcal{L}_3 \\ \vdots & \vdots \end{array}$$

- Anwendungsbeispiele

$L_3 \notin \mathcal{L}_3$ : Wähle Homomorphismus  $h: \{((), )\} \rightarrow \{0, 1\}$  mit  $h(()) = 0, h(()) = 1$

Dann ist  $h(L_3) = \{0^m 1^m \mid m \in \mathbb{N}\} = L_1 \notin \mathcal{L}_3$

$L_4 \notin \mathcal{L}_3$ : Es gilt  $L_4 \cap L(0^* \circ 1^*) = L_1 \notin \mathcal{L}_3$

DEAs können korrekte Klammerausdrücke nicht erkennen!

# EIGENSCHAFTEN REGULÄRER SPRACHEN IM RÜCKBLICK

- **Abschlusseigenschaften**

- Operationen  $\cup, \cap, -, ^R, \circ, *, h, h^{-1}$  erhalten Regularität von Sprachen
- Verwendbar zum Nachweis von Regularität oder zur Widerlegung

- **Automatische Prüfungen**

- Man kann testen ob eine reguläre Sprache leer ist
- Man kann testen ob ein Wort zu einer regulären Sprache gehört
- Man kann testen ob zwei reguläre Sprachen gleich sind

- **Minimierung von Automaten**

- Ein Automat kann minimiert werden, indem man äquivalente Zustände zusammenlegt und unerreichbare Zustände entfernt

- **Pumping Lemma**

- Wiederholt man einen bestimmten Teil ausreichend großer Wörter einer regulären Sprache beliebig oft, so erhält man immer ein Wort der Sprache
- Verwendbar zur Widerlegung von Regularität

# ZUSAMMENFASSUNG: REGULÄRE SPRACHEN

- **Drei Modelle**

- Endliche Automaten (DEA, NEA,  $\epsilon$ -NEA) erkennen Wörter einer Sprache
- Reguläre Ausdrücke beschreiben Struktur der Wörter
- (Typ 3) Grammatiken erzeugen Wörter einer regulären Sprache

- **Alle drei Modelle sind äquivalent**

- $\epsilon$ -NEA  $\rightarrow$  DEA: Teilmengenkonstruktion
- DEA  $\rightarrow$  Typ-3 Grammatik: Verwandle Überführungsfunktion in Regeln
- Typ-3 Grammatik  $\rightarrow$  NEA: Verwandle Regeln in Überführungsfunktion
- DEA  $\rightarrow$  Reguläre Ausdrücke: Erzeuge Ausdrücke für Verarbeitungspfade oder eliminiere Zustände in RA Automaten
- Reguläre Ausdrücke  $\rightarrow$  NEA: Iterative Konstruktion von Automaten

- **Wichtige Eigenschaften von  $\mathcal{L}_3$**

- Abgeschlossen unter  $\cup$ ,  $\cap$ ,  $-$ ,  ${}^R$ ,  $\circ$ ,  $*$ ,  $h$ ,  $h^{-1}$
- Entscheidbarkeit des Wortproblems und Gleichheit von Sprachen
- Endliche Automaten können automatisch **minimiert** werden
- Nachweis der Nichtregularität von Sprachen mit dem Pumping Lemma

# Theoretische Informatik I

## Einheit 3

### Kontextfreie Sprachen



1. Kontextfreie Grammatiken
2. Pushdown Automaten
3. Eigenschaften kontextfreier Sprachen



# VERARBEITUNG VON PROGRAMMIERSPRACHEN

- **Was ist das einfachste Beschreibungsmodell?**

- Analyse und Compilation muss formal beschreibbar sein
- Generisches Modell für “alle” Programmiersprachen erforderlich

- **Typ-3 Sprachen sind einfach und effizient**

- Beschreibung durch Grammatiken oder reguläre Ausdrücke
- Beschreibung umwandelbar in endlichen Automaten
- Erkennung von Wörtern der Sprache in “Echtzeit”

- **Aber Programmiersprachen sind nicht regulär**

- Die meisten Programmstrukturen enthalten Schachtelungen wie Blöcke, if-then-else, arithmetische Ausdrücke, Klammerausdrücke, ...
- Korrekte Klammerausdrücke und Schachtelungen sind nicht regulär



**Syntaxanalyse und Compilation von Programmiersprachen braucht mehr als reguläre Sprachen**

# ALLE BEDEUTENDEN SPRACHEN SIND KONTEXTFREI

- **Programmiersprachen**

- Compiler kann kontextfreie Grammatiken effizient verarbeiten
- Parser kann aus kontextfreier Grammatik automatisch erzeugt werden
  - Standard Unix tool **YACC** unterstützt schnellen Compilerentwurf

- **Markup Sprachen**

- **HTML**: Formatierung von Dokumenten mit Links zu Programmaufrufen
- **XML**: Einheitliche Beschreibung der Semantik von Dokumenten

Beide Sprachen erfordern die Mächtigkeit kontextfreier Grammatiken

Mehr in HMU §5.3

- **Wichtige Fragen**

- **Analyse**: Systematische Rekonstruktion des Syntaxbaums
  - Ist das immer eindeutig möglich?
- **Compilation**: Zuweisung von Objectcode an Wörter der Sprache
- **Maschinenmodell**: effektive generische Erkennungsverfahren

# Theoretische Informatik I

## Einheit 3.1

### Kontextfreie Grammatiken



1. Grammatiken und Ableitungen
2. Ableitungsbäume
3. Mehrdeutigkeiten

## RÜCKBLICK: KONTEXTFREIE GRAMMATIKEN

- Eine **kontextfreie Grammatik (kfg)** ist ein **4-Tupel  $G = (V, T, P, S)$**  mit
  - $T$  endliches **Terminalalphabet**
  - $V$  endliches **Hilfsalphabet** mit  $V \cap T = \emptyset$
  - $P \subseteq V \times \Gamma^*$  endliche Menge der **Produktionen** (wobei  $\Gamma = V \cup T$ )
  - $S \in V$  **Startsymbol**
- Die übliche Schreibweise für Produktionen  $(A, r) \in P$  ist  $A \rightarrow^r$   
Eine kompakte Notation für  $A \rightarrow^{r_1}, A \rightarrow^{r_2} \dots A \rightarrow^{r_n}$  ist  $A \rightarrow^{r_1} | r_2 | \dots | r_n$
- **Ableitbarkeit in einer kontextfreien Grammatik**
  - $w \rightarrow z \equiv \exists x, y \in \Gamma^*. \exists A \rightarrow r \in P. w = x A y \wedge z = x r y$
  - $w \xrightarrow{*} z \equiv \exists n \in \mathbb{N}. w \xrightarrow{n} z$   
wobei  $w \xrightarrow{0} z \equiv w = z$  und  $w \xrightarrow{n+1} z \equiv \exists u \in \Gamma^*. w \xrightarrow{} u \wedge u \xrightarrow{n} z$
- **Von  $G$  erzeugte Sprache:**

$$L(G) \equiv \{w \in T^* \mid S \xrightarrow{*} w\}$$

# GRAMMATIK FÜR GESCHÄCHTELTE KLAMMERAUSDRÜCKE

$$G_5 = (\{S\}, \{(, )\}, \{S \rightarrow (S), S \rightarrow \epsilon\}, S)$$

Zeige:  $L(G_5) = \{(^k)^k \mid k \in \mathbb{N}\}$

- Beweise durch Induktion über Länge der Ableitung

$$-\forall k \in \mathbb{N}. \forall w \in \{(, )\}^*. S \xrightarrow{k+1} w \Leftrightarrow w = (^k)^k$$

- Basisfall

$$- S \xrightarrow{1} w \Leftrightarrow (S \rightarrow w) \in P \Leftrightarrow w = \epsilon \Leftrightarrow w = (^0)^0 \quad \checkmark$$

- Induktionsschritt

$$\begin{aligned} - & \text{Es gelte } \forall v \in \{(, )\}^*. S \xrightarrow{k+1} v \Leftrightarrow v = (^k)^k \\ - & S \xrightarrow{k+2} w \Leftrightarrow S \rightarrow (S) \xrightarrow{k+1} w \\ & \Leftrightarrow \exists v \in \{(, )\}^*. S \xrightarrow{k+1} v \wedge w = (v) \\ & \Leftrightarrow \exists v \in \{(, )\}^*. v = (^k)^k \wedge w = (v) \quad (\text{Annahme}) \\ & \Leftrightarrow w = (^{k+1})^{k+1} \end{aligned}$$

✓

$$\{(^k)^k \mid k \in \mathbb{N}\} \in \mathcal{L}_2 - \mathcal{L}_3$$

# KONTEXTFREIE GRAMMATIK FÜR PALINDROME

Wähle  $G_6 = (\{S\}, \{0, 1\}, P, S)$

mit  $P = \{S \rightarrow \epsilon, S \rightarrow 0, S \rightarrow 1, S \rightarrow 0S0, S \rightarrow 1S1\}$

Zeige:  $L(G_6) = \{w \in \{0, 1\}^* \mid w = w^R\}$

- Beweise durch Induktion über Länge der Ableitung

–  $\forall k \in \mathbb{N}. \forall w \in \{0, 1\}^*. S \xrightarrow{k+1} w \Leftrightarrow w = w^R \wedge |w| \in \{2k, 2k+1\}$

- Basisfall

–  $S \xrightarrow{1} w \Leftrightarrow (S \rightarrow w) \in P \Leftrightarrow w \in \{0, 1, \epsilon\} \Leftrightarrow w = w^R \wedge |w| \in \{0, 1\}$  ✓

- Induktionssschritt

– Es gelte  $\forall v \in \{0, 1\}^*. S \xrightarrow{k+1} v \Leftrightarrow v = v^R \wedge |v| \in \{2k, 2k+1\}$

–  $S \xrightarrow{k+2} w \Leftrightarrow S \xrightarrow{0} S \xrightarrow{k+1} w \vee S \xrightarrow{1} S \xrightarrow{k+1} w$

$\Leftrightarrow \exists v \in \{0, 1\}^*. S \xrightarrow{k+1} v \wedge (w = 0v0 \vee w = 1v1)$

$\Leftrightarrow \exists v \in \{0, 1\}^*. v = v^R \wedge |v| \in \{2k, 2k+1\} \wedge (w = 0v0 \vee w = 1v1)$

$\Leftrightarrow w = w^R \wedge |w| \in \{2k+2, 2k+3\}$

✓

# GRAMMATIK FÜR ARITHMETISCHE AUSDRÜCKE

- Ausdrücke über Operatoren + und \*

- Bezeichner (*I*entifier):

- Buchstabe gefolgt von Buchstaben/Ziffern

- Buchstaben **a**, **b**, **c**, Ziffern **0**, **1**

- Ausdrücke (*E*xpressions):

- Schachtelung mit +, \* und Klammern

- $G_7 = (\{E, I\}, \{a, b, c, 0, 1, +, *, (), \}, P, E)$

mit  $P = \{ E \rightarrow I \mid E+E \mid E*E \mid (E) \}$

$$I \rightarrow a \mid b \mid c \mid Ia \mid Ib \mid Ic \mid I0 \mid I1 \}$$

Kann man die Struktur eines arithmetischen Ausdrucks mit  $G_7$  immer rekonstruieren?

# LINKS- UND RECHTSSEITIGE ABLEITUNGEN

## Rekonstruierbare Auswahl von Produktionen

- Beliebige Ableitung

$$\begin{aligned} E &\longrightarrow E*E \longrightarrow I*E \longrightarrow I*(E) \\ &\longrightarrow I*(E+E) \longrightarrow I*(I+E) \longrightarrow I*(I+I) \longrightarrow I*(a+I) \\ &\longrightarrow I*(a+I0) \longrightarrow I*(a+I00) \longrightarrow I*(a+b00) \longrightarrow a*(a+b00) \end{aligned}$$

- Linksseitige Ableitung  $w \xrightarrow{L} z$

– In  $w$  wird die am weitesten links stehende Variable ersetzt

$$\begin{aligned} E &\xrightarrow{L} E*E \xrightarrow{L} I*E \xrightarrow{L} a*E \xrightarrow{L} a*(E) \\ &\longrightarrow a*(E+E) \xrightarrow{L} a*(I+E) \xrightarrow{L} a*(a+E) \xrightarrow{L} a*(a+I) \\ &\longrightarrow a*(a+I0) \xrightarrow{L} a*(a+I00) \xrightarrow{L} a*(a+b00) \end{aligned}$$

- Rechtsseitige Ableitung  $w \xrightarrow{R} z$

– In  $w$  wird die am weitesten rechts stehende Variable ersetzt

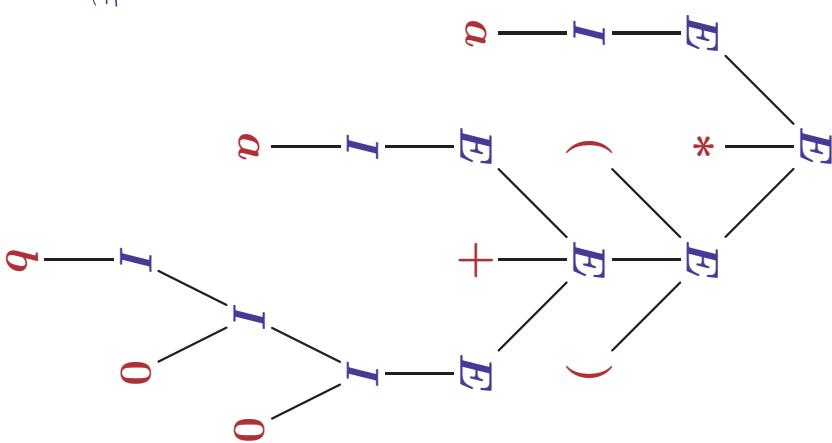
$$\begin{aligned} E &\xrightarrow{R} E*E \xrightarrow{R} E*(E) \xrightarrow{R} E*(E+E) \xrightarrow{R} E*(E+I) \\ &\longrightarrow E*(E+I0) \xrightarrow{R} E*(E+I00) \xrightarrow{R} E*(E+b00) \\ &\longrightarrow E*(E+b00) \xrightarrow{R} E*(a+b00) \xrightarrow{R} I*(a+b00) \xrightarrow{R} a*(a+b00) \end{aligned}$$

# ABLEITUNGSBÄUME (PARSEBÄUME)

## Baumdarstellung von Ableitungen

- **Exkurs: Notation für (geordnete) Bäume**

- Baum: Sammlung von Knoten mit Nachfolgerrelation
  - Nachfolger sind geordnet
  - Ein Knoten hat maximal einen Vorgänger
  - Wurzel: Knoten ohne Vorgänger
  - Blatt / Innerer Knoten: Knoten ohne/mit Nachfolger
  - Nachkommen: transitive Hülle der Nachfolgerrelation
- **Ableitungsbäume sind markierte Bäume**
- Innere Knoten mit Variablen  $A \in V$  markiert
  - Wurzel markiert mit Startsymbol
  - Blätter markiert mit Terminalsymbolen  $a \in T$  oder mit  $\epsilon$
  - Hat ein innerer Knoten Markierung  $A$  und Nachfolger mit Markierungen  $v_1 \dots v_n$ , so ist  $A \rightarrow v_1 \dots v_n \in P$



# ABLEITUNGSBÄUME REPRÄSENTIEREN ABLEITUNGEN

- Blätter repräsentieren Terminalwörter

- Auslesen durch Tiefensuche von links nach rechts
  - $a * (a + b00)$

- Baum repräsentiert Ableitungen

- ## – Rekursive Erzeugung beginnend mit Wurzel

$$\begin{aligned}
 E &\xrightarrow{L} E * E \xrightarrow{L} I * E \xrightarrow{L} a * E \xrightarrow{L} a * (E) \\
 &\xrightarrow{L} a * (E + E) \xrightarrow{L} a * (I + E) \xrightarrow{L} a * (a + E) \\
 &\xrightarrow{L} a * (a + I) \xrightarrow{L} a * (a + I 0) \xrightarrow{L} a * (a + I 00) \\
 &\xrightarrow{L} a * (a + b 00)
 \end{aligned}$$

– Vorrang für tiefe rechte Knoten ergibt Rechtsableitung

$S \xrightarrow{*} w \Leftrightarrow$  es gibt einen Ableitungsbaum mit Blatmarkierung  $w$

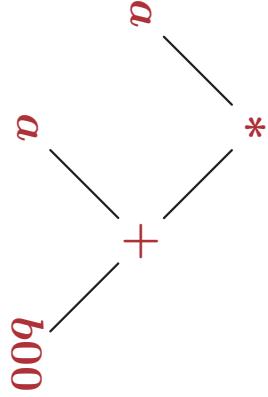
$\Leftarrow$ : Konstruiere Baum induktiv aus Linkstableitung von  $w$  induktiv aus Baum

Details in HMU §5.2

# SYNTAXANALYSE UND COMPIRATION

- **Parser erzeugen Syntaxbaum**

- Rekonstruktion des Ableitungsbäumes aus dem Wort  $a \quad *$
- Entferne Blätter mit Klammern
- Gruppiere Identifier zu lexikalischen Einheiten
- Entferne Vorgänger von Identifern, die mit  $E$  markiert sind
- Entferne  $E$ -Vorgänger von  $+/*$



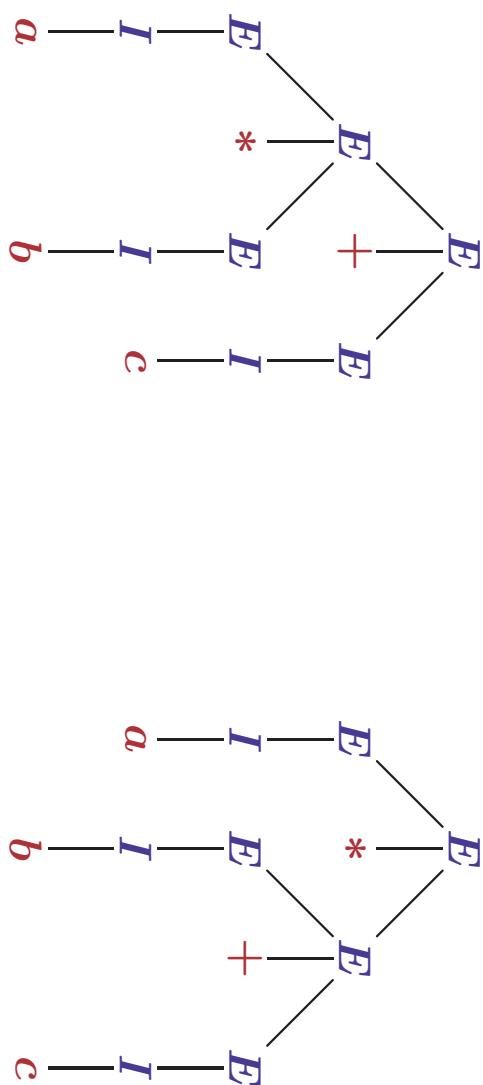
- **Compiler erzeugt Maschinencode**

- Übersetze Identifier in Registernamen
- $+ (x, y)$ :
  - Bestimme Wert von  $y$  und lege ihn im Register ab
  - Bestimme Wert von  $x$
  - Addiere Registerwert
- Coderzeugung für  $*(x, y)$  analog

Details in Vossen/Witt §7.2

**Nur möglich, wenn Ableitungsbau eindeutig ist**

## WANN IST DER ABLEITUNGSBÄUM EINDEUTIG?



- Das Wort  $a * b + c$  hat zwei Ableitungen in  $G_7$

$E \rightarrow E+E \rightarrow E*E+E \rightarrow I*E+E \rightarrow a*E+E \rightarrow a*I+E$   
     $\longrightarrow a*b+E \rightarrow a*b+I \rightarrow a*b+c$

$E \rightarrow E*E \rightarrow I*E \rightarrow a*E \rightarrow a*E+E \rightarrow a*I+E$   
     $\longrightarrow a*b+E \rightarrow a*b+I \rightarrow a*b+c$

Beide Ableitungen sind Linksableitungen

- Die Grammatik  $G_7$  ist mehrdeutig

– Wörter der Sprache können nicht eindeutig analysiert werden

## MEHRDEUTIGKEIT

- **Eindeutige Grammatik**  $G = (V, T, P, S)$

- Jedes Wort  $w \in L(G)$  hat genau einen Ableitungsbau
- Andernfalls ist  $G$  **mehrdeutig**
  - (ein  $w \in L(G)$  hat mindestens zwei verschiedene Ableitungsbäume)
  - $G_7$  ist mehrdeutig

- **Eindeutige Sprache**  $L$

- Es gibt eine eindeutige Grammatik  $G$  mit  $L = L(G)$
- Die Sprache von  $G_7$  ist eindeutig
  - nächste Folie

- **Inhärent mehrdeutige Sprache**  $L$

- Eine eindeutige Grammatik für  $L$  kann nicht angegeben werden
  - $\{0^i 1^j 2^k \mid i=j \vee j=k\}$  ist inhärent mehrdeutig
    - Beweisskizze folgt

**Programmiersprachen müssen eindeutig sein**

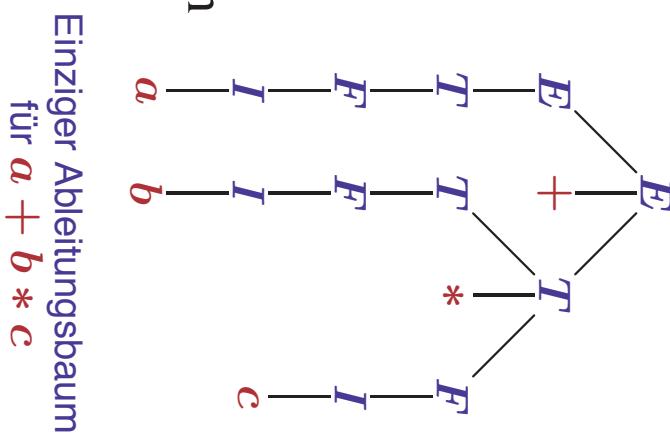
# AUFLÖSUNG VON MEHRDEUTIGKEITEN

- Was fehlt bei  $G_7 = (\{E, I\}, \{\textcolor{red}{a}, \textcolor{blue}{b}, \textcolor{green}{c}, \textcolor{violet}{0}, \textcolor{blue}{1}, +, *, (), \}, P, E)$   
mit  $P = \{E \rightarrow I \mid E+E \mid E*E \mid (E),$   
 $I \rightarrow \textcolor{red}{a} \mid \textcolor{blue}{b} \mid \textcolor{green}{c} \mid Ia \mid Ib \mid Ic \mid I0 \mid I1\}?$   
 $G_7$  beinhaltet nicht die üblichen Konventionen für  $*$  und  $+$ 
    - \* bindet stärker als +
    - \* und + werden als linkssassoziativ angesehenAlle anderen Lesarten benötigen Klammern
  - Prioritätsregeln können Eindeutigkeit erzeugen
    - Niedrigste Priorität + steht linkssassoziativ und außen  $\mapsto F$ aktoren
    - Höhere Priorität \* steht linkssassoziativ und innen  $\mapsto T$ erme
    - Faktoren können Bezeichner oder Ausdrücke in Klammern sein
  - Setze  $G'_7 = (\{E, T, F, I\}, \{\textcolor{red}{a}, \textcolor{blue}{b}, \textcolor{green}{c}, \textcolor{violet}{0}, \textcolor{blue}{1}, +, *, (), \}, P', E)$   
mit  $P' = \{E \rightarrow T \mid E+T, T \rightarrow F \mid T*F, F \rightarrow I \mid (E)$   
 $I \rightarrow \textcolor{red}{a} \mid \textcolor{blue}{b} \mid \textcolor{green}{c} \mid Ia \mid Ib \mid Ic \mid I0 \mid I1\}$
- $G'_7$  ist äquivalent zu  $G_7$  und eindeutig**

## BEGRÜNDUNG DER EINDEUTIGKEIT VON $G'_7$

$$P' = \{ E \rightarrow T \mid E+T, \quad T \rightarrow F \mid T*F, \quad F \rightarrow I \mid (E) \\ I \rightarrow a \mid b \mid c \mid Ia \mid Ib \mid Ic \mid Io \mid I1 \}$$

- **Jeder Ausdruck muss aus einer Termfolge bestehen**
  - Termfolge muss von rechts nach links erzeugt werden
  - Terme haben keine Ausdrücke als direkte Teile
  - Es gibt nur einen Parsebaum für  $t_1 + t_2 + \dots + t_k$
- **Jeder Term muss aus einer Faktorenfolge bestehen**
  - Faktorenfolge muss von rechts nach links erzeugt werden
  - Faktoren haben keine Terme als direkte Teile
  - Es gibt nur einen Parsebaum für  $f_1 * f_2 * \dots * f_n$
- **Jeder Faktor ist Bezeichner oder geklammerter Ausdruck**



$$L = \{0^i 1^j 2^k \mid i=j \vee j=k\} \quad \text{IST INHÄRENZ MEHRDEUTIG}$$

- **$L$  ist Vereinigung zweier kontextfreier Sprachen**

- $L_1 = \{0^i 1^i 2^k \mid i, k \in \mathbb{N}\}$
- $G_1 = (\{S_1, A\}, \{\mathbf{0}, \mathbf{1}, \mathbf{2}\}, \{S_1 \rightarrow S_1 \mathbf{2}, S_1 \rightarrow A, A \rightarrow \mathbf{0}A\mathbf{1}, A \rightarrow \epsilon\}, S_1)$
- $L_2 = \{0^i 1^j 2^j \mid i, j \in \mathbb{N}\}$
- $G_2 = (\{S_2, B\}, \{\mathbf{0}, \mathbf{1}, \mathbf{2}\}, \{S_2 \rightarrow \mathbf{0}S_2, S_2 \rightarrow B, B \rightarrow \mathbf{1}B\mathbf{2}, B \rightarrow \epsilon\}, S_2)$
- Wörter von  $L_1$  und  $L_2$  haben verschiedene Ableitungsbäume

- **Manche Wörter von  $L$  gehören zu  $L_1 \cap L_2$**

- Wörter aus  $L_1 \cap L_2$  haben eine Ableitung in  $G_1$  und eine in  $G_2$
- Wörter aus  $L$ , die zu  $L_1 \cap L_2$  gehören, haben keine eindeutige Ableitung

- **$L_1 \cap L_2 = \{0^i 1^i 2^i \mid i \in \mathbb{N}\}$  ist selbst nicht kontextfrei**

Beweis in §3.3

- Man kann keine einheitliche kfG zur Beschreibung von  $L_1 \cap L_2$  angeben
- Damit lässt sich auch keine bessere (eindeutige) kfG für  $L$  angeben

Intuitives Argument. Präziser Beweis benötigt verallgemeinertes Pumping-Lemma (Wegener §6.7)

- **Beschreibungsform für Programmiersprachen**

- Äquivalente Beschreibungsformen:

Details in Vossen/Witt §7.3, 7.4

**Backus-Naur Form**, **Syntaxdiagramme**, Definitionsgleichungen, ...

- **Compiler benötigt Ableitungsbäume**

- Rekonstruktion nur möglich für eindeutige Grammatiken
- Mehrdeutige Grammatiken können evtl. eindeutig gemacht werden
- Manche kontextfreie Sprachen haben keine eindeutige Grammatik

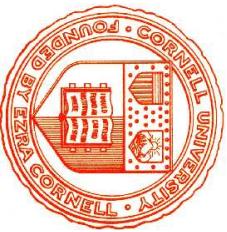
- **Noch zu klärende Fragen**

- Welches **Maschinenmodell** erkennt genau die kontextfreien Sprachen?
- Wie kann man den Ableitungsbaukasten rekonstruieren? (**Syntaxanalyse**)
- Welche **Abschlusseigenschaften** gelten für kontextfreie Sprachen?
- Welche Spracheigenschaften lassen sich nicht kontextfrei beschreiben?

# Theoretische Informatik I

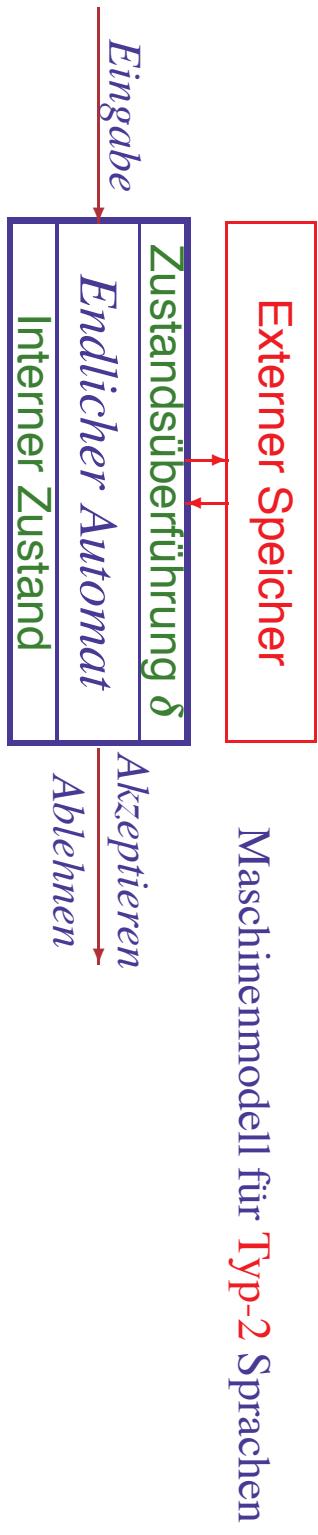
## Einheit 3.2

### Pushdown Automaten



1. Das Maschinenmodell
2. Arbeitsweise & erkannte Sprache
3. Beziehung zu Typ-2 Sprachen
4. Deterministische PDAs

# EIN MASCHINENMODELL FÜR TYP-2 SPRACHEN



- **Typ-3 Sprachen werden von NEAs akzeptiert**
  - Typ-3 Grammatik erzeugt pro Schritt ein Terminalsymbol
  - NEA verarbeitet pro Schritt ein Eingabesymbol
    - Erzeugte Terminalssymbole stehen links von der aktuellen Variablen
    - Verarbeitete Eingabesymbole führen zu aktuellem Zustand
    - Rechts von der aktuellen Variablen steht noch nichts
      - Im Zustand ist nichts über unverarbeitete Eingabesymbole bekannt
- **Welches Maschinenmodell paßt zu Typ-2 Sprachen?**
  - Kontextfreie Grammatiken können  $L_1 = \{0^m 1^m \mid m \in \mathbb{N}\}$  erzeugen
    - Ohne Zwischenspeicher können endliche Automaten  $L_1$  nicht erkennen
- **Typ-2 Maschinenmodell benötigt externen Speicher**

# WELCHES SPEICHERMODELL BRAUCHEN TYP-2 SPRACHEN?

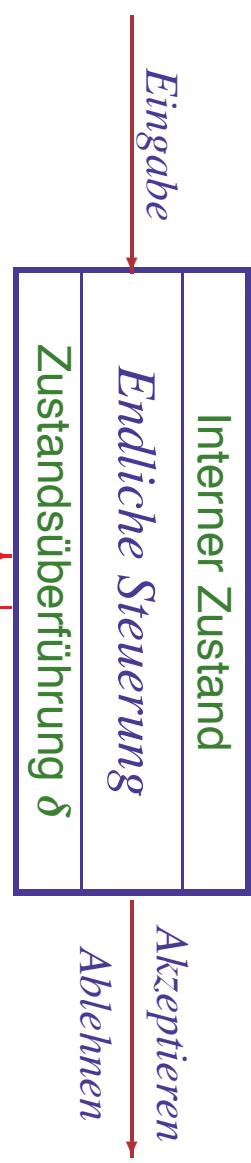
## Analysiere das Verhalten von Linksableitungen

- Links von der aktuellen Variablen  $A$  stehen nur **erzeugte Terminalsymbole**
  - Entspricht den schon verarbeiteten Eingabesymbolen des Automaten
- **Rechts** von  $A$  können bereits Terminalsymbole stehen
  - Bei Verarbeitung eines Eingabewortes muß der Automat Information speichern, welche Symbole am Ende des Wortes kommen müssen
- Ist  $A$  komplett abgearbeitet, so “springt” die Ableitung über Terminalsymbole zur nächsten Variablen
  - Automat muß zuletzt erzeugte Information zuerst abarbeiten



**Speicher des Automaten sollte ein Stack sein**

# PUSHDOWN-AUTOMATEN INTUITIV



- **Endlicher Automat + Stack**

- Endliche Steuerung liest Eingabesymbole

- Gleichzeitig kann das **oberste Symbol im Stack** beobachtet werden

- **Eingabe und Stack wird gleichzeitig bearbeitet**

- Gelesenes Symbol wird aus Eingabe “entfernt”
  - Zustand kann verändert werden
  - Oberstes Stacksymbol wird durch (mehrere) neue Stacksymbole ersetzt
  - Nichtdeterministische Entscheidungen / spontane  $\epsilon$ -Übergänge möglich

# PUSHDOWN-AUTOMATEN – MATHEMATISCH PRÄZISIERT



## Ein Pushdown-Automat (PDA, Kellerautomat)

ist ein 7-Tupel  $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  mit

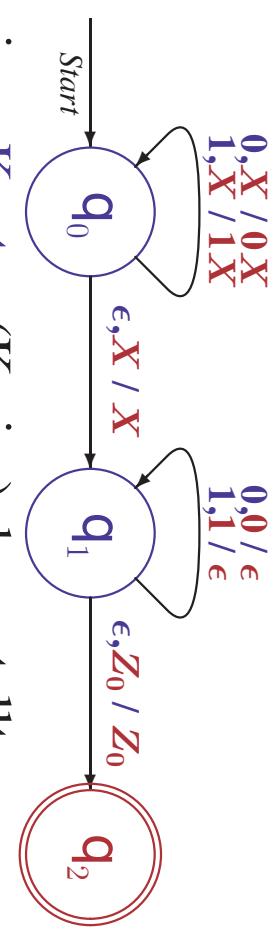
- $Q$  nichtleere endliche **Zustandsmenge**
- $\Sigma$  endliches **Eingabealphabet**
- $\Gamma$  endliches **Stackalphabet**
- $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow \mathcal{P}_e(Q \times \Gamma^*)$  **Überführungsfunktion** (endlich)
- $q_0 \in Q$  **Startzustand**
- $Z_0 \in \Gamma$  **Initialsymbol des Stacks**
- $F \subseteq Q$  Menge von **akzeptierenden (End-)Zuständen**

Pushdown-Automaten sind üblicherweise nichtdeterministisch!

# BESCHREIBUNG VON PUSHDOWN-AUTOMATEN

- **Übergangsdiagramme**

- Jeder Zustand in  $Q$  wird durch einen **Knoten** (Kreise) dargestellt
- Für  $(p, \alpha) \in \delta(q, a, X)$ ,  $a \in \Sigma \cup \{\epsilon\}$  hat das Diagramm eine **Kante**  $q \xrightarrow{a, X / \alpha} p$  (mehrere Beschriftungen derselben Kante und Verwendung von Platzhaltern möglich)
- $q_0$  wird durch einen **mit Start beschrifteten Pfeil** angezeigt
- Endzustände in  $F$  werden durch doppelte Kreise gekennzeichnet
- $\Sigma$  und  $\Gamma$  implizit durch Diagramm bestimmt, Initialsymbol heißt  $Z_0$

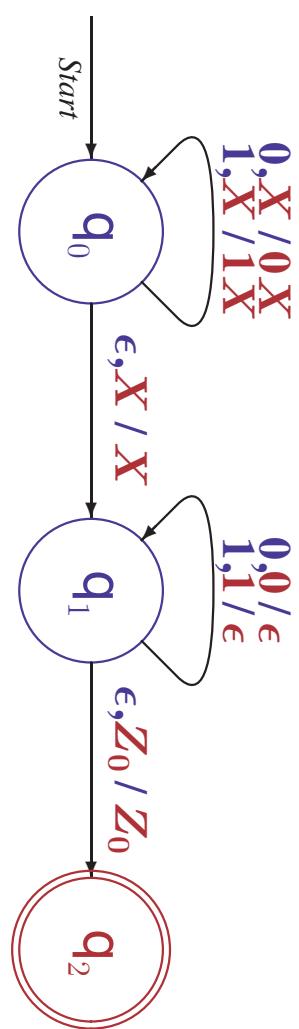


- **Übergangstabellen**

- Tabellarische Darstellung der Funktion  $\delta$
- Kennzeichnung von  $q_0$  durch einen **Pfeil**
- Kennzeichnung von  $F$  durch **Sterne**
- $\Sigma$ ,  $\Gamma$  und  $Q$  implizit durch die Tabelle bestimmt
- **Wildcardvariablen** für  $a \in \Sigma \cup \{\epsilon\}$ ,  $X \in \Gamma$  erlaubt

$Q \times \Sigma + \epsilon \times \Gamma$	Resultat
$\rightarrow q_0 \quad 0 \quad X$	$q_0, 0X$
$\rightarrow q_0 \quad 1 \quad X$	$q_0, 1X$
$\rightarrow q_0 \quad \epsilon \quad X$	$q_1, X$
$q_1 \quad 0 \quad 0$	$q_1, \epsilon$
$q_1 \quad 1 \quad 1$	$q_1, \epsilon$
$q_1 \quad \epsilon \quad Z_0$	$q_2, Z_0$
$*$ $q_2$	

# PUSHDOWN-AUTOMAT FÜR $\{ww^R \mid w \in \{0, 1\}^*\}$



- **Speichere  $w$  in  $q_0$** 
  - Es wird je ein Symbol gelesen und auf den Stack gelegt
    - $\delta(q_0, a, X) = \{(q_0, aX)\}$  für  $a \in \{0, 1\}$ ,  $X \in \Gamma$
- **Spontaner Wechsel “in der Mitte”**
  - $\delta(q_0, \epsilon, X) = \{(q_1, X)\}$  für  $X \in \Gamma$  (nichtdeterministischer  $\epsilon$ -Übergang)
- **Verarbeite  $w^R$  in  $q_1$** 
  - Jedes gelesene Symbol wird dem obersten Stacksymbol verglichen
    - $\delta(q_1, a, a) = \{(q_1, \epsilon)\}$  für  $a \in \{0, 1\}$
- **“Leerer” Stack akzeptiert**
  - Wenn Stack leer ist, wurde  $w^R$  in  $q_1$  verarbeitet
    - $\delta(q_1, \epsilon, Z_0) = \{(q_2, Z_0)\}$  (deterministischer  $\epsilon$ -Übergang)

$$P = (q_0, q_1, q_2, \{0,1\}, \{0,1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

## Generalisiere Konzept der Konfigurationsübergänge

- **Erweitere Begriff der Konfiguration**

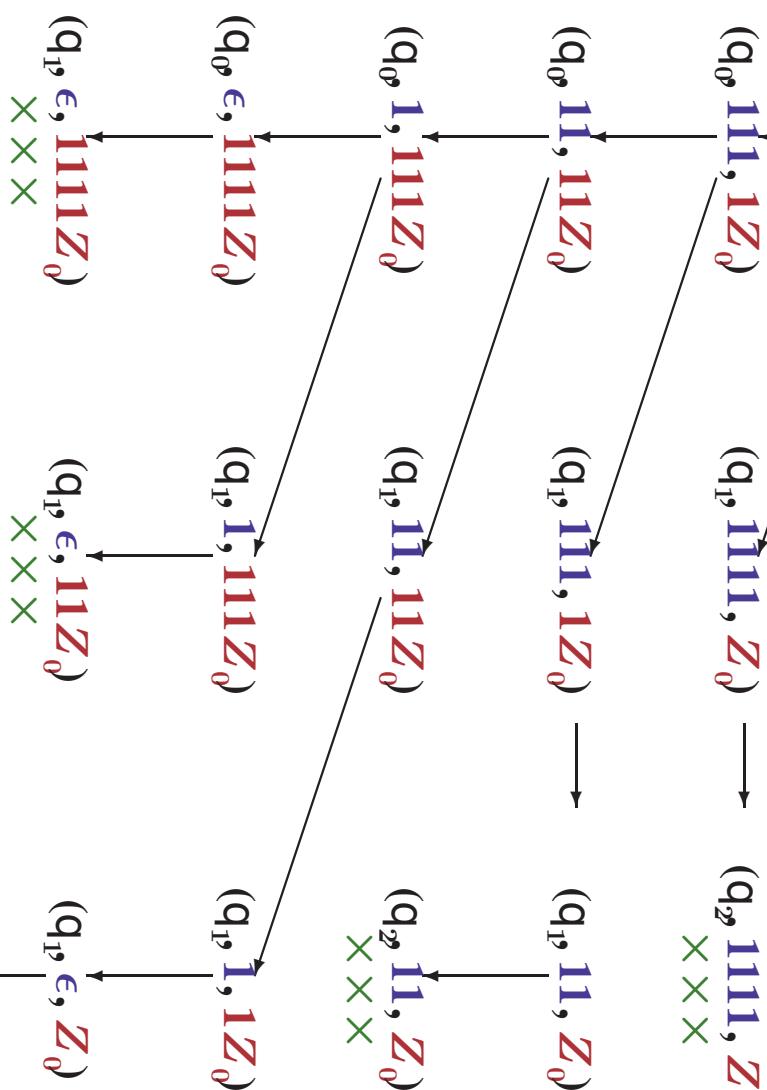
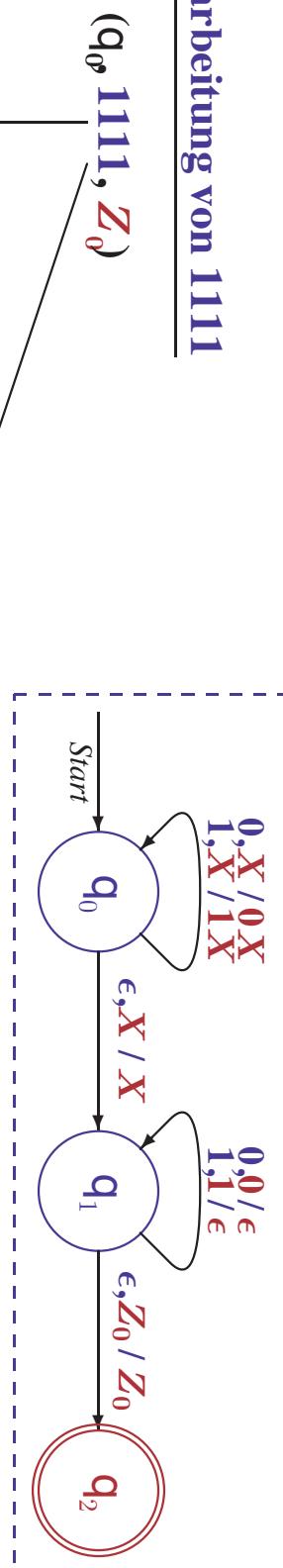
- Aktueller Zustand, Inhalt des Stacks und unverarbeitete Eingabe zählt
- Formal dargestellt als Tripel  $\mathbf{K} = (q, w, \gamma) \in Q \times \Sigma^* \times \Gamma^*$

- **Modifizierte Konfigurationsübergangsrelation  $\vdash^*$**

- Wechsel zwischen Konfigurationen durch Abarbeitung von Wörtern
- $(q, aw, X\beta) \vdash (p, w, \alpha\beta)$ , falls  $(p, \alpha) \in \delta(q, a, X)$
- $(q, w, X\beta) \vdash (p, w, \alpha\beta)$ , falls  $(p, \alpha) \in \delta(q, \epsilon, X)$ 
  - (Im Zustand  $q$  ist  $a$  das erste Eingabesymbol und  $X$  oben im Stack.  
 $a$  wird abgearbeitet,  $X$  durch  $\alpha$  ersetzt, der Rest bleibt stehen)
- $\mathbf{K}_1 \vdash^* \mathbf{K}_2$ , falls  $K_1 = K_2$  oder
  - es gibt eine Konfiguration  $K$  mit  $K_1 \vdash K$  und  $K \vdash^* K_2$

# ABARBEITUNG DES PALINDROM PDA

## Verarbeitung von 111



# AKZEPTIERTE SPRACHE EINES PUSHDOWN-AUTOMATEN

## Zwei alternative Definitionen möglich

- Akzeptanz durch akzeptierende Endzustände

$$L_F(P) = \{ w \in \Sigma^* \mid \exists q \in F. \exists \beta \in \Gamma^*. (q_0, w, Z_0) \vdash^* (q, \epsilon, \beta) \}$$

- Standarddefinition: Nach Abarbeitung der Eingabe entscheidet der Zustand, ob das Wort akzeptiert wird

- Akzeptanz durch leeren Stack

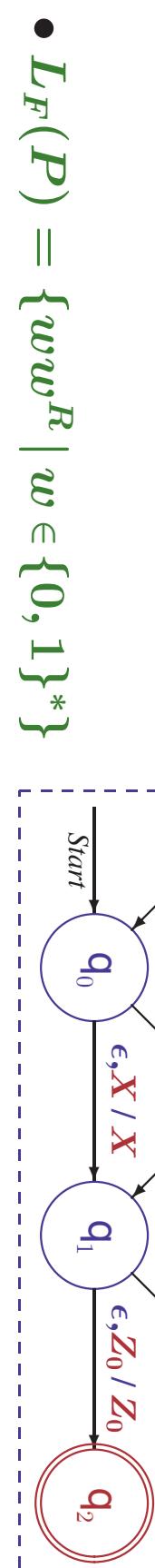
$$L_\epsilon(P) = \{ w \in \Sigma^* \mid \exists q \in Q. (q_0, w, Z_0) \vdash^* (q, \epsilon, \epsilon) \}$$

- Oft praktischer: Nach Abarbeitung der Eingabe sind auch alle zwischengelagerten Symbole verarbeitet

- **Definitionen haben verschiedene Effekte**

- Sprachen können für konkrete PDAs sehr verschieden ausfallen
- **Beide Definitionen sind gleichmächtig**
  - PDA kann passend zur anderen Definition umgewandelt werden

# DIE BEIDEN SPRACHEN DES PALINDROMAUTOMATEN



- $L_F(P) = \{ww^R \mid w \in \{0, 1\}^*\}$

$\supseteq$ : Durch strukturelle Induktion zeige, daß für jedes Wort  $w$  gilt

$$(q_0, ww^R, Z_0) \vdash^* (q_0, w^R, w^R Z_0) \vdash (q_1, w^R, w^R Z_0) \vdash^* (q_1, \epsilon, Z_0) \vdash (q_2, \epsilon, Z_0)$$

$\subseteq$ : Durch strukturelle Induktion über  $x = x_1..x_n$  zeige

Wenn  $(q_0, x, \alpha') \vdash^* (q_1, \epsilon, \alpha')$  für ein  $\alpha \in \Gamma^*$ , dann  $x = ww^R$  für ein  $w \in \{0, 1\}^*$

Kernidee des Induktionssschrittes

(Details in HMU §6.2.1)

$$\begin{aligned} \text{Wenn } (q_0, x_1..x_n, \alpha') \vdash^* & (q_0, x_2..x_n, x_1\alpha') \vdash^* (q_1, x_i..x_n, \beta x_1\alpha') \\ \vdash^* & (q_1, x_n, x_1\alpha') \vdash^* (q_1, \epsilon, \alpha') \quad \text{für } \alpha', \beta \in \Gamma^*, \\ \text{dann folgt } (q_0, x_1..x_{n-1}, \alpha') \vdash^* & (q_0, x_2..x_{n-1}, x_1\alpha') \vdash^* \dots \vdash^* (q_1, \epsilon, x_1\alpha') \end{aligned}$$

und  $x_1 = x_n$  und per Induktion  $x_2..x_{n-1} = vv^R$  für ein  $v \in \{0, 1\}^*$

- $L_\epsilon(P) = \emptyset$  weil  $Z_0$  nie gelöscht wird

Modifikation von  $P$ : Ändere Kantenschriftung von  $q_1$  nach  $q_2$  in  $\epsilon, Z_0 / \epsilon$

Für den resultierenden PDA  $P'$  gilt:  $L_\epsilon(P') = L_F(P) = \{ww^R \mid w \in \{0, 1\}^*\}$

# WICHTIGE ERKENNTNISSE ZU AUSSAGEN ÜBER KONFIGURATIONSÜBERGÄNGE IN BEWEISEN

- **Ungelesene Eingaben können ignoriert werden**

Gilt  $(q,xw,\alpha) \vdash^* (p,yw,\beta)$  dann gilt auch  
 $(q,x,\alpha) \vdash^* (p,y,\beta)$  für alle  $w \in \Sigma^*$

Dagegen kann es von Bedeutung sein, ob im Stack hinter  $\alpha$  etwas steht

- Beweis durch Induktion über Anzahl der Konfigurationsschritte
- Kernargument:  $(q,ayw,X\beta) \vdash (p,yw,\gamma\beta)$  verlangt  $(p,\gamma) \in \delta(q,a,X)$   
also  $(q,ay,X\beta) \vdash (p,y,\gamma\beta)$

- **Erweiterung von Eingabe oder Stack ändert nichts**

Gilt  $(q,x,\alpha) \vdash^* (p,y,\beta)$  dann gilt auch  
 $(q,xw,\alpha\gamma) \vdash^* (p,yw,\beta\gamma)$  für alle  $w \in \Sigma^*, \gamma \in \Gamma^*$

Weder  $w$  noch  $\gamma$  werden bei der Verarbeitung angesehen

- Beweis durch Induktion über Anzahl der Konfigurationsschritte
- Kernargument:  $(q,aw,X\gamma) \vdash (p,w,\beta\gamma)$ , falls  $(p,\beta) \in \delta(q,a,X)$   
was hinter  $a$  bzw.  $X$  kommt, bleibt unangetastet

# ERKENNEN MIT LEEREM STACK IST OFT EINFACHER

## Konstruiere PDA für korrekte Klammerausdrücke

- **Rahmenbedingungen an Eingabewörter**

- Anzahl geöffneter und geschlossener Klammern muß gleich sein
- In keinen Anfangssegment dürfen mehr ( als ) vorkommen

- **Zähle Überschuß geöffneter Klammern im Stack**

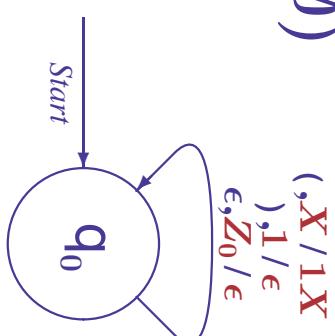
- Jedes ( erhöht die Anzahl, jedes ) erniedrigt sie
- ) ist nicht erlaubt, wenn der Stackboden erreicht ist
- Am Ende des Wortes wird der Stackboden entfernt

- Setze  $P_1 = (\{q\}, \{((), )\}, \{Z_0, 1\}, \delta, q, Z_0, \emptyset)$

mit  $\delta(q, (, X) = \{(q, 1X)\}$

$$\delta(q, ), 1) = \{(q, \epsilon)\}$$

$$\delta(q, \epsilon, Z_0) = \{(q, \epsilon)\}$$



## TRANSFORMATION VON $L_\epsilon$ - IN $L_F$ -AUTOMATEN

**Zu jedem PDA  $P_\epsilon = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$  kann ein PDA  $P_F$  konstruiert werden mit  $L_\epsilon(P_\epsilon) = L_F(P_F)$**

- Bei leerem Stack wechsele in einen Endzustand

- Neues Initialsymbol  $X_0$  markiert unteres Ende des Stacks von  $P_F$
- Neuer Anfangszustand  $p_0$  für  $P_F$  schreibt Initialsymbol von  $P_\epsilon$  auf Stack
- Neuer Endzustand  $p_f$ , in den bei “leerem” Stack gewechselt wird

- $P_F = (Q \cup \{p_0, p_f\}, \Sigma, \Gamma \cup \{X_0\}, \delta_F, p_0, X_0, \{p_f\})$

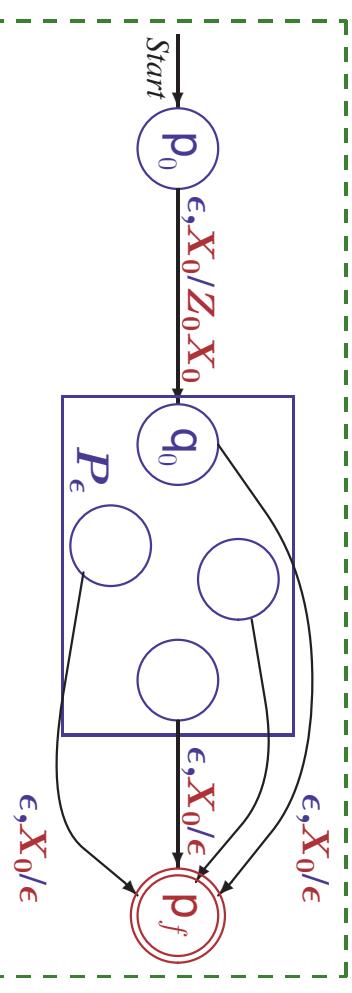
- $\delta_F(q, a, X) = \delta(q, a, X)$

für alle  $q \in Q, X \in \Gamma$

- $\delta_F(p_0, \epsilon, X_0) = \{(q_0, Z_0 X_0)\}$

- $\delta_F(q, \epsilon, X_0) = \{(p_f, \epsilon)\}$

für alle  $q \in Q$



Korrektheitsbeweis durch Detailanalyse

# UMWANDLUNG EINES $L_\epsilon$ -PDA IN EINEN $L_F$ -PDA

- Gegeben  $P_\epsilon = (\{q\}, \{(,\ )\}, \{Z_0, 1\}, \delta, q, Z_0, \emptyset)$

mit  $\delta(q, (, \mathbf{X}) = \{(q, \mathbf{1X})\}$

$\delta(q, ), \mathbf{1} = \{(q, \epsilon)\}$

$\delta(q, \epsilon, \mathbf{Z}_0) = \{(q, \epsilon)\}$

- Äquivalenter PDA  $P_F$  mit Endzuständen ist

$(\{p_0, q, p_f\}, \{(,\ )\}, \{X_0, Z_0, 1\}, \delta_F, p_0, X_0, \{p_f\})$

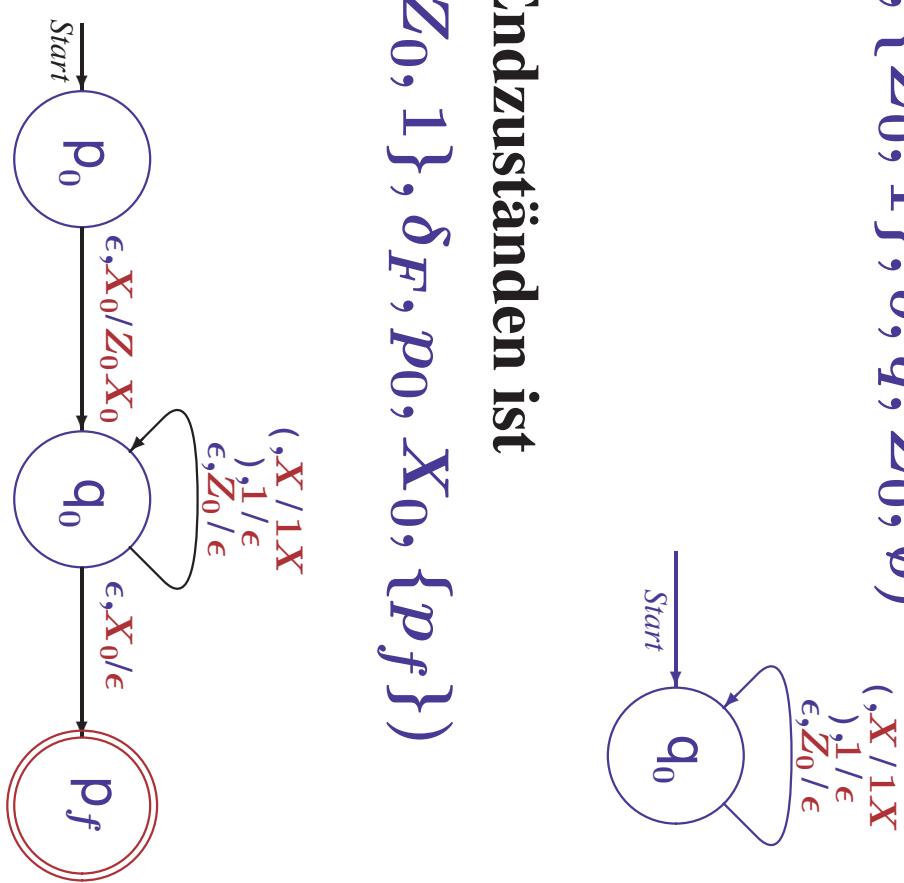
mit  $\delta_F(p_0, \epsilon, \mathbf{X}_0) = \{(q, \mathbf{Z}_0 \mathbf{X}_0)\}$

$\delta_F(q, (, \mathbf{X}) = \{(q, \mathbf{1X})\}$

$\delta_F(q, ), \mathbf{1} = \{(q, \epsilon)\}$

$\delta_F(q, \epsilon, \mathbf{Z}_0) = \{(q, \epsilon)\}$

$\delta_F(q, \epsilon, \mathbf{X}_0) = \{(p_f, \epsilon)\}$



# TRANSFORMATION VON $L_F$ - IN $L_\epsilon$ -AUTOMATEN

**Zu jedem PDA  $P_F = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  kann ein PDA  $P_\epsilon$  konstruiert werden mit  $L_F(P_F) = L_\epsilon(P_\epsilon)$**

- **Im Endzustand leere den Stack**

- Neuer Stacklösch-Zustand  $p$ , in den von Endzuständen gewechselt wird
- Neues Initialsymbol  $X_0$  für  $P_\epsilon$  verhindert irrtümliches Leeren des Stacks
- Neuer Anfangszustand  $p_0$  für  $P_\epsilon$  schreibt Initialsymbol von  $P_F$  auf Stack

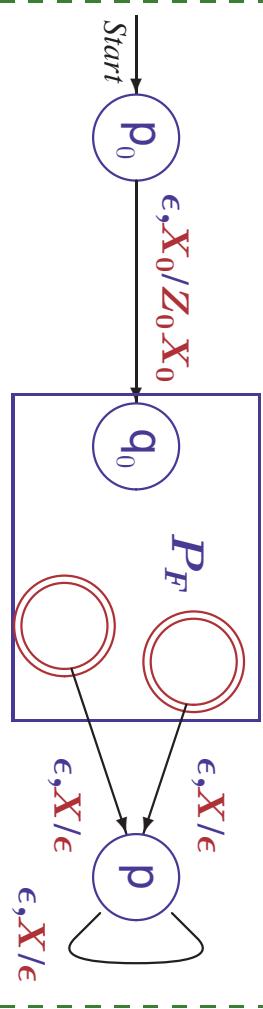
- $P_\epsilon = (Q \cup \{p_0, p\}, \Sigma, \Gamma \cup \{X_0\}, \delta_\epsilon, p_0, X_0, \emptyset)$

- $\delta_\epsilon(q, a, X) = \delta(q, a, X)$   
für alle  $q \in Q, X \in \Gamma$

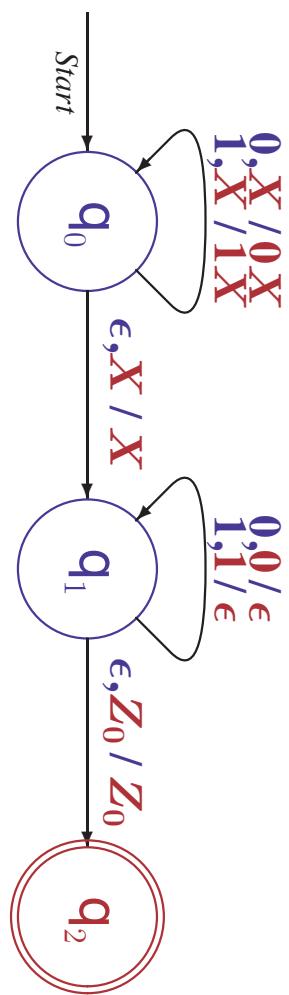
- $\delta_\epsilon(p_0, \epsilon, X_0) = \{(q_0, Z_0 X_0)\}$
- $\delta_\epsilon(q, \epsilon, X) = \{(p, \epsilon)\}$   
für alle  $q \in F$

- $\delta_\epsilon(p, \epsilon, X) = \{(p, \epsilon)\}$   
für alle  $X \in \Gamma \cup \{X_0\}$

Korrektheitsbeweis durch Detailanalyse



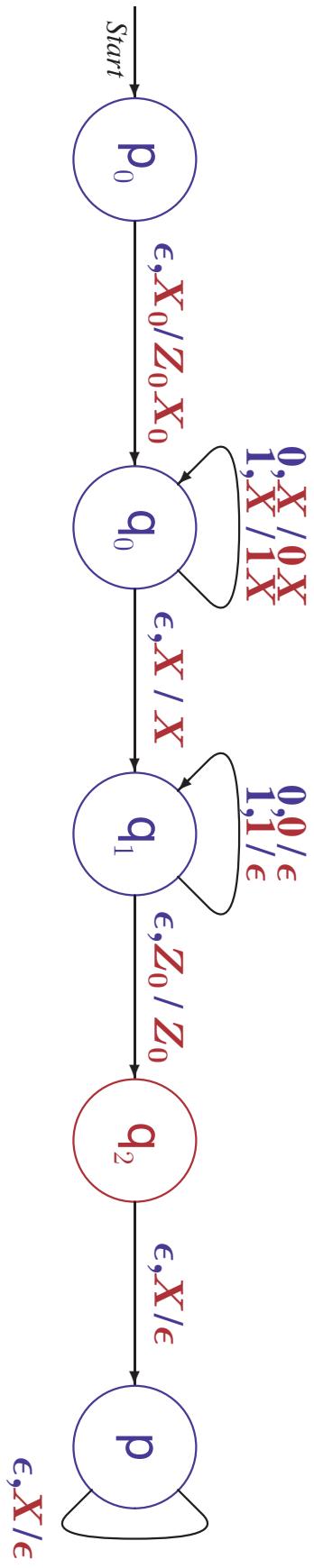
## UMWANDLUNG EINES $L_F$ -PDA IN EINEN $L_\epsilon$ -PDA



- $P_F = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$  mit  $\delta$  wie oben erkennt  $\{ww^R \mid w \in \{0, 1\}^*\}$  mit Endzustand

• Äquivalenter PDA  $P_\epsilon$  mit leerem Stack ist

$(\{p_0, q_0, q_1, q_2, p\}, \{0, 1\}, \{0, 1, Z_0, X_0\}, \delta_\epsilon, p_0, X_0, \{p\})$



# SIND PDAs WIRKLICH MASCHINEN FÜR TYP-2 SPRACHEN?

$$\mathcal{L}_2 = \mathcal{L}_{PDA} = \{ L \mid \exists P: \text{PDAs. } L = L_\epsilon(P) \}$$

- **Konfigurationsübergänge  $\hat{=}$  Linksableitungen**

- $(q_0, xy, Z_0) \vdash^* (q, y, A\alpha)$  bedeutet, daß  $P$  nach Verarbeitung von  $x$  im Zustand  $q$  ist und noch  $y$  und den Stack  $A\alpha$  zu verarbeiten hat
- $A\alpha$  muß gespeichert und beim Lesen von  $y$  komplett verarbeitet werden
- Linksableitung  $S \xrightarrow{*} xA\alpha \xrightarrow{*} xy$  erzeugt aus dem Startsymbol zuerst das Wort  $xA\alpha$  und muß dann  $y$  aus  $A\alpha$  ableiten

- **Grammatik  $\rightarrow$  Pushdown-Automat**

- PDA muß Linksableitung auf Stack simulieren
- Erzeugte linke Terminalteilelörter müssen mit Teil der Eingabe verglichen werden, um nächste Variable freizulegen

- **Pushdown-Automat  $\rightarrow$  Grammatik**

- Grammatik muß Abarbeitung von Symbolen des Stacks simulieren
- Regeln beschreiben wie PDA bei Abarbeitung des Stacksymbols  $X$  mit  $\delta$  Zwischenwörter im Stack auf- und schließlich wieder abbaut

# VON GRAMMATIKEN ZU PUSHDOWN-AUTOMATEN

**Zu jeder kontextfreien Grammatik  $G = (V, T, P_G, S)$  kann ein PDA  $P$  konstruiert werden mit  $L(G) = L_\epsilon(P)$**

- **Stack simuliert Linksableitungen von  $G$**

- Beginne mit Startsymbol von  $G$

- $A \in V$  wird im Stack durch rechte Seite  $\beta$  einer Regel  $A \rightarrow \beta$  ersetzt
- $a \in T$  wird vom Stack entfernt, wenn es als Eingabe erscheint, um im Stack die nächsten Variable einer Linksableitung freizulegen

- **Generierter PDA  $P = (\{q\}, T, V \cup T, \delta, q, S, \emptyset)$**

- $\delta(q, \epsilon, A) = \{(q, \beta) \mid A \rightarrow \beta \in P_G\}$  für alle  $A \in V$
- $\delta(q, a, a) = \{(q, \epsilon)\}$  für alle  $a \in T$

- **Korrektheitsbeweis  $L(G) = L_\epsilon(P)$**  (Details folgen)

Zeige: ( $\subseteq$ ) Wenn  $S = x_1 A_1 \alpha_1 \dots \xrightarrow{L} x_m A_m \alpha_m \xrightarrow{L} w \in T^*$  dann gibt es für alle  $i$  ein  $y_i$  mit  $w = x_i y_i$  und  $(q, w, S) \vdash^* (q, y_i, A_i \alpha_i)$

( $\supseteq$ ) Wenn  $(q, w, X) \vdash^* (q, \epsilon, \epsilon)$  dann  $X \xrightarrow{*} w$

# UMWANDLUNG EINER GRAMMATIK IN EINEN PDA

- $G_7 = (\{E, I\}, \{a, b, c, 0, 1, +, *, (), (\ )\}, P_G, E)$

mit  $P_G = \{ E \rightarrow I \mid E+E \mid E*E \mid (E) \}$   
 $I \rightarrow a \mid b \mid c \mid Ia \mid Ib \mid Ic \mid I0 \mid I1 \}$

- Erzeuge  $P = (\{q\}, T, V \cup T, \delta, q, E, \emptyset)$

mit  $V = \{E, I\}$  und  $T = \{a, b, 0, 1, +, *, (), (\ )\}$

- $\delta(q, \epsilon, E) = \{(q, I), (q, E+E), (q, E*E), (q, (E))\}$
- $\delta(q, \epsilon, I) = \{(q, a), (q, b), (q, c), (q, Ia), (q, Ib), (q, Ic), (q, I0), (q, I1)\}$
- $\delta(q, a, a) = \{(q, \epsilon)\} \quad - \delta(q, +, +) = \{(q, \epsilon)\}$
- $\delta(q, b, b) = \{(q, \epsilon)\} \quad - \delta(q, *, *) = \{(q, \epsilon)\}$
- $\delta(q, c, c) = \{(q, \epsilon)\} \quad - \delta(q, (, )) = \{(q, \epsilon)\}$
- $\delta(q, 0, 0) = \{(q, \epsilon)\} \quad - \delta(q, , ) = \{(q, \epsilon)\}$
- $\delta(q, 1, 1) = \{(q, \epsilon)\}$

## KORREKTHEITSBEWEIS IM DETAIL: $L(G) \subseteq L_\epsilon(P)$

**Wenn  $S = x_1 A_1 \alpha_1 \dots \xrightarrow{L} x_m A_m \alpha_m \xrightarrow{L} w \in T^*$  ( $x_i \in T^*$ ,  $A_i \in V$ ) dann gibt es für alle  $i$  ein  $y_i$  mit  $w = x_i y_i$  und  $(q, w, S) \vdash^* (q, y_i, A_i \alpha_i)$**

- Beweis durch Induktion über  $i \leq m$
- Basisfall  $i = 1$ :  $S = x_1 A_1 \alpha_1 \xrightarrow{*} w$ 
  - Es folgt  $S = A_1$  und  $x_1 = \alpha_1 = \epsilon$ , also muß  $y_1 = w$  gewählt werden
  - $(q, w, S) \vdash^* (q, w, S)$  gilt mit 0 Konfigurationsübergängen
- InduktionsSchritt:  $S.. \xrightarrow{L} x_i A_i \alpha_i \xrightarrow{L} x_{i+1} A_{i+1} \alpha_{i+1} \xrightarrow{*} w$ 
  - $x_i A_i \alpha_i \xrightarrow{L} x_{i+1} A_{i+1} \alpha_{i+1}$  verlangt  $A_i \xrightarrow{\beta_i} \in P_G$  für ein  $\beta_i$ ,
  - wobei  $x_i \beta_i \alpha_i = x_i z A_{i+1} \alpha_{i+1}$  für ein  $z \in T^*$  und  $x_{i+1} = x_i z \sqsubseteq w$ .
  - Per Konstruktion gilt dann  $(q, \beta_i) \in \delta(q, \epsilon, A_i)$  und mit der Induktionsannahme folgt  $(q, w, S) \vdash^* (q, y_i, A_i \alpha_i) \vdash (q, y_i, z A_{i+1} \alpha_{i+1})$
  - Wegen  $x_{i+1} = x_i z \sqsubseteq w$  und  $w = x_i y_i$  kann  $y_i$  zerlegt werden in  $z y_{i+1}$  und der PDA arbeitet  $z$  ab:  $(q, y_i, z A_{i+1} \alpha_{i+1}) \vdash^* (q, y_{i+1}, A_{i+1} \alpha_{i+1})$
  - Schlußfolgerung:  $S = x_1 A_1 \alpha_1 \dots \xrightarrow{L} x_m A_m \alpha_m \xrightarrow{L} w \in T^*$ 
    - Wegen  $w \in T^*$  folgt  $A_m \xrightarrow{\beta_m} \in P_G$  für ein  $\beta_m \in T^*$  und  $w = x_m \beta_m \alpha_m$
    - Also  $(q, w, S) \vdash^* (q, \beta_m \alpha_m, A_m \alpha_m) \vdash (q, \beta_m \alpha_m, \beta_m \alpha_m) \vdash^* (q, \epsilon, \epsilon)$ , d.h.  $w \in L_\epsilon(P)$

# KORREKTHEITSBEWEIS IM DETAIL: $L(G) \supseteq L_\epsilon(P)$

Für alle  $X \in V$  gilt: wenn  $(q, w, X) \vdash^* (q, \epsilon, \epsilon)$  dann  $X \xrightarrow{*} w$

- Beweis durch Induktion über Länge der PDA Berechnung
- Basisfall:  $(q, w, X) \vdash (q, \epsilon, \epsilon)$ 
  - Es folgt  $X \xrightarrow{\epsilon} \epsilon \in P_G$  und  $w = \epsilon$ , also  $X \xrightarrow{*} w$
- Induktionsschritt:  $(q, w, X) \vdash^{n+1} (q, \epsilon, \epsilon)$ 
  - Da  $X$  oben im Stack steht, muß der erste Schritt die Form  $(q, w, X) \vdash (q, w, Y_1..Y_k)$  für ein  $X \xrightarrow{*} Y_1..Y_k \in P_G$  haben ( $Y_i \in V \cup T$ )
  - Dann gibt eine Zerlegung  $w = w_1..w_k$  mit
$$(q, w_1w_2..w_k, Y_1Y_2..Y_k) \vdash^* (q, w_2..w_k, Y_2..Y_k) \vdash^* (q, \epsilon, \epsilon)$$
  - Es folgt  $(q, w_iw_{i+1}..w_k, Y_i) \vdash^* (q, w_{i+1}..w_k, \epsilon)$  also  $(q, w_i, Y_i) \vdash^* (q, \epsilon, \epsilon)$
  - Per Induktionsannahme folgt  $Y_i \xrightarrow{*} w_i$  für alle  $i$  (für  $Y_i \in T$  ist  $Y_i = w_i$ )  
also  $X \xrightarrow{*} Y_1..Y_k \xrightarrow{*} w_1..w_k = w$
- Es folgt  $L_\epsilon(P) = \{w \mid (q, w, S) \vdash^* (q, \epsilon, \epsilon)\} \subseteq \{w \mid S \xrightarrow{*} w\} = L(G)$

# VON PUSHDOWN-AUTOMATEN ZU GRAMMATIKEN

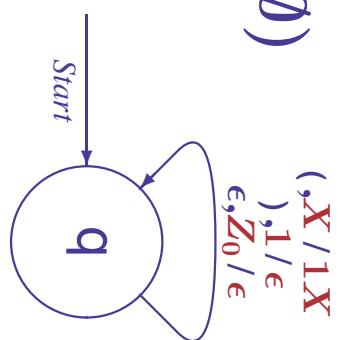
**Zu jedem PDA  $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  kann eine kfG  $G$  konstruiert werden mit  $L_\epsilon(P) = L(G)$**

- **Simuliere Abarbeitung eines Symbols vom Stack**

- Verarbeite Variablen der Form “ $(q, X, p)$ ” mit impliziter Bedeutung  
“*Entfernen von  $X$  kann von Zustand  $q$  zu Zustand  $p$  führen*”
- Entfernen von  $X$  kann zuerst ein  $Y_1..Y_m$  auf- und dann abbauen
- Beginne mit Erzeugung von  $Z_0$  und zeige, daß  $Z_0$  entfernt werden kann
- **Generiere  $G = (\{S\} \cup Q \times \Gamma \times Q, \Sigma, P_G, S)$  mit**
  - $S \xrightarrow{} (q_0, Z_0, q) \in P_G$  für alle  $q \in Q$
  - $(q, X, q_m) \xrightarrow{} a (p, Y_1, q_1) \dots (q_{m-1}, Y_m, q_m) \in P_G$ ,  
für beliebige Kombinationen  $q_1, \dots, q_m \in Q$ , falls  $(p, Y_1..Y_m) \in \delta(q, a, X)$   
 $(q, X, p) \xrightarrow{} a \in P_G$ ,
- **Korrekttheitsbeweis  $L_\epsilon(P) = L(G)$** 
  - Zeige:  $(q, X, p) \xrightarrow{*} w \in \Sigma^*$  genau dann, wenn  $(q, w, X) \vdash^* (p, \epsilon, \epsilon)$   
⊆: Induktion über Länge der PDA Berechnung  
⊇: Induktion über Länge der Ableitung  
(viele Details)

## UMWANDLUNG EINES PDA IN EINE GRAMMATIK

- Gegeben  $P_1 = (\{q\}, \{(,\ )\}, \{Z_0, 1\}, \delta, q, Z_0, \emptyset)$
- mit  $\delta(q, (, X) = \{(q, 1X)\}$  für  $X \in \{Z_0, 1\}$
- $\delta(q, ), 1) = \{(q, \epsilon)\}$
- $\delta(q, \epsilon, Z_0) = \{(q, \epsilon)\}$



- Generiere  $G = (\{(\, , )\}, \{S, (q, Z_0, q), (q, 1, q)\}, P_G, S)$

mit  $P_G = S \rightarrow (q, Z_0, q)$

$(q, Z_0, q) \rightarrow ((q, 1, q)(q, Z_0, q))$

$(q, Z_0, q) \rightarrow \epsilon$

$(q, 1, q) \rightarrow ((q, 1, q)(q, 1, q))$

$(q, 1, q) \rightarrow )$

Wähle Kurzschreibweise  $A/B$  für Hilfssymbole  $(q, Z_0, q)$  bzw.  $(q, 1, q)$ :

$$G = (\{(\, , )\}, \{S, A, B\}, P, S)$$

$$\text{mit } P = \{S \rightarrow A, A \rightarrow (BA, A \rightarrow \epsilon, B \rightarrow (BB, B \rightarrow ))\}$$

# BRÄUCHEN WIR NICHTDETERMINISTISCHE AUTOMATEN?

- **Grammatiken sind nichtdeterministisch**
  - Nichtdeterministische Automaten sind das “natürliche” Gegenstück
    - Grammatikregeln führen zu mengenwertiger Überführungsfunktion
    - “Wirkliche” Automaten müssen deterministisch sein
- **Typ-3 Sprachen haben deterministische Modelle**
  - NEAs können in äquivalente DEAs umgewandelt werden
  - Teilmengenkonstruktion kann Automaten exponentiell vergrößern
- **Reichen deterministische PDAs für Typ-2 Sprachen?**
  - Überführungsfunktion  $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow Q \times \Gamma^*$  muß eindeutig sein
  - Gibt es für PDAs immer äquivalente deterministische PDAs?

# DETERMINISTISCHE PUSHDOWN-AUTOMATEN PRÄZISIERT

- Ein Deterministischer Pushdown-Automat (DPDA)  
ist ein 7-Tupel  $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  mit
  - $Q$  nichtleere endliche Zustandsmenge
  - $\Sigma$  endliches Eingabealphabet
  - $\Gamma$  endliches Stackalphabet
  - $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow Q \times \Gamma^*$  Überführungsfunktion
    - $\delta(q, \epsilon, X)$  nur definiert, wenn  $\delta(q, a, X)$  für alle  $a \in \Sigma$  undefiniert
  - $q_0 \in Q$  Startzustand
  - $Z_0 \in \Gamma$  Initialsymbol des Stacks
  - $F \subseteq Q$  Menge von akzeptierenden (End-)Zuständen
- Erkannte Sprache
  - $L_F(P) = \{ w \in \Sigma^* \mid \exists q \in F. \exists \beta \in \Gamma^*. (q_0, w, Z_0) \vdash^* (q, \epsilon, \beta) \}$
  - $L_\epsilon(P) = \{ w \in \Sigma^* \mid \exists q \in Q. (q_0, w, Z_0) \vdash^* (q, \epsilon, \epsilon) \}$

# DPDAs SIND NICHT MÄCHTIG GENUG

## DPDA-Sprachen sind eine echte Teilklasse von $\mathcal{L}_2$

- $L(DPDA) \subseteq \mathcal{L}_2$

- Jeder DPDA ist ein spezieller PDA

- **DPDAs können  $\{ww^R \mid w \in \{0, 1\}^*\}$  nicht erkennen**

Intuitiv: DPDA  $P$  kann nicht entscheiden, wo die Mitte eines Wortes liegt

- Wenn  $0^n 1 1 0^n$  (großes  $n$ ) gelesen ist, ist Stack durchs Zählen geleert
- Wenn noch einmal  $0^m 1 1 0^m$  gelesen wird, muß  $P$  akzeptieren
- Wenn stattdessen  $0^m 1 1 0^m$  ( $m \neq n$ ) kommt, darf  $P$  nicht akzeptieren
- Aber die Information über  $n$  ist nicht mehr gespeichert

Technisches Argument:  $\{ww^R \mid w \in \{0, 1\}^*\}$  ist keine präfixfreie Sprache  
*„kein Wort aus  $L_\epsilon(P)$  ist echtes Präfix eines anderen Wortes aus  $L_\epsilon(P)$ “*

- **DPDAs erkennen nur präfixfreie Sprachen**

- Ist  $u \in L_\epsilon(P)$  dann stoppt  $P$  nach Verarbeitung von  $u$  mit leerem Stack
- Eine echte Erweiterung von  $u$  wird von  $P$  nicht komplett abgearbeitet und damit nicht akzeptiert

## DETERMINISTISCHE SPRACHEN SIND EINDEUTIG

- DPDAs erkennen nur eindeutige Typ-2 Sprachen
  - 1. Für jeden DPDA  $P$  hat  $L_\epsilon(P)$  eine eindeutige Grammatik  
Für DPDAs ergibt die Umwandlung eine eindeutige Typ-2 Grammatik
    - Folge der Konfigurationsübergänge bestimmt Linksableitung eindeutig
  - 2. Für jeden DPDA  $P$  hat  $L_F(P)$  eine eindeutige Grammatik  
Umwandlung in  $L_\epsilon$  – DPDA kann deterministisch gemacht werden
- Nicht jede eindeutige Typ-2 Sprache ist deterministisch
  - $\{ww^R \mid w \in \{0, 1\}^*\}$  ist eindeutig, aber nicht deterministisch erkennbar

# DPDAS SIND MÄCHTIGER ALS ENDLICHE AUTOMATEN

- $\mathcal{L}_3 = L(DEA) \subseteq L_F(DPDA)$

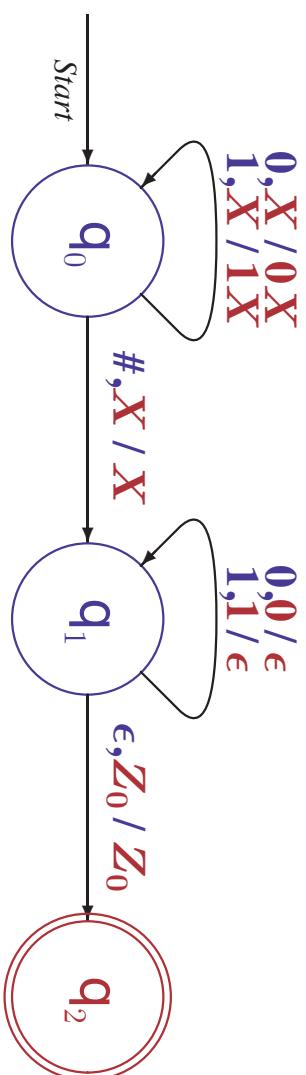
– Jeder DEA ist ein spezieller DPDA, der mit Endzustand akzeptiert

- $L = \{ww^R \mid w \in \{0, 1\}^*\} \in L_F(DPDA) - L(DEA)$

–  $L$  ist nicht regulär

· Beweis durch Pumping Lemma, analog zu  $\{ww^R \mid w \in \{0, 1\}^*\}$

–  $L = L_F(P)$  für folgenden DPDA  $P$



–  $P$  ist deterministisch, da  $\epsilon$ -Übergang in  $q_1$  genau bei Stacksymbol  $Z_0$

- $\{0\}^* \notin L_\epsilon(DPDA)$

–  $\{0\}^*$  ist nicht präfixfrei

# PUSHDOWN-AUTOMATEN – ZUSAMMENFASSUNG

- **Maschinemodell für kontextfreie Sprachen**
  - Nichtdeterministischer endlicher Automat mit Stack und  $\epsilon$ -Übergängen
  - Erkennung von Wörtern durch Endzustand oder leeren Stack
  - Erkennungsmodelle sind ineinander transformierbar
- **Verhaltensanalyse durch Konfigurationsübergänge**
  - Konfigurationen beschreiben ‘Gesamtzustand’ von Pushdown-Automaten
  - Konfigurationsübergänge verallgemeinern Überführungsfunktionen
- **Äquivalent zu kontextfreien Grammatiken**
  - Umwandlung von Konfigurationsübergängen in Regeln und umgekehrt
- **Deterministische PDAs sind weniger mächtig**
  - DPDAs erkennen nur eindeutige Typ-2 Sprachen
  - $L_\epsilon$ -DPDAs können nicht einmal alle regulären Sprachen erkennen

# Theoretische Informatik I

## Einheit 3.3

### Eigenschaften kontextfreier Sprachen



1. Abschlußeigenschaften
2. Normalformen
3. Prüfen von Eigenschaften / Syntaxanalyse
4. Wann sind Sprachen nicht kontextfrei?

# ABSCHLUSSEIGENSCHAFTEN KONTEXTFREIER SPRACHEN

## Typ-2 Sprachen sind komplizierter als Typ-3 Sprachen

- **Abgeschlossenheit gilt nur für 6 Operationen**
  - Vereinigung zweier kontextfreier Sprachen  $L_1 \cup L_2$
  - Spiegelung einer kontextfreien Sprache  $L^R$
  - Hülle einer kontextfreien Sprache  $L^*$
  - Verkettung zweier kontextfreier Sprachen  $L_1 \circ L_2$
  - Substitution/Homomorphismus einer kontextfreien Sprache  $\sigma(L)$
  - Inverser Homomorphismus einer kontextfreien Sprache  $h^{-1}(L)$
- **Keine Abgeschlossenheit für**
  - Komplement einer kontextfreien Sprache  $\overline{L}$
  - Durchschnitt zweier kontextfreier Sprachen  $L_1 \cap L_2$
  - Differenz zweier kontextfreier Sprachen  $L_1 - L_2$
- **Nachweis mit Grammatiken und PDAs**
  - Modelle sind ineinander umwandelbar – wähle das passendste
  - Negative Nachweise mit einem Typ-2 Pumping Lemma

# SUBSTITUTIONEN VON SPRACHEN

## Verallgemeinerung von Homomorphismen

- **Abbildung  $\sigma$  von Wörtern in Sprachen**

$\sigma: \Sigma^* \rightarrow \mathcal{L}$  ist **Substitution**, wenn  $\sigma(v_1..v_n) = \sigma(v_1) \circ .. \circ \sigma(v_n)$  für alle  $v_i \in \Sigma$

$\sigma(L) = \bigcup \{\sigma(w) \mid w \in L\}$  ist das Abbild der Wörter von  $L$  unter  $\sigma$

- **Beispiel:**  $\sigma(0) = \{a^n b^n \mid n \in \mathbb{N}\}$ ,  $\sigma(1) = \{aa, bb\}$ 
  - $\sigma: \{0, 1\}^* \rightarrow \mathcal{L}$  ist eindeutig definiert durch  $\sigma(0)$  und  $\sigma(1)$
  - $\sigma(01) = \{a^n b^n \mid n \in \mathbb{N}\} \circ \{aa, bb\}$
  - =  $\{w \in \{a, b\}^* \mid w = a^n b^{n+2} \vee w = a^n b^n aa \text{ für ein } n \in \mathbb{N}\}$
  - $\sigma(\{0\}^*) = \{a^n b^n \mid n \in \mathbb{N}\}^*$
  - =  $\{w \in \{a, b\}^* \mid w = a^{n_1} b^{n_1} a^{n_2} b^{n_2} .. a^{n_k} b^{n_k} \text{ für ein } k \text{ und } n_i \in \mathbb{N}\}$
- **Extrem ausdrucksstarker Mechanismus**
  - $L_1 \cup L_2 = \sigma(\{1, 2\})$  für  $\sigma(1) = L_1$ ,  $\sigma(2) = L_2$
  - $L_1 \circ L_2 = \sigma(\{1, 2\})$  für  $\sigma(1) = L_1$ ,  $\sigma(2) = L_2$
  - $L^* = \sigma(\{1\}^*)$  für  $\sigma(1) = L$

## ABGESCHLOSSENHEIT UNTER SUBSTITUTIONEN

$L \in \mathcal{L}_2, \sigma \text{ Substitution}, \sigma(a) \in \mathcal{L}_2 \text{ für } a \in T \Rightarrow \sigma(L) \text{ kontextfrei}$

- **Beweis mit Grammatiken**

“Ersetze  $a \in T$  durch Startsymbol der kontextfreien Grammatik für  $\sigma(a)$ ”

Seien  $L$  und  $\sigma(a)$  kontextfrei für alle  $a \in T$

Dann gibt es Typ-2 Grammatiken  $G = (V, T, P, S)$  mit  $L = L(G)$

und  $G_a = (V_a, T_a, P_a, S_a)$  mit  $\sigma(a) = L(G_a)$

Dann ist  $\sigma(L) = \sigma(L(G)) = \bigcup \{\sigma(a_1) \circ \dots \circ \sigma(a_n) \mid S \xrightarrow{*} a_1 \dots a_n\}$

$$= \{w_1 \dots w_n \mid \exists a_1 \dots a_n. S \xrightarrow{*} a_1 \dots a_n \wedge S_{a_i} \xrightarrow{*} w_i\}$$

Sei  $P_\sigma = \{A \xrightarrow{\alpha_\sigma} A \xrightarrow{\alpha} \alpha \in P\} \cup \bigcup_{a \in T} P_a$ , wobei  $\alpha_\sigma$  aus  $\alpha \in (V \cup T)^*$  entsteht, indem jedes  $a \in T$  durch  $S_a$  ersetzt wird

und  $G_\sigma = (V_\sigma, T_\sigma, P_\sigma, S)$  wobei  $V_\sigma = V \cup \bigcup_{a \in T} V_a$  und  $T_\sigma = \bigcup_{a \in T} T_a$

Dann gilt  $w_1 \dots w_n \in L(G_\sigma) \Leftrightarrow S \xrightarrow{*}_{G_\sigma} w_1 \dots w_n$   
 $\Leftrightarrow \exists a_1 \dots a_n \in T^*. S \xrightarrow{*}_{G_\sigma} a_1 \dots a_n \wedge S_{a_i} \xrightarrow{*}_{G_\sigma} w_i$   
 $\Leftrightarrow w_1 \dots w_n \in \sigma(L)$

Also ist  $\sigma(L)$  kontextfrei

## Verwende Abgeschlossenheit unter Substitutionen

- **$L_1, L_2$  kontextfrei  $\Rightarrow L_1 \cup L_2$  kontextfrei**
  - Sei  $\sigma(1)=L_1$  und  $\sigma(2)=L_2$
  - Dann ist  $\sigma:\{1,2\} \rightarrow \mathcal{L}_2$  Substitution und  $L_1 \cup L_2 = \sigma(\{1,2\}) \in \mathcal{L}_2$
- **$L_1, L_2$  kontextfrei  $\Rightarrow L_1 \circ L_2$  kontextfrei**
  - Sei  $\sigma(1)=L_1$  und  $\sigma(2)=L_2$
  - Dann ist  $\sigma:\{1,2\} \rightarrow \mathcal{L}_2$  Substitution und  $L_1 \circ L_2 = \sigma(\{12\}) \in \mathcal{L}_2$
- **$L$  kontextfrei  $\Rightarrow L^*$  kontextfrei**
  - Für  $\sigma(1)=L$  ist  $\sigma:\{1\} \rightarrow \mathcal{L}_2$  Substitution und  $L^* = \sigma(\{1\}^*) \in \mathcal{L}_2$
- **$L$  kontextfrei  $\Rightarrow L^+$  kontextfrei**
  - Für  $\sigma(1)=L$  ist  $\sigma:\{1\} \rightarrow \mathcal{L}_2$  Substitution und  $L^+ = \sigma(\{1\}^+) \in \mathcal{L}_2$
- **$L \in \mathcal{L}_2, h$  Homomorphismus  $\Rightarrow h(L)$  kontextfrei**
  - Für  $\sigma(a)=\{h(a)\}$  ist  $\sigma:T \rightarrow \mathcal{L}_2$  Substitution und  $h(L) = \sigma(L) \in \mathcal{L}_2$

## ABSCHLUSS UNTER SPIEGELUNG

$L$  kontextfrei  $\Rightarrow L^R = \{w_n..w_1 \mid w_1..w_n \in L\}$  kontextfrei

- **Beweis mit Grammatiken**

- Bilde **Spiegelgrammatik** zu  $G = (V, T, P, S)$  mit  $L = L(G)$
- Setze  $G_R = (V, T, P_R, S)$  mit  $P_R = \{A \xrightarrow{} \alpha^R \mid A \xrightarrow{} \alpha \in P\}$
- Dann gilt für alle  $A \in V$ ,  $w \in (V \cup T)^*$ :  $A \xrightarrow{*} G w \Leftrightarrow A \xrightarrow{*} G_R w^R$ 
  - Beweis durch Induktion über Länge der Ableitung
- Also  $L(G_R) = \{w \in T^* \mid S \xrightarrow{*} G_R w\} = \{v^R \in T^* \mid S \xrightarrow{*} G v\} = (L(G))^R$

- **Beweis mit PDAs ähnlich wie bei Typ-3 Sprachen**

- Bilde Umkehrautomaten zu  $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  mit  $L = L_F(P)$

# ABSCHLUSS UNTER INVERSEN HOMOMORPHISMEN

$L \in \mathcal{L}_2, h$  Homomorphismus  $\Rightarrow h^{-1}(L)$  kontextfrei

## • Beweis mit Pushdown Automaten

“Berechnung von  $h$  vor Abarbeitung der Wörter im Automaten”

Sei  $L$  kontextfrei und  $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  ein PDA

mit  $L = L_F(P) = \{ v \in \Sigma^* \mid \exists q \in F. \exists \beta \in \Gamma^*. (q_0, v, Z_0) \vdash^* (q, \epsilon, \beta) \}$

Dann ist  $h^{-1}(L) = \{ w \in \Sigma'^* \mid \exists q \in F. \exists \beta \in \Gamma^*. (q_0, h(w), Z_0) \vdash^* (q, \epsilon, \beta) \}$

Konstruiere PDA  $P_h = (Q_h, \Sigma', \Gamma, \delta_h, q_{0_h}, Z_0, F_h)$  mit der Eigenschaft

$(q_{0_h}, w, Z_0) \vdash^* (q_h, \epsilon, \beta) \Leftrightarrow (q_0, h(w), Z_0) \vdash^* (q, \epsilon, \beta)$  für Endzustände

Ein Ansatz wie  $\delta_h(q, a, X) = \hat{\delta}(q, h(a), X)$  funktioniert nicht!

Wie bei DEAs muß  $h(a)$  schrittweise in den Zuständen abgearbeitet werden

Setze  $Q_h = Q \times \{ v \in \Sigma^* \mid v \text{ Suffix von } h(a) \text{ für ein } a \in \Sigma' \}$

$\delta_h((q, \epsilon), a, X) = \{ ((q, h(a)), X) \}$

$\delta_h((q, bv), \epsilon, X) = \{ ((p, v), \alpha) \mid (p, \alpha) \in \delta(q, b, X) \} \quad b \in \Sigma \cup \{\epsilon\}, v \in \Sigma^*, X \in \Gamma$

$q_{0_h} = (q_0, \epsilon)$   $F_h = \{ (q, \epsilon) \mid q \in F \}$

Dann gilt  $((q, \epsilon), a, X) \vdash_{P_h}^* ((p, \epsilon), \epsilon, \beta) \Leftrightarrow (q, h(a), X) \vdash_P^* (p, \epsilon, \beta)$

Also ist  $h^{-1}(L) = L(P_h)$  und damit kontextfrei

# DURCHSCHNITT, KOMPLEMENT UND DIFFERENZ

## Abgeschlossenheit gilt **nicht** für diese Operationen

- **Durchschnitt:**  $L_1, L_2 \in \mathcal{L}_2 \not\Rightarrow L_1 \cap L_2 \in \mathcal{L}_2$ 
  - $\underline{L} = \{0^n 1^n 2^n \mid n \in \mathbb{N}\}$  ist nicht kontextfrei  
(Beweis später)
  - Aber  $L = \{0^n 1^n 2^m \mid n, m \in \mathbb{N}\} \cap \{0^m 1^n 2^n \mid n, m \in \mathbb{N}\}$  und  $\{0^n 1^n 2^m \mid n, m \in \mathbb{N}\}$  sind kontextfrei  
(Regeln für erste Sprache:  $S \rightarrow AB, A \rightarrow 0A1, A \rightarrow 01, B \rightarrow 2B, B \rightarrow 2$ )
- Der Durchschnitt kontextfreier und regulärer Sprachen ist kontextfrei  
(HMU Satz 7.27)
- **Komplement**  $\underline{L} \in \mathcal{L}_2 \not\Rightarrow \overline{L} \in \mathcal{L}_2$ 
  - Es ist  $\underline{L}_1 \cap \underline{L}_2 = \overline{\overline{L}_1 \cup \overline{L}_2}$
  - Bei Abgeschlossenheit unter Komplementbildung würde Abgeschlossenheit unter Durchschnitt folgen
- **Differenz:**  $L_1, L_2 \in \mathcal{L}_2 \not\Rightarrow L_1 - L_2 \in \mathcal{L}_2$ 
  - Es ist  $\overline{\underline{L}} = \Sigma^* - \underline{L}$
  - Aus Abschluß unter Differenz folgt Abschluß unter Komplement

# TESTS FÜR EIGENSCHAFTEN KONTEXTFREIER SPRACHEN

## Welche Eigenschaften sind automatisch prüfbar?

- Ist eine kontextfreie Sprache leer?
  - Entspricht Test auf Erreichbarkeit von Endzuständen
  - Nicht ganz so einfach, da Stackinhalt die Erreichbarkeit beeinflusst
- **Zugehörigkeit: gehört ein Wort zur Sprache?**
  - Verarbeitung durch Pushdown-Automaten ist nichtdeterministisch
  - Deterministische Pushdown-Automaten sind nicht mächtig genug
  - Frage nach Zugehörigkeit beinhaltet oft Frage nach Ableitungsbaum
- **Äquivalenz: sind zwei Typ-2 Sprachen identisch?**
  - Zusammenfassen äquivalenter Zustände im PDA kaum durchführbar
- **Kontextfreie Grammatiken sind zu kompliziert**
  - Analyse braucht einfachere Versionen von Typ-2 Grammatiken
  - Bringt Grammatik auf “**Normalform**” (äquivalente einfachere Struktur)

# DIE CHOMSKY NORMALFORM

## Trenne Variablen von Terminalsymbolen

- **Grammatik in Chomsky-Normalform**

- Grammatik  $G = (V, T, P, S)$ , bei der jede Produktion die Form  
 $A \rightarrow B C$  oder  $A \rightarrow a$  hat  
( $A, B, C \in V, a \in T$ )
- Grammatiken in Chomsky Normalform sind auch kontextsensitiv

- **Jede kontextfreie Grammatik  $G$  mit  $\epsilon \notin L(G)$  ist in Chomsky-Normalform transformierbar**

1. Eliminierung von  $\epsilon$ -Produktionen  $A \rightarrow \epsilon$
  2. Eliminierung von Einheitsproduktionen  $A \rightarrow B$
  3. Eliminierung unnützer Symbole
  4. Separieren von Terminalsymbolen und Variablen in Produktionen
  5. Aufspalten von Produktionen  $A \rightarrow \alpha$  mit  $|\alpha| > 2$
- Aufzählung/Transformationszeit quadratisch relativ zur Größe von  $G$

## $\epsilon$ -PRODUKTIONEN ELIMINIEREN

- **$\epsilon$ -Produktionen sind überflüssig, falls  $\epsilon \notin L(G)$** 
  - Variablen  $A \in V$  mit  $A \xrightarrow{*} \epsilon$  sind **eliminierbar**
  - Menge eliminierbarer Symbole kann iterativ bestimmt werden
    - Ist  $A \rightarrow \epsilon \in P$  dann ist  $A$  eliminierbar
    - Ist  $A \rightarrow X_1..X_n \in P$  und alle  $X_i$  eliminierbar, dann ist  $A$  eliminierbar
  - Verfahren terminiert nach maximal  $|V| + 1$  Iterationen
- **Erzeuge Grammatik ohne eliminierbare Symbole**
  - Für  $G = (V, T, P, S)$  bestimme alle eliminierbare Variablen
  - Für  $A \rightarrow \alpha \in P$  mit eliminierbaren Symbolen  $X_1, .., X_m$  in  $\alpha$  erzeuge  $2^m$  Regeln  $A \rightarrow \alpha_{i_1,..,i_k}$  ( $\{i_1, .., i_k\}$  Teilmenge von  $\{1, .., m\}$ )
  - Entferne alle Regeln der Form  $A \rightarrow \epsilon$  (auch neu erzeugte)
  - Wenn  $S$  eliminierbar ist, kann  $S' \rightarrow S$  und  $S' \rightarrow \epsilon$  ergänzt werden
- **Erzeugte Grammatik ist äquivalent**
  - Zeige  $A \xrightarrow[G]{*} w \Leftrightarrow A \xrightarrow{G} w \wedge (w \neq \epsilon \vee A = S')$  durch Induktion über Länge der Ableitung

## ELIMINATION VON $\epsilon$ -PRODUKTIONEN AM BEISPIEL

$$P = \{ S \rightarrow AB, A \rightarrow aAA \mid \epsilon, B \rightarrow bBB \mid \epsilon \}$$

- **Ermittlung eliminierbarer Symbole**

- 1.:  $A$  und  $B$  sind eliminierbar
- 2.:  $S$  ist ebenfalls eliminierbar

- **Verändere Regeln der Grammatik**

- Aus  $S \rightarrow AB$  wird  $S \rightarrow AB \mid A \mid B$
- Aus  $A \rightarrow aAA \mid \epsilon$  wird  $A \rightarrow aAA \mid aA \mid a$
- Aus  $B \rightarrow bBB \mid \epsilon$  wird  $B \rightarrow bBB \mid bB \mid b$

**Grammatik erzeugt  $L(G) - \{\epsilon\}$  ohne  $\epsilon$ -Produktionen**

- **Ergänze neues Startsymbol**

- $S'$  war eliminierbar: ergänze Produktionen  $S' \rightarrow S \mid \epsilon$

**Grammatik erzeugt  $L(G)$  mit initialer  $\epsilon$ -Produktion**

## EINHEITSPRODUKTIONEN ELIMINIEREN

Einheitsproduktionen verlängern Ableitungen  
und verkomplizieren technische Beweise

- Bestimme alle **Einheitspaare**  $(A,B)$  mit  $A \xrightarrow{*} B$ 
  - Wie üblich ... iteratives Verfahren:
    - Alle Paare  $(A,A)$  für  $A \in V$  sind Einheitspaare
    - Ist  $(A,B)$  Einheitspaar und  $B \xrightarrow{} C \in P$  dann ist  $(A,C)$  Einheitspaar
  - Verfahren terminiert nach maximal  $|V| + 1$  Iterationen
- Erzeuge Grammatik ohne Einheitsproduktionen  $A \rightarrow B$ 
  - Bestimme alle Einheitspaare in  $G$
  - Für jedes Einheitspaar  $(A,B)$  erzeuge Produktionen
    - $\{ A \xrightarrow{\alpha} | B \xrightarrow{\alpha} \in P \text{ keine Einheitsproduktion} \}$
- Erzeugte Grammatik ist äquivalent
  - Ableitungen in  $G'$  sind “Kurzformen” von Ableitungen in  $G$
  - Beweis, wie immer, durch Induktion über Länge der Ableitung

## ELIMINATION VON EINHEITSPRODUKTIONEN AM BEISPIEL

$$P' = \{ \begin{array}{l} E \rightarrow T \mid E+T, \quad T \rightarrow F \mid T*F, \quad F \rightarrow I \mid (E) \\ I \rightarrow a \mid b \mid c \mid Ia \mid Ib \mid Ic \mid I0 \mid I1 \end{array} \}$$

### • Bestimme alle Einheitspaare ( $A, B$ ) mit $A \xrightarrow{*} B$

- 1.:  $(E,E), (T,T), (F,F)$  und  $(I,I)$  sind Einheitspaare
- 2.:  $(E,T), (T,F)$  und  $(F,I)$  sind ebenfalls Einheitspaare
- 3.:  $(E,F)$  und  $(T,I)$  sind ebenfalls Einheitspaare
- 4.:  $(E,I)$  ist ebenfalls Einheitspaar
- 5.: Keine weiteren Einheitspaare möglich

### • Erzeuge Grammatik ohne Einheitsproduktionen

- Einheitspaare mit  $E$ :  $\{E \rightarrow E+T \mid T*F \mid (E) \mid a \mid b \mid c \mid Ia \mid Ib \mid Ic \mid I0 \mid I1\}$
- Einheitspaare mit  $T$ :  $\{T \rightarrow T*F \mid (E) \mid a \mid b \mid c \mid Ia \mid Ib \mid Ic \mid I0 \mid I1\}$
- Einheitspaare mit  $F$ :  $\{F \rightarrow (E) \mid a \mid b \mid c \mid Ia \mid Ib \mid Ic \mid I0 \mid I1\}$
- Einheitspaare mit  $I$ :  $\{I \rightarrow a \mid b \mid c \mid Ia \mid Ib \mid Ic \mid I0 \mid I1\}$

## UNNÜTZE SYMBOLE ELIMINIEREN

- **X nützlich, falls  $S \xrightarrow{*} \alpha X \beta \xrightarrow{*} w \in T^*$** 
  - **Erzeugend** ( $X \xrightarrow{*} v \in T^*$ ) und **erreichbar** ( $S \xrightarrow{*} \alpha X \beta$ )
- **Beispiel:  $P = \{ S \xrightarrow{} AB \mid a, A \xrightarrow{} b \}$** 
  - Erreichbar:  $S, A, B, a$ , und  $b$  erreichend:  $S, A, a$ , und  $b$
  - Nach Elimination von  $B$ :  $\{ S \xrightarrow{} a, A \xrightarrow{} b \}$
  - Erreichbar:  $S$  und  $a$  erreichend:  $S, A, a$ , und  $b$
  - Nach Elimination von  $A$ :  $\{ S \xrightarrow{} a \}$
  - Erreichbar:  $S$  und  $a$  erreichend:  $S$  und  $a$
- Erzeugte Produktionenmenge ist äquivalent zu  $P$
- **Eliminationsverfahren für  $G$  mit  $L(G) \neq \emptyset$** 
  - Eliminiere nichterzeugende Symbole und Produktionen, die sie enthalten
  - Eliminiere unerreichbare Symbole und Produktionen, die sie enthalten
- Resultierende Grammatik  $G'$  erzeugt dieselbe Sprache wie  $G$** 
  - $G'$  enthält nur nützliche Symbole und  $S \in V'$
  - Also  $w \in L(G) \Leftrightarrow S \xrightarrow{*}_G w \Leftrightarrow S \xrightarrow{*}_{G'} w \Leftrightarrow w \in L(G')$

# BERECHNUNG ERZEUGENDER / ERREICHBARER SYMBOLE

- **Generiere Menge erzeugender Symbole iterativ**
  - Alle Terminalsymbole  $a \in T$  sind erzeugend
  - Ist  $A \rightarrow X_1..X_n \in P$  und alle  $X_i$  erzeugend, dann ist  $A$  erzeugend
  - Verfahren terminiert nach maximal  $|V| + 1$  Iterationen
- **Generiere Menge erreichbarer Symbole iterativ**
  - $S$  ist erreichbar
  - Ist  $A \rightarrow X_1..X_n \in P$  und  $A$  erreichbar dann sind alle  $X_i$  erreichbar
  - Verfahren terminiert nach maximal  $|V| + |T|$  Iterationen
- **Beispiel:**  $P = \{ S \rightarrow AB \mid a, A \rightarrow b \}$ 
  - Erzeugende Symbole: 1.:  $a$  und  $b$  sind erzeugend
  - 2.:  $S$  und  $A$  sind ebenfalls erzeugend
  - 3.: Keine weiteren Symbole sind erzeugend
  - Erreichbare Symbole: 1.:  $S$  ist erreichbar
  - 2.:  $A, B$  und  $a$  sind ebenfalls erreichbar
  - 3.:  $b$  ist ebenfalls erreichbar

# ERZEUGUNG DER CHOMSKY-NORMALFORM

## Nur Produktionen der Form $A \rightarrow BC$ oder $A \rightarrow a$

- Jede kontextfreie Grammatik  $G$  ist umwandelbar in eine äquivalente Grammatik ohne unnütze Symbole, (echte)  $\epsilon$ -Produktionen und Einheitsproduktionen

– Falls  $L(G) = \emptyset$ , wähle  $G' = (V, T, \emptyset, S)$  (Test auf  $\emptyset$  später)

– Sonst eliminiere  $\epsilon$ -Produktionen, Einheitsproduktionen, unnütze Symbole

### • Separiere Terminalsymbole von Variablen

- Für jedes Terminalsymbol  $a \in T$  erzeuge neue Variable  $X_a$
- Ersetze Produktionen  $A \rightarrow \alpha$  mit  $|\alpha| \geq 2$  durch  $A \rightarrow \alpha_X$  ( $a \in T$  ersetzt durch  $X_a$ )
- Ergänze Produktionen  $X_a \rightarrow a$  für alle  $a \in T$

### • Spalte Produktionen $A \rightarrow \alpha$ mit $|\alpha| > 2$

- Ersetze jede Produktion  $A \rightarrow X_1..X_k$  durch  $k-1$  Produktionen

$A \rightarrow X_1 Y_1, Y_1 \rightarrow X_2 Y_2, \dots Y_{k-2} \rightarrow X_{k-1} X_k$ , wobei alle  $Y_i$  neue Variablen

# ERZEUGUNG DER CHOMSKY-NORMALFORM AM BEISPIEL

$$\begin{aligned}
 P = \{ & E \rightarrow E + T \mid T * F \mid (E) \mid a \mid b \mid c \mid Ia \mid Ib \mid Ic \mid I0 \mid I1 \\
 & T \rightarrow T * F \mid (E) \mid a \mid b \mid c \mid Ia \mid Ib \mid Ic \mid I0 \mid I1 \\
 & F \rightarrow (E) \mid a \mid b \mid c \mid Ia \mid Ib \mid Ic \mid I0 \mid I1 \\
 & I \rightarrow a \mid b \mid c \mid Ia \mid Ib \mid Ic \mid I0 \mid I1 \}
 \end{aligned}$$

- Separiere Terminalsymbole von Variablen

$$\begin{aligned}
 P' = \{ & E \rightarrow EX_+T \mid TX_*F \mid X(EX) \mid a \mid b \mid c \mid IX_a \mid IX_b \mid IX_c \mid IX_0 \mid IX_1 \\
 & T \rightarrow TX_*F \mid X(EX) \mid a \mid b \mid c \mid IX_a \mid IX_b \mid IX_c \mid IX_0 \mid IX_1 \\
 & F \rightarrow X(EX) \mid a \mid b \mid c \mid IX_a \mid IX_b \mid IX_c \mid IX_0 \mid IX_1 \\
 & I \rightarrow a \mid b \mid c \mid IX_a \mid IX_b \mid IX_c \mid IX_0 \mid IX_1 \\
 & X_a \rightarrow a, X_b \rightarrow b, X_c \rightarrow c, X_0 \rightarrow 0, X_1 \rightarrow 1, X_+ \rightarrow +, X_* \rightarrow *, X_{(\rightarrow (, X) \rightarrow )} \}
 \end{aligned}$$

- Spalte Produktionen  $A \rightarrow \alpha$  mit  $|\alpha| > 2$

$$\begin{aligned}
 P' = \{ & E \rightarrow EY_1 \mid TY_2 \mid X(Y_3) \mid a \mid b \mid c \mid IX_a \mid IX_b \mid IX_c \mid IX_0 \mid IX_1 \\
 & T \rightarrow TY_2 \mid X(Y_3) \mid a \mid b \mid c \mid IX_a \mid IX_b \mid IX_c \mid IX_0 \mid IX_1 \\
 & F \rightarrow X(Y_3) \mid a \mid b \mid c \mid IX_a \mid IX_b \mid IX_c \mid IX_0 \mid IX_1 \\
 & I \rightarrow a \mid b \mid c \mid IX_a \mid IX_b \mid IX_c \mid IX_0 \mid IX_1 \\
 & Y_1 \rightarrow X_+T, Y_2 \rightarrow X_*F, Y_3 \rightarrow EX \\
 & X_a \rightarrow a, X_b \rightarrow b, X_c \rightarrow c, X_0 \rightarrow 0, X_1 \rightarrow 1, X_+ \rightarrow +, X_* \rightarrow *, X_{(\rightarrow (, X) \rightarrow )} \}
 \end{aligned}$$

# TESTS FÜR EIGENSCHAFTEN KONTEXTFREIER SPRACHEN

- Ist eine kontextfreie Sprache leer?

- Für  $G = (V, T, P, S)$  gilt
  - $L(G)$  ist leer genau dann wenn  $S$  nicht erzeugend ist**
  - Menge erzeugender Variablen kann iterativ bestimmt werden
  - Mit speziellen Datenstrukturen ist Test in linearer Zeit durchführbar

(Details ins HMU §7.4.3)

- Gehört ein Wort zu einer kontextfreien Sprache?

- Naive Methode für den Test  $w \in L(G)$ :
  1. Erzeuge Chomsky-Normalform  $G'$  von  $G$
  2. In  $G'$  erzeuge alle Ableitungsbäume mit  $2|w| - 1$  Variablenknoten
  3. Teste, ob einer dieser Bäume das Wort  $w$  erzeugt
- Hochgradig ineffizient, da exponentiell viele Bäume zu erzeugen
- Iterative Analyseverfahren sind besser

# SYNTAXANALYSE: COCKE-YOUNGER-KASAMI ALGORITHMUS

Bestimme Variablenmengen, aus denen  $w_i..w_j$  ableitbar

- Eingabe: Grammatik  $G = (V, T, P, S)$  in Chomsky-NF,  $w \in T^*$

- Berechne Mengen  $V_{i,j} = \{A \in V \mid A \xrightarrow{*} w_i \dots w_j\}$  iterativ

$j=i:$ $V_{i,i} = \{A \in V \mid A \xrightarrow{*} w_i \in P\}$	$V_{1,n}$
$j > i:$ $V_{i,j} = \{A \in V \mid$	$V_{1,n-1} \quad V_{2,n}$
	$\vdots \quad \vdots$
$\exists i \leq k < j:$	$V_{1,2} \quad V_{2,3} \dots \quad V_{n-1,n}$
$\exists A \xrightarrow{*} BC \in P.$	$V_{1,1} \quad V_{2,2} \dots \quad V_{n-1,n-1} \quad V_{n,n}$
$B \in V_{i,k} \wedge C \in V_{k+1,j}\}$	$w_1 \quad w_2 \dots \quad w_{n-1} \quad w_n$

- Akzeptiere  $w$  genau dann, wenn  $S \in V_{1,|w|}$

---

Entscheidet  $w \in L(G)$  in kubischer Zeit relativ zur Größe von  $w$   
Konstruiert gleichzeitig den Syntaxbaum von  $w$

## DER CYK-ALGORITHMUS AM BEISPIEL

$\{ S \rightarrow AB | BC, A \rightarrow BA | a, B \rightarrow CC | b, C \rightarrow AB | a \}$

- Prüfe  $w = baaba \in L(G)$

- Berechne  $V_{i,j} = \{A \in V \mid A \xrightarrow{*} w_i \dots w_j\}$

$\{S, A, C\}$				
$\vdash$	$\{S, A, C\}$			
$\{S, A\}$	$\{B\}$	$\{B\}$		
$\{B\}$	$\{B\}$	$\{S, C\}$	$\{S, A\}$	
$b$	$a$	$a$	$b$	$a$

- $S \in V_{1,5}$ , also  $w \in L(G)$

# UNENTSCHEIDBARE PROBLEME FÜR TYP-2 SPRACHEN

Die folgenden Probleme können nicht getestet werden

- $L(G) = T^*$
- $L(G_1) = L(G_2)$
- $L(G_1) \subseteq L(G_2)$
- $L(G_1) \cap L(G_2) = \emptyset$
- $L(G) \in \mathcal{L}_3$
- $\overline{L(G)} \in \mathcal{L}_2$
- $L(G_1) \cap L(G_2) \in \mathcal{L}_2$

Äquivalenz von Grammatiken

Welche Menge beschreibt  $G$ ?

Beweise brauchen Berechenbarkeitstheorie / Th2

## Warum ist $L = \{0^n 1^{n2^n} \mid n \in \mathbb{N}\}$ nicht kontextfrei?

- **Typ-2 Grammatiken arbeiten lokal**

- Anwendbarkeit einer Produktion hängt nur von einer Variablen ab  
(der Kontext der Variablen ist irrelevant)

- Eine Regel kann nur an einer Stelle im Wort etwas erzeugen
- Eine Typ-2 Grammatik kann entweder  $0/1$  oder  $1/2$  simultan erhöhen aber nicht beides gleichzeitig

- Grammatik müßte die Anzahl der  $0/1$  oder  $1/2$  im Voraus bestimmen und diese Anzahl für die  $2$  bzw.  $0$  im Namen der Variablen codieren

- **Grammatiken sind endlich**

- Es gibt nur endlich viele Variablen
- Für  $n > |V|$  muß eine Variable  $X$  doppelt benutzt worden sein zur Codierung von  $0^n 1^n$  und  $0^i 1^i$  mit  $i < n$
- Grammatik würde auch  $0^n 1^{n2^n}$  und  $0^i 1^{i2^m}$  generieren

- **Genaues Argument ist etwas komplizierter**

- Allgemeine Version: **Pumping Lemma** für kontextfreie Sprachen

# DAS PUMPING LEMMA FÜR KONTEXTFREIE SPRACHEN

Wie zeigt man, daß eine Sprache nicht kontextfrei ist?

- Für jede kontextfreie Sprache  $L \in \mathcal{L}_2$  gibt es eine Zahl  $n \in \mathbb{N}$ , so daß jedes Wort  $z \in L$  mit Länge  $|z| \geq n$  zerlegt werden kann in  $z = u v w x y$  mit den Eigenschaften
  - (1)  $v \circ x \neq \epsilon$ ,
  - (2)  $|v w x| \leq n$  und
  - (3) für alle  $i \in \mathbb{N}$  ist  $u v^i w x^i y \in L$
- Aussage ist wechselseitig konstruktiv
  - Die Zahl  $n$  kann zu jeder kontextfreien Sprache  $L$  bestimmt werden
  - Die Zerlegung  $z = u v w x y$  kann zu jedem Wort  $z \in L$  bestimmt werden
- Beweis benötigt Chomsky-Normalform
  - Ableitungen der Länge  $k$  können maximal Wörter der Länge  $2^k$  erzeugen
  - Ableitungen der Länge  $k > |V|$  benutzen ein Hilfssymbol  $X$  doppelt
  - Die Schleife der Ableitung von  $X$  aus  $X$  kann beliebig wiederholt werden

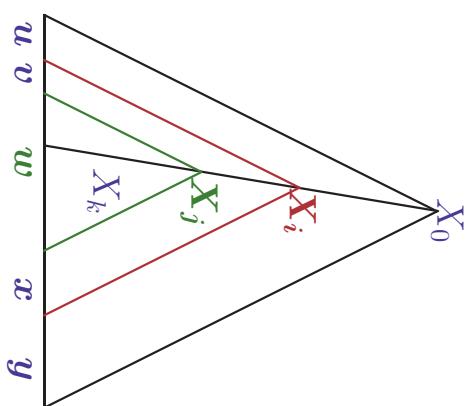
## BEWEIS DES PUMPING LEMMAS

Für jede Sprache  $L \in \mathcal{L}_2$  gibt es ein  $n \in \mathbb{N}$ , so daß jedes  $z \in L$  mit Länge  $|z| \geq n$  zerlegt werden kann in  $z = u v w x y$  mit

- (1)  $v o x \neq \epsilon$ ,
- (2)  $|v w x| \leq n$
- (3)  $u v^i w x^i y \in L$  für alle  $i \in \mathbb{N}$

### Beweis mit Grammatiken in Chomsky-Normalform

- Für  $L = \emptyset$  oder  $L = \{\epsilon\}$  gilt die Behauptung trivialerweise
- Andernfalls sei  $G = (V, T, P, S)$  in Chomsky-Normalform mit  $L = L(G)$
- Wähle  $n = 2^{|V|}$  und betrachte  $z = z_1 \dots z_m$  mit  $|z| \geq n$
- Dann hat jeder Ableitungsbaum für  $z$  eine Tiefe von mindestens  $|V| + 1$
- Sei  $X_0, \dots, X_k$  die Folge der verarbeiteten Variablen auf dem längsten Pfad
- Dann erscheint eine Variable zweimal:  $X_i = X_j$  für ein  $i < j$  mit  $k - |V| < i - j$
- Seien  $w$  und  $t$  die aus  $X_j$  bzw.  $X_i$  abgeleiteten Teilwörter
- Dann gilt  $t = v w x$  und  $z = u t y$  für Wörter  $u, v, x$  und  $y$
- Da  $G$  in Chomsky-Normalform ist, gilt  $v o x \neq \epsilon$
- Wegen  $k - |V| < i$  gilt  $|v w x| = |t| \leq n$
- Wegen  $X_i = X_j$  kann die Ableitung von  $X_i$  bis  $X_j$  beliebig wiederholt werden und es gilt  $u v^i w x^i y \in L$  für alle  $i \in \mathbb{N}$



## ANWENDUNGEN DES PUMPING LEMMAS

- $L = \{0^m 1^m 2^m \mid m \in \mathbb{N}\}$  ist nicht kontextfrei
  - Verwende Kontraposition des Pumping Lemmas
  - $$(\forall n \in \mathbb{N}. \exists z \in L. |z| \geq n \wedge \forall u, v, w, x, y \in T^*. (z = u v w x y \wedge v o x \neq \epsilon \wedge |v w x| \leq n) \Rightarrow \exists i \in \mathbb{N}. u v^i w x^i y \notin L) \Rightarrow L \notin \mathcal{L}_2$$
  - Sei  $n \in \mathbb{N}$  beliebig. Wir wählen  $z = 0^m 1^m 2^m$  für ein  $m > n$
  - Sei  $u, v, w, x, y \in T^*$  beliebig mit  $z = u v w x y$ ,
  - und (1)  $v o x \neq \epsilon$  und (2)  $|v w x| \leq n$
  - Wir wählen  $i = 0$  und zeigen  $u w y = u v^i w x^i y \notin L$
  - Wegen (2) enthält  $v w x$  keine Nullen oder keine Zweien
  - . Falls  $v w x$  keine Null enthält, dann enthält  $u w y$  genau  $m$  Nullen aber wegen (1) weniger Einsen und/oder Zweien
  - . Falls  $v w x$  keine Zwei enthält, dann enthält  $u w y$  genau  $m$  Zweien aber wegen (1) weniger Nullen und/oder Einsen
  - Damit kann  $u w y = u v^0 w x^0 y$  nicht zu  $L$  gehören
  - Mit dem Pumping Lemma folgt nun, daß  $L$  nicht kontextfrei ist
- $L' = \{ww \mid w \in \{0, 1\}^*\} \not\subseteq \mathcal{L}_2$ 
  - Ähnliches Argument mit Wörtern der Form  $0^m 1^m 0^m 1^m$

# RÜCKBLICK: EIGENSCHAFTEN KONTEXTFREIER SPRACHEN

## Kontextfreie Sprachen sind deutlich komplizierter

- **Abschlußeigenschaften**

- Operationen  $\cup$ ,  $\overset{R}{\cup}$ ,  $\circ$ ,  ${}^*$ ,  $\sigma$ ,  $h^{-1}$  erhalten Kontextfreiheit von Sprachen
- Keine Abgeschlossenheit unter  $\cap$ ,  $-$ ,  $\neg$

- **Automatische Prüfungen**

- Man kann testen ob eine kontextfreie Sprache leer ist
- Man kann testen ob ein Wort zu einer kontextfreien Sprache gehört
- Man kann nicht testen ob zwei kontextfreie Sprachen gleich sind
- Viele wichtige Fragen sind nicht automatisch prüfbar

- **Pumping Lemma**

- Wiederholt man bestimmte Teile genügend großer Wörter einer kontext-freien Sprache beliebig oft, so erhält man immer ein Wort der Sprache
- Konsequenz: viele einfache Sprachen sind nicht kontextfrei
- Für diese sind aufwendigere Mechanismen erforderlich  
    → Th2

# Theoretische Informatik I

## Einheit 4

### Allgemeine und kontextsensitive Sprachen



1. Turingmaschinen
2. Maschinenmodelle für  $\mathcal{L}_0$  und  $\mathcal{L}_1$
3. Eigenschaften von  $\mathcal{L}_0/\mathcal{L}_1$ -Sprachen



# JENSEITS VON KONTEXTFREIHEIT

- **Viele wichtige Konzepte sind nicht kontextfrei**

- Sind Bezeichner im Programmkörper deklariert?
- $\{ww \mid w \in \{0,1\}^*\}$ : erscheint Programmcode doppelt?
- $\{0^n1^n2^n \mid n \in \mathbb{N}\}$ : kommen mehrere Bestandteile gleich oft vor?
- Zählen jenseits von Addition und Multiplikation

- **Wie verarbeitet man Typ-1 / Typ-0 Sprachen?**

- Welches Maschinenmodell ist zur Beschreibung geeignet?
- Wie analysiert man Wörter der Sprache
- Wie kann man Sprachen aus Bausteinen zusammensetzen?
- Welche Spracheigenschaften kann man testen?

# Theoretische Informatik I

## Einheit 4.1

### Turingmaschinen



1. Das Maschinenmodell
2. Arbeitsweise & erkannte Sprache
3. Programmiertechniken
4. Ausdruckskraft



## Maschinenmodell für Typ-0 Sprachen

- **Erweiterung des Konzepts endlicher Automaten**

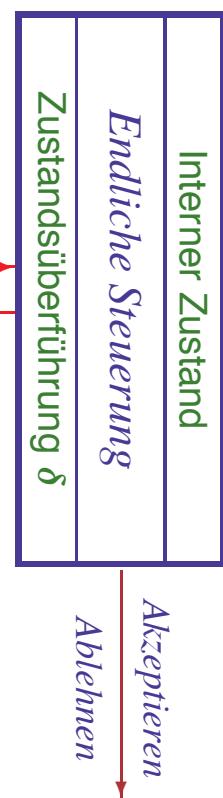
- Verarbeitung interner **Zustände** abhängig von gelesenen Daten
- Lese- und Schreibzugriff auf externen **Speicher**
- Minimal mögliche Erweiterung

- **Maximal mögliche Ausdruckskraft**

- Speicher muß Fähigkeiten von Typ-0 Grammatiken widerspiegeln
  - Keine Einschränkung an Ersetzungsregeln
- Auch Terminalsymbole und ganze Wörter dürfen ersetzt werden
  - Automat muß **Eingabe** an jeder **Stelle** verarbeiten können
  - Gesamte Eingabe muß gespeichert werden
  - Speicher muß Veränderungen an jeder Stelle zulassen
  - Speicher muß beliebig erweiterbar sein

## Wähle unendliches, bewegliches Band als Speicher

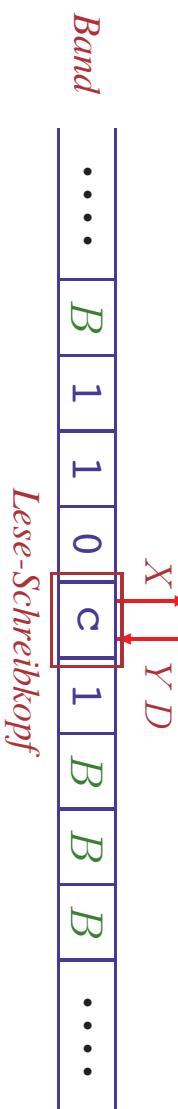
# TURINGMASCHINEN INTUITIV



- **Endlicher Automat + lineares Band**
  - Endliche Steuerung liest Eingabesymbole
  - Gleichzeitig wird Bandsymbol unter **Lese-Schreibkopf** gelesen
- **Vereinfachung: keine separate Eingabe**
  - Eingabewort steht zu Anfang bereits auf dem Band
- **Einfacher Verarbeitungsmechanismus**
  - Bandsymbol **X** wird gelesen
  - Interner Zustand **q** wird zu **q'** verändert
  - Neues Symbol **Y** wird auf das Band geschrieben
  - Kopf wird in eine Richtung **D** (rechts oder links) bewegt

# TURINGMASCHINEN – MATHEMATISCH PRÄZISIERT

Interner Zustand	$\xrightarrow{\text{Akzeptieren}}$ $\xrightarrow{\text{Ablehnen}}$
Endliche Steuerung	
Zustandsüberführung $\delta$	



Eine **Turingmaschine (TM)** ist ein 7-Tupel

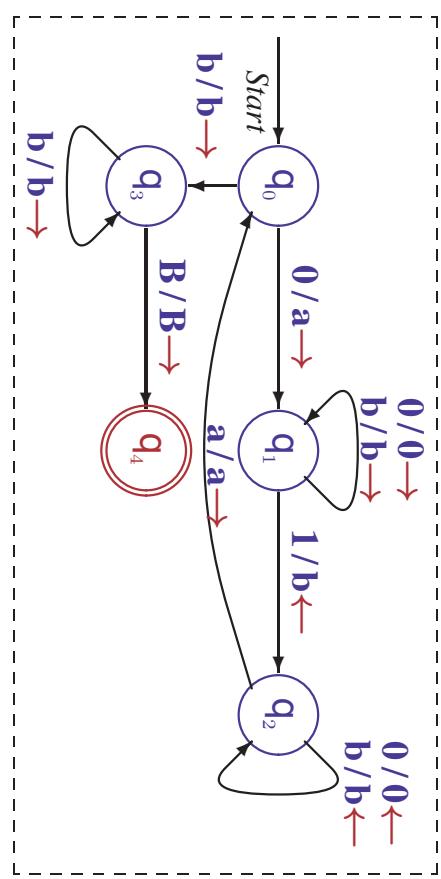
$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$
 mit

- $Q$  nichtleere endliche **Zustandsmenge**
- $\Sigma$  endliches **Eingabealphabet**
- $\Gamma \supseteq \Sigma$  endliches **Bandalphabet**
- $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$  (partielle) **Überführungsfunktion**
- $q_0 \in Q$  **Startzustand**
- $B \in \Gamma \setminus \Sigma$  **Leersymbol des Bands**  
(“blank”)
- $F \subseteq Q$  Menge von **akzeptierenden (End-)Zuständen**

# BESCHREIBUNG VON TURINGMASCHINEN

- **Übergangsdiagramme**

- Zustände durch Knoten dargestellt
- $q_0$  markiert durch *Start*-Pfeil,
- Endzustände durch doppelte Kreise
- Für  $\delta(q, X) = (p, Y, D)$  hat das Diagramm eine Kante  $q \xrightarrow{X/YD} p$
- $\Sigma$  und  $\Gamma$  implizit durch Diagramm bestimmt, LeerSymbol heißt  $B$



- **Übergangstabellen**

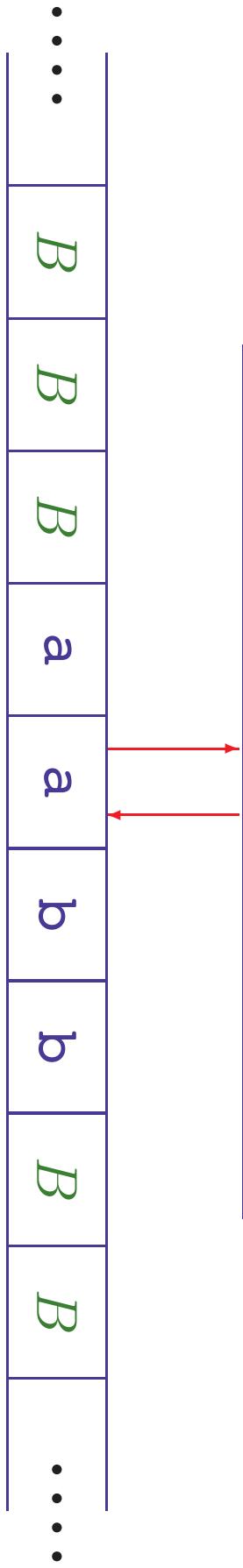
- Funktionstabelle für  $\delta$ 
  - heißt “ $\delta$  nicht definiert”,
- Pfeil  $\rightarrow$  kennzeichnet  $q_0$
- Stern \* kennzeichnet  $F$
- $\Sigma$ ,  $\Gamma$  und  $B$  implizit bestimmt

$Q \setminus \Gamma$	0	1	a	b	B
$\rightarrow q_0$	$(q_1, a, R)$	—	—	$(q_3, b, R)$	—
$q_1$	$(q_1, 0, R)$	$(q_2, b, L)$	—	$(q_1, b, R)$	—
$q_2$	$(q_2, 0, L)$	—	$(q_0, a, R)$	$(q_2, b, L)$	—
$q_3$	—	—	—	$(q_3, b, R)$	$(q_4, B, R)$
* $q_4$	—	—	—	—	—

# ABARBEITUNG VON TURING-PROGRAMMEN

$Q \setminus \Gamma$	0	1	a	b	B
$\rightarrow q_0$	$(q_1, a, R)$	-	-	$(q_3, b, R)$	-
$q_1$	$(q_1, 0, R)$	$(q_2, b, L)$	-	$(q_1, b, R)$	-
$q_2$	$(q_2, 0, L)$	-	$(q_0, a, R)$	$(q_2, b, L)$	-
$q_3$	-	-	-	$(q_3, b, R)$	$(q_4, B, R)$
* $q_4$	-	-	-	-	-

Akzeptieren



Maschine hält im Endzustand  $q_4$  an

# ARBEITSWEISE VON TURINGMASCHINEN INTUITIV

- **Anfangssituation**
    - Eingabewort  $w$  steht auf dem Band, umgeben von Leerzeichen
    - Kopf ist über erstem Symbol, Startzustand ist  $q_0$
  - **Arbeitsschritt**
    - Im Zustand  $q$  lese Bandsymbol  $X$  und bestimme  $\delta(q,X) = (p,Y,D)$
    - Wechsle in Zustand  $p$ , schreibe  $Y$  aufs Band, bewege Kopf gemäß  $D$
  - **Terminierung, wenn  $\delta(q,X)$  nicht definiert**
    - Alternativ: Maschine hält bei Erreichen eines Endzustands
    - **Konvention:  $\delta(q,X)$  undefiniert für Endzustände  $q \in F$**
  - **Ergebnis**
    - Eingabewort  $w$  wird akzeptiert, wenn Maschine im Endzustand anhält
  - **Hilfsmittel zur Präzisierung: Konfigurationen**
    - Verallgemeinerte bekanntes Konzept der Konfigurationsübergänge
- Details in Literatur sehr unterschiedlich!!**

# ARBEITSWEISE VON TURINGMASCHINEN PRÄZIERT

## • Erweiterte Begriff der Konfiguration

- Zustand  $q$ , Inhalt des Bandes und Kopfposition
- Formal dargestellt als Tripel  $\mathbf{K} = (u, q, v) \in \Gamma^* \times Q \times \Gamma^+$
- $u, v$ : String links/rechts vom Kopf

Achtung: im Buch wird das Tripel als ein (!) String  $uqv$  geschrieben

- Nur der bereits ‘besuchten’ Teil des Bandes wird betrachtet
- Blanks am Anfang von  $u$  oder am Ende von  $v$  entfallen, wo möglich

## • Modifizierte Konfigurationsübergangsrelation $\vdash^*$

- $(uZ, q, Xv) \vdash (u, p, ZYv)$ , falls  $\delta(q, X) = (p, Y, L)$
- $(u, q, Xv) \vdash (uY, p, v)$ , falls  $\delta(q, X) = (p, Y, R)$

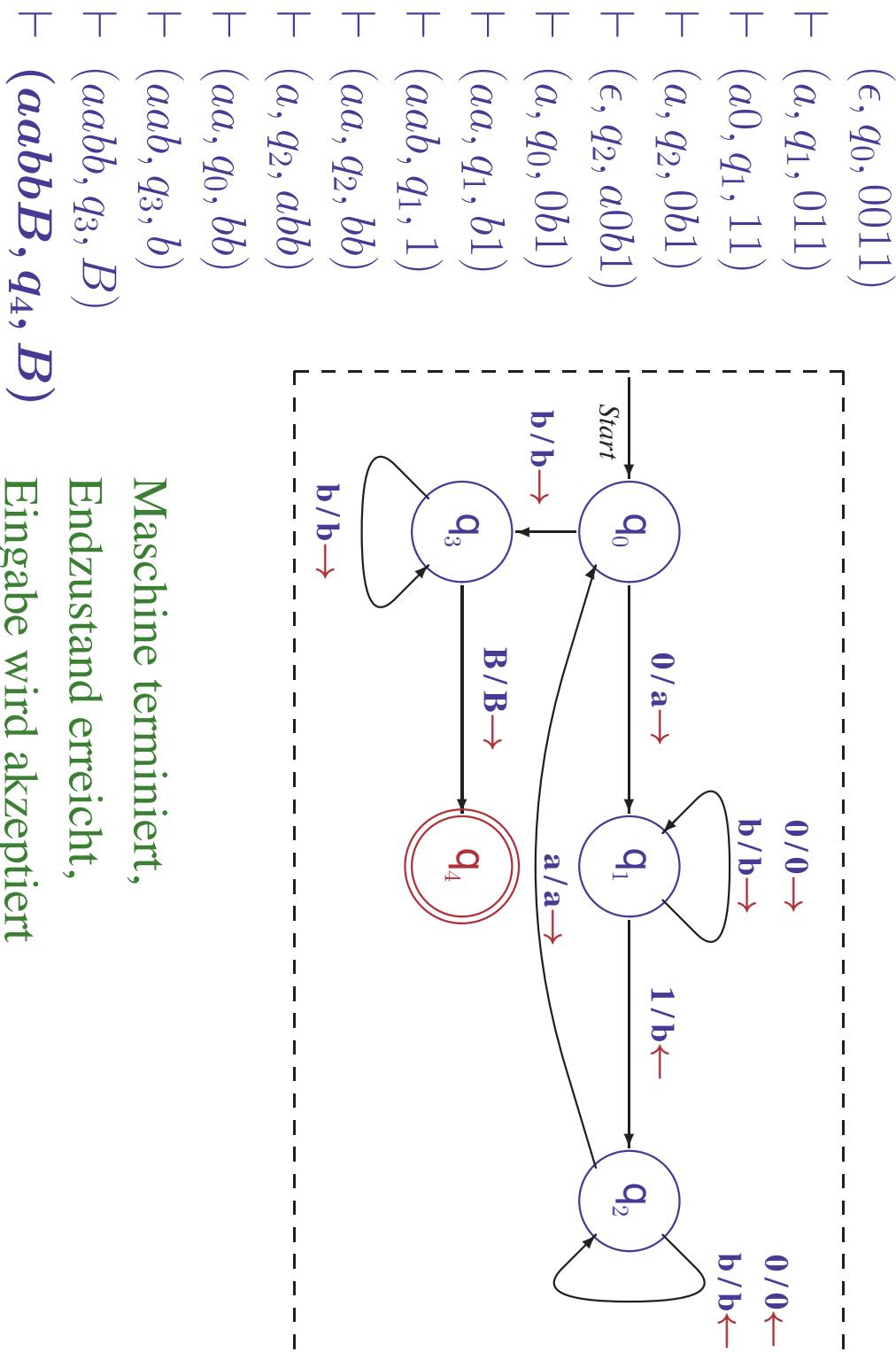
Sonderfälle für Verhalten am Bandende

- $(\epsilon, q, Xv) \vdash (\epsilon, p, BYv)$ , falls  $\delta(q, X) = (p, Y, L)$
- $(uZ, q, X) \vdash (u, p, Z)$ , falls  $\delta(q, X) = (p, B, L)$
- $(u, q, X) \vdash (uY, p, B)$ , falls  $\delta(q, X) = (p, Y, R)$
- $(\epsilon, q, Xv) \vdash (\epsilon, p, v)$ , falls  $\delta(q, X) = (p, B, R)$

$K_1 \vdash^* K_2$ , falls  $K_1 = K_2$  oder es gibt ein  $K$  mit  $K_1 \vdash K$  und  $K \vdash^* K_2$

# VERARBEITUNG EINES EINGABEWORTES

**Eingabewort 0011 ergibt Anfangskonfiguration  $(\epsilon, q_0, 0011)$**



- Maschine terminiert,  
Endzustand erreicht,
- Eingabe wird akzeptiert

# DIE SPRACHE EINER TURINGMASCHINE

- **Akzeptierte Sprache**

- Menge der Eingaben, für die  $\vdash^*$  zu akzeptierendem Zustand führt

$$L(M) = \{w \in \Sigma^* \mid \exists p \in F. \exists u, v \in \Gamma^*. (\epsilon, q_0, w) \xrightarrow{*} (u, p, v)\}$$

Bei Einhalten der Konvention hält  $M$  im akzeptierenden Zustand an

- **Semi-entscheidbare Sprache**

- Sprache, die von einer Turingmaschine  $M$  akzeptiert wird

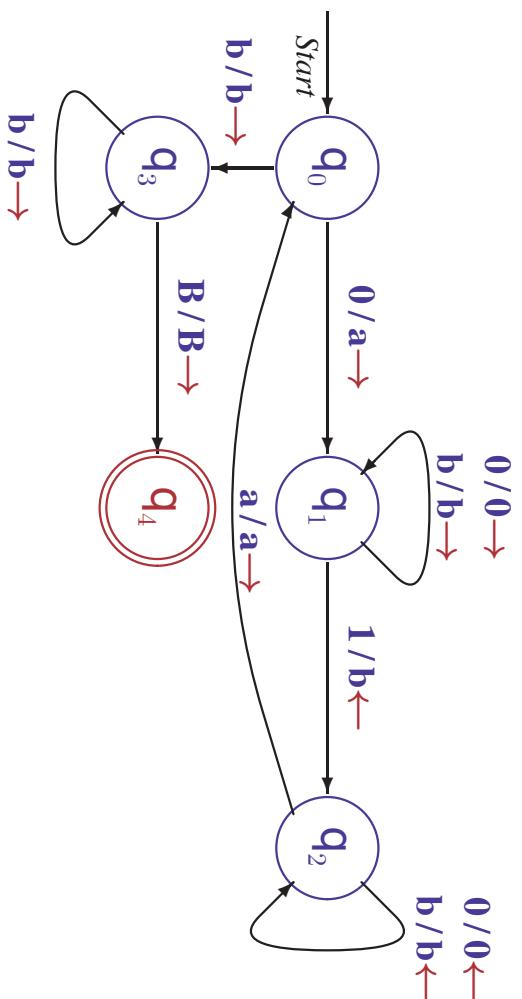
– Alternative Bezeichnungen: **(rekursiv) aufzählbare Sprache**

**Turing-akzeptierbare Sprache**

- **Entscheidbare Sprache**

- Sprache, die von einer Turingmaschine  $M$  akzeptiert wird,  
die bei jeder Eingabe terminiert
- Alternative Bezeichnung: **rekursive Sprache**

# ERKANNTE SPRACHE EINER TURINGMASCHINE



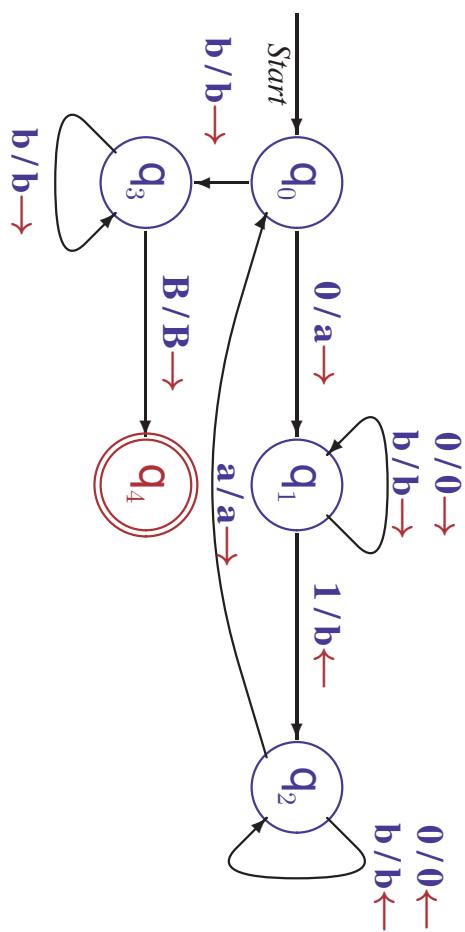
- **Analyse:  $M$  zählt Nullen und Einsen gleichzeitig**

- Umwandeln einer 0 in  $a$  triggert Umwandeln einer 1 in  $b$
- Maschine stoppt in  $q_1$ , wenn zuwenig Einsen vorhanden sind
- Maschine stoppt in  $q_3$ , wenn zuwenig Nullen vorhanden sind
- Maschine akzeptiert in  $q_4$ , wenn Anzahl der Nullen und Einsen gleich

- **Zeige:  $L(M) = \{0^n 1^n \mid n \geq 1\}$**

- $(\epsilon, q_0, w) \vdash^* (u, q_4, v)$  genau dann, wenn  $w = 0^n 1^n$  für ein  $n \geq 1$

## NACHWEIS DER ERKANNNTEN SPRACHE



$(\epsilon, q_0, w) \vdash^* (u, q_4, v)$  wenn  $w = 0^n 1^n$  für ein  $n \geq 1$

- $(u, q_0, 0v) \vdash^* (ua, q_1, v)$  für alle  $u, v \in \Gamma^*$
- $(u, q_0, 0wv) \vdash^* (uaw, q_1, v)$  für alle  $u, v \in \Gamma^*, w \in \{0, b\}^*$
- $(u, q_0, 0w1v) \vdash^* (u, q_2, awbv)$  für alle  $u, v \in \Gamma^*, w \in \{0, b\}^*$
- $(u, q_0, 0w1v) \vdash^* (ua, q_0, wbv)$  für alle  $u, v \in \Gamma^*, w \in \{0, b\}^*$
- $(\epsilon, q_0, 0^k w 1^k v) \vdash^* (a^k, q_0, wb^k v)$  für alle  $v \in \Gamma^*, w \in \{0, b\}^*, k \in \mathbb{N}$
- $(\epsilon, q_0, 0^k 1^k v) \vdash^* (a^k b^k, q_3, v)$  für alle  $v \in \Gamma^*, k \geq 1$
- $(\epsilon, q_0, 0^k 1^k) \vdash^* (a^k b^k, q_3, B)$  für alle  $k \geq 1$
- $(\epsilon, q_0, 0^k 1^k) \vdash^* (a^k b^k B, q_4, B)$  für alle  $v \in \Gamma^*, k \geq 1$

Argument, warum andere Wörter nicht akzeptiert werden, ist aufwendiger

## AUSDRUCKSKRAFT VON TURINGMASCHINEN

### Genauso leistungsfähig wie konventionelle Computer

- **Reale Computer bieten viele Freiheiten**

- Programme als Daten im Speicher
- Datenregister und Programmzähler
- “Simultaner” direkter Zugriff auf mehrere Speicherzellen
- Unterprogramme

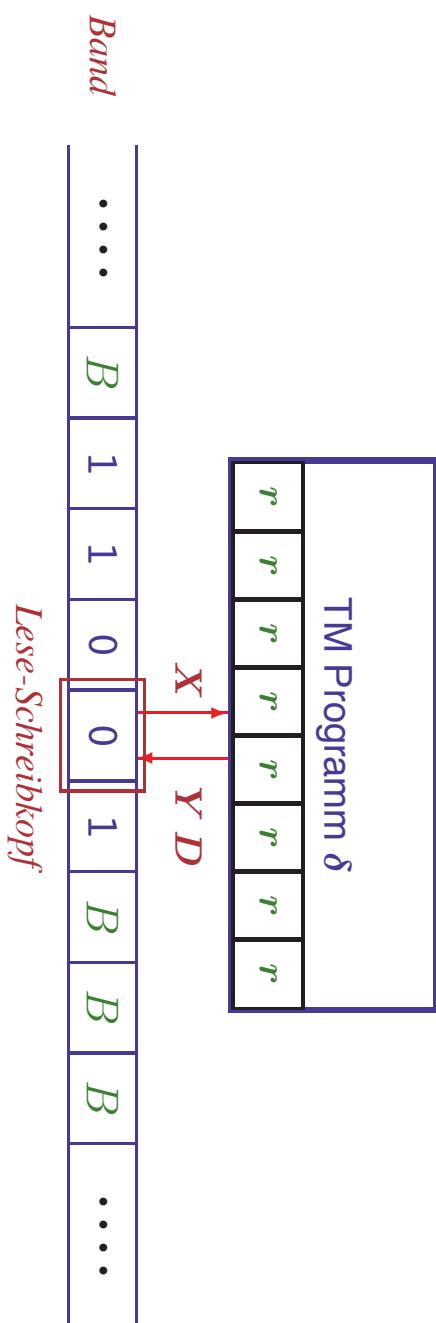
- **Turingmaschinen sind unbeschränkt**

- Beliebig große Alphabete (statt binären Daten)
- Unendliches Speicherband

- **Gegenseitige Simulation ist möglich**

- Zusätzliche Freiheiten als Programmiertechniken einer TM simulierbar
- Beschränkungen des TM Modells verringern die Ausdruckskraft nicht

# PROGRAMMIERTECHNIK: DATENREGISTER



- **TM hat zusätzlich endliche Menge von Registern**

- Jedes Register kann einen Wert aus einer endlichen Menge  $\Delta$  enthalten
- Maschine kann jeweils eine Bandzelle und alle Register bearbeiten
- Verwendung: Speichern einer Menge von Daten separat vom Band

- **Simulation durch erweiterte Zustandsmenge**

- Bei  $k$  Registern wähle Zustandsmenge  $Q' := Q \times \Delta^k$
- Simuliere Zustandsübergang in  $Q$  und Änderung der Register durch entsprechenden Zustandsübergang in  $Q'$

# SIMULATION EINER MASCHINE MIT REGISTERN

## Beschreibe Maschine, die $L((01^*) + (10^*))$ erkennt

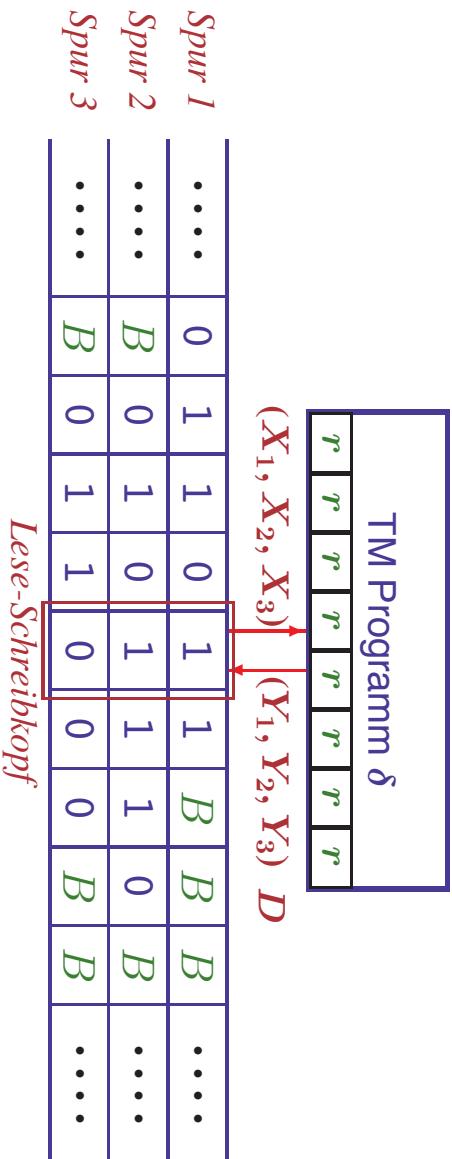
- **Einfache Lösung mit Registern**

- Speichere erstes Bandsymbol im Register
- $q_0$ : Prüfe ob das gespeicherte Symbol im restlichen Wort vorkommt
- $q_1$ : Akzeptiere, wenn gesamtes Wort erfolgreich überprüft

- **Simulation mit  $Q' := \{q_0, q_1\} \times \{0,1,B\}$**

	0	1	B	
$\rightarrow$	$(q_0, B)$	$((q_0, 0), 0, R)$	$((q_0, 1), 1, R)$	–
	$(q_0, 0)$	–	$((q_0, 0), 1, R)$	<i>Mit 0 vergleichen</i>
	$(q_0, 1)$	$((q_0, 1), 0, R)$	–	<i>Mit 1 vergleichen</i>
*	$(q_1, B)$	–	–	<i>Vergleich war erfolgreich</i>
	$(q_1, 0)$	–	–	<i>(Nicht erreichbar)</i>
	$(q_1, 1)$	–	–	<i>(Nicht erreichbar)</i>

# PROGRAMMIERTECHNIK: MEHRERE SPUREN



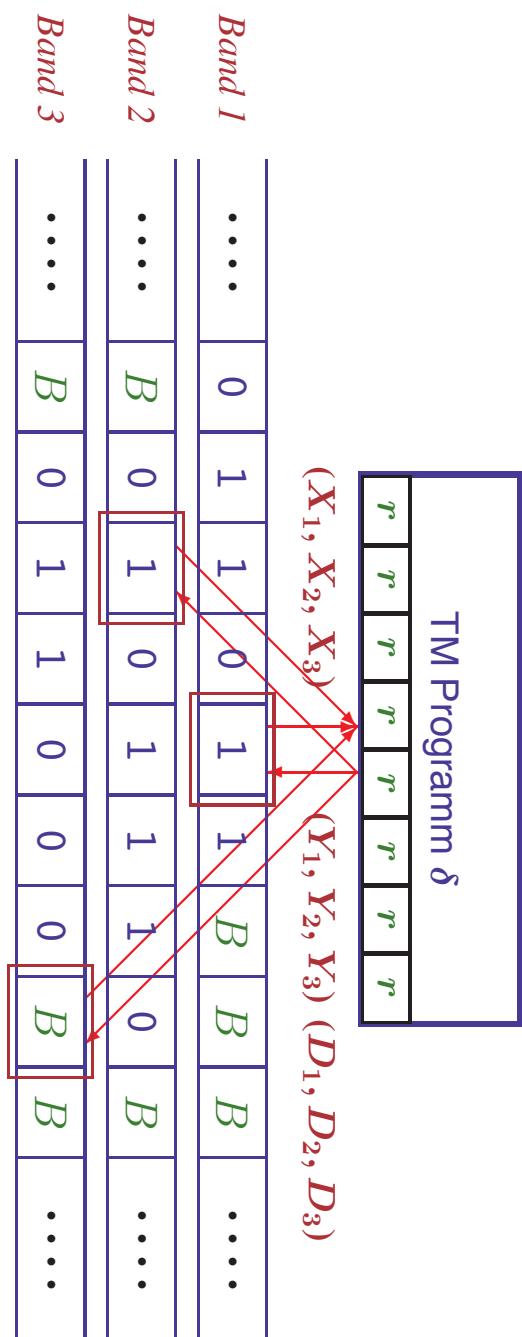
- **Band hat mehrere Datenspuren**

- Jede Spur enthält ein Symbol des Bandalphabets  $\Gamma$
- Alle Symbole werden simultan gelesen und geschrieben
- Kopf wird ‘synchron’ über das Band bewegt
- Verwendung: Simultane Verarbeitung von Teilen der Eingabe  
z.B. zur Erkennung von  $\{w\#w \mid w \in \{0, 1\}^*\}$        $\mapsto$  HMU, §8.3.2

- **Simulation durch erweitertes Bandalphabet**

- Bei  $k$  Spuren wähle Tupelalphabet  $\Sigma' := \Sigma^k$
- In jedem Schritt wird ‘ein’ Symbol  $X := (x_1, \dots, x_k)$  verarbeitet, wobei  $x_i$  dem Symbol auf Spur  $i$  entspricht

# PROGRAMMIERTECHNIK: MEHRERE BÄNDER



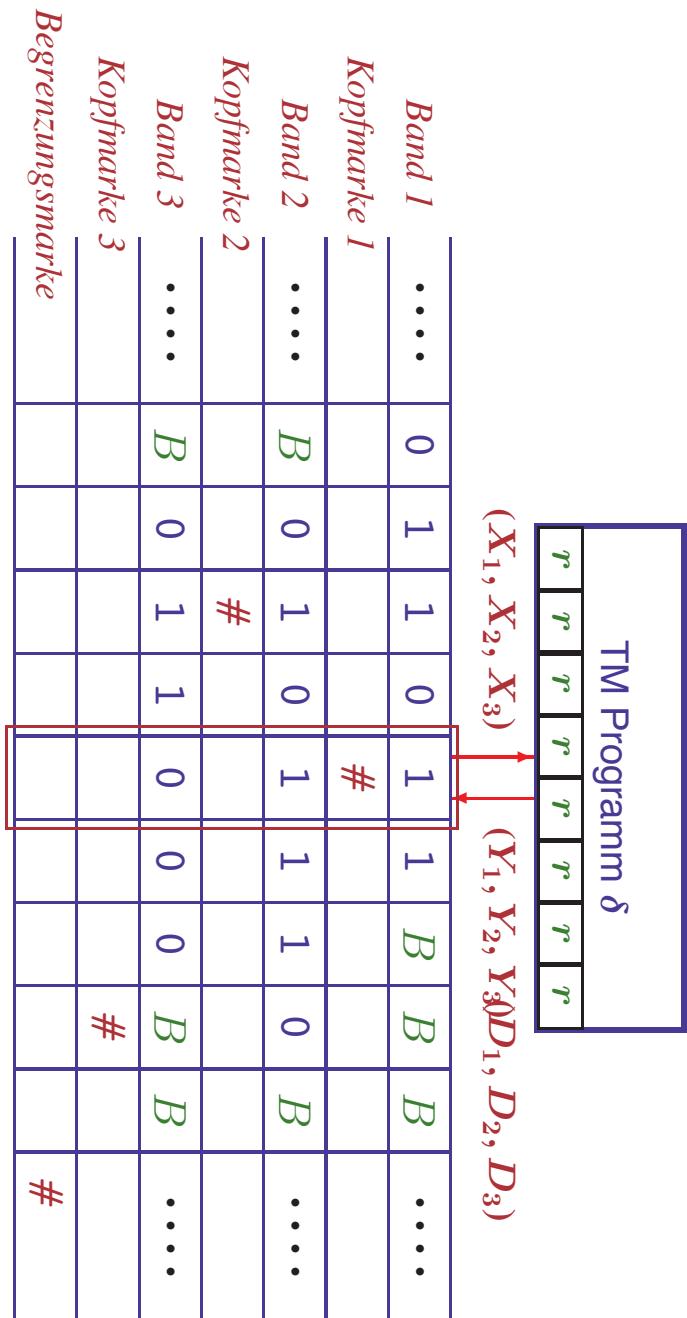
- **Maschine verwaltet mehrere Bänder**

- Jedes Band enthält ein Symbol des Bandalphabets  $\Gamma$
- Alle Symbole werden simultan gelesen und geschrieben
- Köpfe werden **unabhängig** über die Bänder bewegt
- Erheblich größere Freiheiten bei der Programmierung

- **Simulation aufwendiger**

- Mehrspurband + Verwaltung der Kopfpositionen auf separaten Spuren
- Spuren werden “einzelne aufgesucht” und modifiziert

# SIMULATION EINER MEHRBANDMASCHINE



## • Sequentielle Verarbeitung der einzelnen Bänder

- Lesen: Suche Begrenzungsmarke, laufe rückwärts zu Kopfmarken, sammle zu lesende Symbole in Registern
- Schreiben + Kopfbewegungen: lege Symbole und Richtungen in Register suchе Kopfmarken und überschreibe Teilzelle entsprechend

**Simulation benötigt quadratischen Zeitaufwand**

→ HMU, §8.4.3

## Ausführung einer anderen TM als Zwischenschritt

- **Aufruf von  $M'$  in Überführungsfunktion von  $M$**

- $M'$  erhält Eingabewort von  $M$  und gibt Resultat an  $M$  zurück
- $M$  wechselt nach Ausführung von  $M'$  in festen Folgezustand
- Anwendungsbeispiel: Multiplikation als wiederholte Addition

- **Simulation wie bei Assembler-Unterprogrammen**

- Umbenennung aller Zustände von  $M'$  zur Konfliktvermeidung
- Ergänze Zustand  $q_r$  für Rücksprung ins aufrufende Programm
- Ergänze separates Arbeitsband für Unterprogramm
- Aufruf: Speichere Rücksprungadresse (Zustand von  $M$ ) in Register
- Kopiere Eingabe für Unterprogramme auf Arbeitsband für  $M'$
- Nach Abarbeitung kopiere Resultate auf Arbeitsband von  $M$
- Wechsle in Zustand, der im Register gespeichert ist

# BESCHRÄNKTE TURINGMASCHINENMODELLE

## Restriktionen vereinfachen Analysen von TM

Einfachere Annahmen und weniger Alternativen in Beweisen

**Kein Verlust der Ausdruckskraft:** Simulation normaler TMs möglich

### 1. Halbseitig unendliches Band

- Beidseitig unendliches Band durch Tupelalphabet  $\Gamma^2$  simulierbar
- Im Paar  $(X_l, X_r)$  repräsentiert  $X_l$  die linke,  $X_r$  die rechte Bandhälfte
- Register (simulierbar im Zustand) gibt an, welche Hälfte aktiv ist

### 2. Binäres Bandalphabet $\Gamma = \{1, B\}$

- Symbole beliebiger Alphabete als Strings über  $\{1B, 11\}$  simulierbar

### 3. Zwei Stacks statt Turingband

- 2 Stacks + Zustand können jede Konfiguration  $(u, q, v)$  beschreiben

### 4. Zählermaschinen

- Endliche Zahl von Registern kann beliebig große Zahlen verarbeiten
- Operationen: Test auf Null, Addition oder Subtraktion von Eins
- Zähler können Stacks simulieren (aufwendige Codierung von Wörtern als Zahl)

→ HMU, §8.5.1

→ HMU, §8.5.2

→ HMU, §8.5.3/4

# DER VERGLEICH MIT REALEN COMPUTERN

- **Computer können Turingmaschinen simulieren**

- Repräsentiere binäres Bandalphabet und halbseitig unendliches Band
- (Endliche) reale Speicher können nach Bedarf beliebig erweitert werden

- **Turingmaschinen können Computer simulieren**

- Speicher wird durch einseitiges Band mit binärem Alphabet repräsentiert
- Register enthalten Programmzähler, Speicheraddressregister, etc.
- Aufsuchen einer Speicherzelle vom Bandanfang durch Zählen
- Gesuchter Speicherinhalt wird im Register abgelegt und analysiert
- Identifizierte Anweisungen werden durch Unterprogramme ausgeführt
- Nach Ausführung wird Anweisungszähler angepaßt und die nächste Anweisung aus dem Speicher geholt

- **Simulationsaufwand ist polynomiell**

→ HMU, §8.6.3

- $n$  Schritte des realen Computers benötigen maximal  $n^6$  Schritte
- Optimierungen möglich

# TURINGMASCHINEN IM RÜCKBLICK

- **Allgemeinstes Maschinенmodell**
  - Deterministischer endlicher Automat mit unendlichem Speicherband
  - “Beliebiger” Zugriff auf Speicherzellen
  - Erkennung von Wörtern durch Endzustand
- **Verhaltensanalyse durch Konfigurationsübergänge**
  - Konfigurationen beschreiben Zustand, Bandinhalt & Kopfposition
- **Äquivalent zu realen Computern**
  - Register, mehrere Bänder, Unterprogramme, etc. simulierbar
  - Beschränkte Maschinenmodelle sind ebenfalls gleich mächtig

# Theoretische Informatik I

## Einheit 4.2

### Modelle für Typ-0 & Typ-1 Sprachen



1. Nichtdeterministische Turingmaschinen
2. Äquivalenz zu Typ-0 Sprachen
3. Linear beschränkte Automaten  
und Typ-1 Sprachen

# MASCHINENMODELLE VS. GRAMMATIKEN

- Ableitbarkeit  $w \xrightarrow{G} z$  ist nichtdeterministisch
  - In  $w$  können verschiedene Teilworte ersetzt werden
  - Auf ein Teilwort können verschiedene Regeln angewandt werden
  - Simulation erfordert nichtdeterministisches Maschinenmodell
- Maschinenmodelle sind i.a. deterministisch
  - Nichtdeterministische Modelle sind “unrealistisch” und nur für elegantere Modellierung geeignet
  - Nichtdeterministische Modelle sind evtl. deterministisch simulierbar aber nur mit exponentiellem Aufwand
- Verwende nichtdeterministische Turingmaschinen
  - “Simultane” Behandlung vieler alternativer Konfigurationen
  - Zeige Äquivalenz zu deterministischen Turingmaschinen
  - Zeige Äquivalenz zu Typ-0 Grammatiken
  - Zeige Äquivalenz zu Typ-1 Grammatiken für eingeschränktes Modell

## NICHTDETERMINISTISCHE TURINGMASCHINEN

- Eine nichtdeterministische Turingmaschine (**NTM**) ist ein 7-Tupel  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$  mit
  - $Q$  nichtleere endliche Zustandsmenge
  - $\Sigma$  endliches Eingabealphabet
  - $\Gamma \supseteq \Sigma$  endliches Bandalphabet
  - $\delta: Q \times \Gamma \rightarrow \mathcal{P}_e(Q \times \Gamma \times \{L, R\})$  endliche Überführungsfunktion
  - $q_0 \in Q$  Startzustand
  - $B \in \Gamma \setminus \Sigma$  Leersymbol des Bands
  - $F \subseteq Q$  Menge von akzeptierenden (End-)Zuständen
- Definition von  $\vdash^*$  und  $L(M)$  analog zu DTM
  - $(uZ, q, Xv) \vdash (u, p, ZYv)$ , falls  $(p, Y, L) \in \delta(q, X)$
  - $(u, q, Xv) \vdash (uY, p, v)$ , falls  $(p, Y, R) \in \delta(q, X)$
  - ⋮
  - $L(M) = \{w \in \Sigma^* \mid \exists p \in F. \exists u, v \in \Gamma^*. (\epsilon, q_0, w) \vdash^* (u, p, v)\}$

# JEDE NTM IST DURCH EINE DTM SIMULIERBAR

**Verarbeite alle Alternativen sequentiell**

- Für  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$  definiere Mengen  $K_i^w$

- Menge der in  $i$  Schritten erzeugbaren Konfigurationen bei Eingabe  $w$

$$K_0^w := \{(\epsilon, q_0, w)\}, \quad K_{i+1}^w := \{\kappa' \mid \exists \kappa \in K_i^w. \kappa \vdash \kappa'\}$$

- Es gilt  $w \in L(M) \Leftrightarrow \exists i. \exists p \in F. \exists u, v \in \Gamma^*. (u, p, v) \in K_i^w$

- Beschreibe DTM zur Erzeugung der  $K_i^w$

$K_0^w$

$K_1^w$

$K_2^w$

$q_0$	$w_1..w_n$	\$	#	$u_1^1 q_1^1 v_1^1$	#	$u_1^2 q_1^2 v_1^2$	#	...	$u_1^{j_1} q_1^{j_1} v_1^{j_1}$	\$	#	$u_2^1 q_2^1 v_2^1$	#	...	$u_2^{j_2} q_2^{j_2} v_2^{j_2}$	\$	...
-------	------------	----	---	---------------------	---	---------------------	---	-----	---------------------------------	----	---	---------------------	---	-----	---------------------------------	----	-----

- Arbeitsband beschreibt alle bisher erzeugten Konfigurationen der NTM
- Die aktuell betrachtete Konfiguration  $\kappa$  wird markiert
- **Lesen:** Extrahiere aus  $\kappa$  das gelesene Symbol  $X$  und Zustand  $q$  der NTM
- **Verarbeiten:** Erzeuge aus  $\kappa$  und  $\delta(q, X)$  alle Nachfolgekonfigurationen
- Lösche Markierung von  $\kappa$  und markiere nächste Konfiguration auf Band
- Jede mögliche Konfiguration der NTM wird von der DTM aufgesucht

# SIMULATION VON NTMS IST EXPONENTIELL

- Größe der DTM linear mit Größe der NTM
  - Zustandsüberführungstabelle wird durch Unterprogramme codiert
  - Berechnung der Nachfolgekonfigurationen auf einem Hilfsband
- Rechenzeit wächst exponentiell
  - Einzelschritte linear in Größe einer NTM Konfiguation simulierbar
    - Bestimmen einer Nachfolgekonfiguration ist “konstant”
    - Schreiben der Nachfolgekonfiguration linear (zwei Arbeitsbänder)
  - Aber ...
  - Rechenzeit der NTM ist Länge des kürzesten akzeptierenden Pfades
  - Bei  $k$  Alternativen pro Schritt muß die Simulation für  $n$  Schritte der NTM im schlimmsten Fall bis zu  $k^n$  Konfigurationen erzeugen

# TURINGMASCHINEN ALS MASCHINENMODELL FÜR $\mathcal{L}_0$

**Typ-0 Grammatiken und Turingmaschinen beschreiben dieselbe Klasse von Sprachen**

## • Grammatik → Turingmaschine

- Turingmaschine simuliert Anwendung der Produktionsregeln
- Ableitbare Wörter werden schrittweise auf Hilfsband geschrieben
- Wörter auf dem Hilfsband werden mit der Eingabe verglichen
- Maschine akzeptiert, wenn  $w \in L(G)$ , und terminiert sonst nicht

## • Turingmaschine → Grammatik

- Grammatik simuliert Konfigurationsübergänge der Turingmaschine
- Erzeuge alle möglichen Eingabewörter und Anfangskonfigurationen
- Codiere Konfigurationsübergänge von  $M$  als Regeln
- Simuliere Akzeptieren durch Löschen von Nonterminalsymbolen
- Grammatik generiert genau alle Wörter, die  $M$  akzeptiert

**SATZ:**  $L \in \mathcal{L}_0 \Rightarrow L$  SEMI-ENTSCHEIDBAR

**Zu jeder Grammatik  $G = (V, T, P, S)$  kann eine NTM  $M$  konstruiert werden mit  $L(G) = L(M)$**

- 1. Schreibe die Eingabe  $w$  auf Hilfsband 1**
- 2. Schreibe das Startsymbol  $S$  auf Hilfsband 2**
- 3. Simuliere eine Regelanwendung in  $G$** 
  - Wähle nichtdeterministisch ein Teilwort  $u$  des Wortes auf Band 2
  - Wähle nichtdeterministisch eine Regel der Form  $u \rightarrow v$  aus  $P$
  - Verschiebe Symbole, die rechts von  $u$  stehen, um  $|v| - |u|$  Stellen
  - Ersetze  $u$  durch  $v$
- 4. Vergleiche  $w$  mit dem Wort auf Hilfsband 2**
  - Akzeptiere  $w$ , wenn die Worte gleich sind
  - Ansonsten fahre fort mit 3.

## KORREKTHEIT DER KONSTRUKTION

- **$M$  simuliert Ableitbarkeit in  $G$** 
  - Nach  $i$  Schritten steht auf Band 2 ein Wort  $w_i$  mit  $S \xrightarrow[G]{i} w_i$
  - Wenn  $M$  das Wort  $w$  nach  $i$  Schritten akzeptiert, dann gilt  $w = w_i$
- **$M$  akzeptiert  $L(G)$** 
  - Es gilt  $w \in L(G) \Leftrightarrow \exists i. S \xrightarrow[G]{i} w$
  - Wenn  $M$  das Wort  $w$  nach  $i$  Schritten akzeptiert, dann gilt  
 $S \xrightarrow[G]{i} w$ , also  $w \in L(G)$
  - Wenn  $S \xrightarrow[G]{i} w$  gilt, dann kann  $M$  in  $i$  Schritten das Wort  $w$  auf Band 2 erzeugen und akzeptieren, also  $w \in L(M)$
- **$M$  terminiert nicht immer**
  - Für  $w \notin L(G)$  gilt  $w \neq w_i$  für alle  $i$

SATZ:  $L$  SEMI-ENTSCHEIDBAR  $\Rightarrow L \in \mathcal{L}_0$

## Simuliere Abarbeitung der Turingmaschine

- **Idee: Generiere alle Konfigurationen von  $M$** 
  - Konfigurationen  $(u, q, v)$  werden als Wörter  $uqv$  codiert
  - Begrenzer  $\#$  trennt Eingabe  $w$  von Konfigurationen
    - Verarbeitung von  $w$  simuliert durch Wörter der Form  $w \# uqv \#$
- **Beschreibe Konfigurationsübergänge durch Regeln**
  - Regeln simulieren Vorschriften für Erzeugung von  $\vdash$  aus  $\delta$
- **Lege  $w$  frei, wenn  $M$  akzeptiert hat**
  - Entferne Wort nach  $\#$ , wenn  $M$  einen Endzustand erreicht
- **Grammatik erzeugt von  $M$  akzeptierte Sprache**
  - $L(G) = \{w \in \Sigma^* \mid \exists p \in F. \exists u, v \in \Gamma^*. ((\epsilon, q_0, w) \vdash^* (u, p, v)) = L(M)\}$

# REGELN DER GRAMMATIK $G$

- **Erzeugung von Anfangskonfigurationen**

- Regeln zur Erzeugung aller Wörter der Form  $w \# q_0 w \#$

- **Simulation der Konfigurationsübergänge**

- Regeln der Form  $\textcolor{red}{q} X V \mapsto Y \textcolor{red}{p} V$  für  $V \in \Gamma$ ,  $\delta(q, X) = (p, Y, R)$
- Regeln der Form  $\textcolor{red}{q} X \# \mapsto Y \textcolor{red}{p} B \#$  für  $\delta(q, X) = (p, Y, R)$
- Regeln der Form  $Z \textcolor{red}{q} X \mapsto \textcolor{red}{p} Z Y$  für  $Z \in \Gamma$ ,  $\delta(q, X) = (p, Y, L)$
- Regeln der Form  $\# \textcolor{red}{q} X \mapsto \# \textcolor{red}{p} B Y$  für  $\delta(q, X) = (p, Y, L)$

- **Schlußregeln für Endzustände**

- Regeln der Form  $Z \textcolor{blue}{q} \mapsto \textcolor{red}{q}$  für  $Z \in \Gamma$ ,  $q \in F$
- Regeln der Form  $\textcolor{red}{q} Z \mapsto q$  für  $Z \in \Gamma$ ,  $q \in F$
- Regeln der Form  $\# \textcolor{red}{q} \# \mapsto \epsilon$  für  $q \in F$

Detailbeweise z.B. in Erk-Priese, Seite 199–201

# LINEAR BESCHRÄNKTE AUTOMATEN

## Welches Modell passt zu Typ-1 Sprachen?

- **Typ-1 Sprachen werden “expansiv” erzeugt**
  - In jeder Ableitung  $S \xrightarrow{} w_1 \xrightarrow{} w_2 \dots \xrightarrow{} w$  eines Wortes  $w \in L(G)$  ist keines der  $w_i$  länger als  $w$  (Ausnahme  $w = \epsilon$ )
  - Turingmaschine braucht maximal  $|w|$  Bandzellen zur Simulation
- **Beschränke NTMs auf linearen Bandverbrauch**
  - Das Arbeitsband ist nur halbseitig unendlich
  - Anfangskonfigurationen haben die Form  $(\epsilon, q_0, w\#)$
  - $\#$  ist ein spezielles Bandende-Symbol, das niemals überlaufen oder überschrieben werden darf
- **Formal: linear beschränkter Automat (LBA)**
  - NTM  $M = Q, \Sigma, \Gamma, \delta, q_0, B, F$  mit halbseitig unendlichem Band und ausgezeichnetem Symbol  $\# \in \Gamma \setminus (\Sigma \cup \{B\})$
  - und der Einschränkung  $\delta(q, \#) \subseteq \{(p, \#, L) \mid p \in Q\}$  für alle  $q \in Q$

# LINEAR BESCHRÄNKTER AUTOMAT FÜR $\{0^n 1^n 2^n \mid n \in \mathbb{N}\}$

- 1. Anfangskonfiguration ist  $(\epsilon, q_0, w\#)$**
  - 2. Wenn das Band leer ist, akzeptiere die Eingabe**
  - 3. Ansonsten ersetze die erste 0 durch B**
    - Wenn keine 0 unter dem Kopf steht, halte an ohne zu akzeptieren
  - 4. Gehe rechts zur ersten 1; ersetze diese durch B**
    - Vor der 1 dürfen nur Nullen oder Blanks kommen (!)
    - Wenn keine 1 vorkommt, halte an ohne zu akzeptieren
  - 5. Gehe rechts zur ersten 2; ersetze diese durch B**
    - Vor der 2 dürfen nur noch Einsen oder Blanks kommen (!)
    - Wenn keine 2 am Ende steht, halte an ohne zu akzeptieren
  - 6. Laufe zurück zum Anfang des restlichen Wortes**
    - Fahre fort mit Schritt 2
- 

## Optimierung: Schließe Lücken durch Verschieben

- Verfahren funktioniert analog auch für  $\{0^n 1^n 2^n 3^n 4^n \mid n \in \mathbb{N}\}$

# LBAs SIND MASCHINENMODELL FÜR TYP-1 SPRACHEN

$$\mathcal{L}_1 = \{L \mid L = L(M) \text{ für einen LBA } M\}$$

- Beweise für  $\mathcal{L}_0$  können modifiziert werden
- **Typ-1 Grammatik** → LBA
  - Turingmaschine simuliert Anwendung der Produktionsregeln
  - Ableitbare Wörter werden schrittweise erzeugt und mit  $w$  verglichen
  - Beschränkung der Simulation auf Regelanwendungen, die Wörter mit maximaler Länge  $|w|$  erzeugen
- **Maschine ist linear beschränkter Automat**
  - Linkes und rechtes Ende des Bandes wird niemals überschritten
  - Lineare Simulation der Hilfsbänder mit größerem Bandalphabet
- **LBA → Typ-1 Grammatik**
  - LBA muß bei Eingabe  $w$  das Band nicht mehr erweitern
  - Simulierte Verarbeitung der Eingabe  $w$  mit Wörtern der Form  $(w_1, u_1) .. (w_i, u_i)(w_{i+1}, q)(w_{i+1}, v_1) .. (w_n, v_j)$  statt  $w \# uqv \#$
  - Kürzende Grammatikregeln können jetzt expansiv formuliert werden

# Theoretische Informatik I

## Einheit 4.3

### Eigenschaften von $\mathcal{L}_0/\mathcal{L}_1$ -Sprachen



1. Abschlußeigenschaften
2. Prüfen von Eigenschaften
3. Grenzen der Sprachklassen



## SPRACHKLASSEN

- **Semi-entscheidbare Sprache**

- Sprache, die von einer Turingmaschine akzeptiert wird
- Auch aufzählbare Sprache genannt

- **Entscheidbare Sprache**

- Sprache, die von einer Turingmaschine akzeptiert wird, die bei jeder Eingabe terminiert
- Auch rekursive Sprache genannt

- **Kontextsensitive Sprache**

- Sprache, die von einem linear beschränkten Automaten akzeptiert wird

- **Entscheidbare Sprachen sind aufzählbar**

- Offensichtlich, da engere Bedingung

- **Kontextsensitive Sprachen sind entscheidbar**

- Ein LBA hat bei Eingabe  $w$  maximal  $(|\Gamma| + |Q|)^{|w|+1}$  Konfigurationen
- Wenn der LBA nach  $(|\Gamma| + |Q|)^{|w|+1}$  Schritten nicht akzeptiert, dann gehört  $w$  nicht zur Sprache und die Berechnung kann anhalten

## ABSCHLUSSEIGENSCHAFTEN SUMMARISCH

- Alle drei Sprachklassen sind abgeschlossen unter

- Vereinigung  $L_1 \cup L_2$
- Durchschnitt  $L_1 \cap L_2$
- Spiegelung  $L^R$

- Verkettung

- Hüllenbildung

- Homomorphismen

- Inverse Homomorphismen

- Urbild berechenbarer Funktionen

- Typ-1 und entscheidbare Sprachen zusätzlich

- Komplement

- Differenz

- Aufzählbare Sprachen: Bild berechenbarer Funktionen

$$\overline{L}$$

$$L_1 - L_2$$

$$f(L)$$

# NACHWEIS DER ABSCHLUSSEIGENSCHAFTEN

## Beweisführung mit Grammatiken

- **Vereinigung  $L_1 \cup L_2$** 
  - Sei  $L_i = L(G_i)$ , wobei  $G_i = (V_i, T_i, P_i, S_i)$  disjunkt
  - Wähle  $G = (V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \xrightarrow{} S_1, S \xrightarrow{} S_2\}, S)$
  - Die Eigenschaften der  $G_i$  bleiben erhalten und es gilt  $L(G) = L_1 \cup L_2$
- **Spiegelung  $L_R$** 
  - Bilde **Spiegelgrammatik** zu  $G = (V, T, P, S)$  mit  $L = L(G)$ 
    - Setze  $G_R = (V, T, P_R, S)$  mit  $P_R = \{l^R \xrightarrow{} \alpha^R \mid l \xrightarrow{} \alpha \in P\}$
    - Die Eigenschaften von  $G$  bleiben erhalten und es gilt  $L(G_R) = L^R$
- **Analoge Beweise für  $L_1 \circ L_2, L^*, h(L)$** 
  - Verzweige aus Startsymbol oder modifiziere rechte Seite der Regeln
- **Grammatiken helfen wenig bei Entscheidbarkeit**
  - Beweisführung mit Turingmaschinen ist sinnvoller

## NACHWEIS DER ABSCHLUSSEIGENSCHAFTEN

- **Vereinigung  $L_1 \cup L_2$**

- Sei  $L_i = L(M_i)$ , wobei  $M_i = (Q_i, \Sigma_i, \Gamma_i, \delta_i, q_{0,i}, B, F_i)$  disjunkt
- Bei Eingabe eines Wortes  $w$  kopiert  $\textcolor{blue}{M}$  das Wort auf zwei Hilfsbänder und simuliert  $M_1$  und  $M_2$  auf den beiden Bändern
  - $\textcolor{blue}{M}$  akzeptiert genau dann, wenn  $M_1$  oder  $M_2$  akzeptieren
  - Die Eigenschaften der  $M_i$  bleiben erhalten und es gilt  $L(M) = L_1 \cup L_2$

- **Durchschnitt  $L_1 \cap L_2$**

- Bei Eingabe eines Wortes  $w$  kopiert  $\textcolor{blue}{M}$  das Wort auf zwei Hilfsbänder und simuliert  $M_1$  und  $M_2$  auf den beiden Bändern
  - $\textcolor{blue}{M}$  akzeptiert genau dann, wenn  $M_1$  und  $M_2$  akzeptieren
  - Die Eigenschaften der  $M_i$  bleiben erhalten und es gilt  $L(M) = L_1 \cap L_2$

- **Spiegelung  $L_1^R$**

- Bei Eingabe eines Wortes  $w$  kopiert  $\textcolor{blue}{M}$  das Wort umgedreht auf ein Hilfsband und simuliert  $M_1$  auf diesem Band
  - $\textcolor{blue}{M}$  akzeptiert genau dann, wenn  $M_1$  akzeptiert
  - Die Eigenschaften von  $M_1$  bleiben erhalten und es gilt  $L(M) = L_1^R$

# NACHWEIS DER ABSCHLUSEIGENSCHAFTEN II

- **Verkettung  $L_1 \circ L_2$**

- Bei Eingabe eines Wortes  $w$  wählt  $M$  nichtdeterministisch eine Zerlegung das Wort  $w = w_1 \circ w_2$ , kopiert die  $w_i$  auf zwei Hilfsbänder und simuliert  $M_1$  und  $M_2$  entsprechend
- $M$  akzeptiert genau dann, wenn  $M_1$  und  $M_2$  akzeptieren
- Die Eigenschaften der  $M_i$  bleiben erhalten und es gilt  $L(M) = L_1 \circ L_2$

- **Hülle  $L_1^*$**

- Bei Eingabe eines Wortes  $w$  wählt  $M$  nichtdeterministisch eine Zerlegung des Wortes  $w = w_1 \circ \dots \circ w_n$ , kopiert die  $w_i$  der Reihe nach auf ein Hilfsband und simuliert  $M_1$  entsprechend
- $M$  akzeptiert genau dann, wenn  $M_1$  alle  $w_i$  akzeptiert
- Die Eigenschaften von  $M_1$  bleiben erhalten und es gilt  $L(M) = L_1^*$

- **Homomorphismen  $h(L_1)$**

- Bei Eingabe eines Wortes  $w$  wählt  $M$  nichtdeterministisch eine Zerlegung das Wort  $w = w_1 \circ \dots \circ w_n$  mit  $w_i = h(a_i)$ , kopiert  $v = a_1 \dots a_n$  auf ein Hilfsband und simuliert  $M_1$  entsprechend
- $M$  akzeptiert genau dann, wenn  $M_1$  das Wort  $v$  akzeptiert
- Die Eigenschaften von  $M_1$  bleiben erhalten und es gilt  $L(M) = h(L_1)$

# NACHWEIS DER ABSCHLUSSEIGENSCHAFTEN III

- Inverse Homomorphismen  $h^{-1}(L_1)$

- Bei Eingabe eines Wortes  $w = a_1..a_n$  bestimmt  $M$  das Wort  $v = h(a_1)..h(a_n)$ , kopiert es auf ein Hilfsband und simuliert  $M_1$
- $M$  akzeptiert genau dann, wenn  $M_1$  das Wort  $v$  akzeptiert
- Es gilt  $L(M) = h^{-1}(L_1)$

- Beweis gilt in dieser Form nur für (semi-)entscheidbare Sprachen  
Für LBA's ist Simulation eines Bandes  $k$ -facher Länge erforderlich

- Komplement  $\overline{L_1}$

- Bei Eingabe eines Wortes  $w$  simuliert  $M$  die Berechnung von  $M_1$  und akzeptiert genau dann, wenn  $M_1$  nicht akzeptiert
- Es gilt  $L(M) = \overline{L_1}$
- Die Eigenschaften von  $M_1$  bleiben nur erhalten, wenn  $M_1$  terminiert  
Bei aufzählbaren Sprachen terminiert die Berechnung für  $w \in \overline{L_1}$  nicht

- Differenz  $L_1 - L_2$

- Mathematische Begründung:  $L_1 - L_2 = L_1 \cap \overline{L_2}$
- Abgeschlossenheit unter Differenz gilt für aufzählbare Sprachen nicht!

# AUFZÄHLBARKEIT VS. ENTSCHEIDBARKEIT

- **$L$  entscheidbar  $\Leftrightarrow L$  und  $\overline{L}$  aufzählbar**

“ $\Rightarrow$ ”: Entscheidbare Sprachen sind abgeschlossen unter Komplement

“ $\Leftarrow$ ”: Sei  $L=L(M_1)$  und  $\overline{L}=L(M_2)$ .

- Bei Eingabe eines Wortes  $w$  kopiert  $M$  das Wort auf zwei Hilfsbänder und simuliert  $M_1$  und  $M_2$  auf den beiden Bändern
- $M$  akzeptiert genau dann, wenn  $M_1$  akzeptiert und terminiert ohne zu akzeptieren, wenn  $M_2$  akzeptiert
- Da eine der beiden Maschinen das Wort  $w$  akzeptieren muß, terminiert  $M$  und es gilt  $L(M)=L$

- **Jede endliche Sprache  $L$  ist entscheidbar**

- Jede endliche Sprache ist als Liste von Wörtern  $[w_1; \dots; w_n]$  darstellbar
- Bei Eingabe eines Wortes  $w$  vergleicht  $M$  das Wort mit dieser Liste

- **$L$  aufzählbar  $\Leftrightarrow$  es gibt ein entscheidbares**

$L' \subseteq \Sigma^* \times \Sigma^*$  mit  $L = \{w \mid \exists v. (w, v) \in L'\}$  (**Projektionssatz**)

- Aufwendiger Beweis, benötigt schrittweise Simulation von Maschinen

# PRÜFEN VON EIGENSCHAFTEN SUMMARISCH

- “ $x \in L$ ” kann automatisch geprüft werden für
    - Kontextsensitive und entscheidbare Sprachen  
(Folgt unmittelbar aus der Definition von Entscheidbarkeit)
    - Aber nicht für aufzählbare Sprachen  
(Folgt aus Existenz einer aufzählbaren, aber unentscheidbaren Sprache)
  - Für keine Sprachklasse kann getestet werden ob
    - eine Sprache  $L$  der Klasse leer ist
    - zwei Sprachen  $L_1$  und  $L_2$  der Klasse gleich sind
    - zwei Sprachen  $L_1$  und  $L_2$  der Klasse ineinander enthalten sind
    - der Durchschnitt zweier Sprachen der Klasse leer ist
- Beweise benötigen Beispiele für Sprachen, die nicht zur Klasse gehören

# GRENZEN DER SPRACHKLASSEN

- Entscheidbare, nicht kontextsensitive Sprache
  - Menge aller äquivalenten regulären Ausdrücke (gelesen als Text), wenn diese eine Iteration  $E^k = \underbrace{E \circ E \dots \circ E}_{k\text{-mal}}$  enthalten dürfen
  - Äquivalenztest benötigt exponentiell großen Speicherplatz
- Aufzählbare, nicht entscheidbare Sprache
  - Selbstanwendbarkeitsproblem: Menge aller Programme von Turingmaschinen, die bei Eingabe des eigenen Programms als Text terminieren
- Nicht aufzählbare Sprache
  - Totale Berechenbarkeit: Menge aller Programme von Turingmaschinen, die bei jeder Eingabe terminieren

## Mehr dazu in Theoretischer Informatik II

# ZUSAMMENFASSUNG: TYP-0 UND TYP-1 SPRACHEN

## • Turingmaschine als allgemeinstes Maschinenmodell

- Deterministischer endlicher Automat mit unendlichem Speicherband
- Gleiche Ausdruckskraft wie reale Computer (aber einfacher strukturiert)
- Nichtdeterministische Variante mit exponentiellem Aufwand simulierbar
- Äquivalent zu Typ-0 Grammatiken

- Bei linearer Bandbeschränkung äquivalent zu Typ-1 Grammatiken
- Entscheidbare Sprachen stehen zwischen  $\mathcal{L}_0$  und  $\mathcal{L}_1$

## • Wichtige Eigenschaften der Sprachklassen

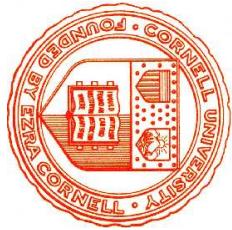
- Abgeschlossen unter  $\cup, \cap, R^*, \circ, ^*, h, h^{-1}$
- $\mathcal{L}_1$  und entscheidbare Sprachen zusätzlich unter  $-, -$
- Viele Eigenschaften können nicht automatisch getestet werden
  - Fast alle nichtrivialen Eigenschaften sind für keine Klasse entscheidbar
    - Für  $\mathcal{L}_0$  ist selbst das Wortproblem nicht mehr entscheidbar

# Theoretische Informatik I

## Einheit 5

### Rückblick

### Theoretische Informatik I



# THEMEN DER THEORETISCHEN INFORMATIK I & II

## • Mathematische Methodik in der Informatik

TI-1

## • Automatentheorie und Formale Sprachen

TI-1

- Endliche Automaten und reguläre Sprachen
- Lexikalische Analyse
- Kontextfreie Sprachen und Pushdown Automaten
- Syntaxanalyse und Semantik
- Allgemeine und kontextsensitive Sprachen

## • Theorie der Berechenbarkeit

TI-2

- Berechenbarkeitsmodelle
- Aufzählbarkeit und Entscheidbarkeit
- Unlösbare Probleme (Unentscheidbarkeit)

## • Komplexitätstheorie

TI-2

- Komplexitätsmaße und -klassen für Algorithmen und Probleme
- Nicht handhabbare Probleme ( $\mathcal{NP}$ -Vollständigkeit)

# REGULÄRE SPRACHEN

- **Endliche Automaten**

- Endliche Menge von Zuständen und Eingabesymbolen
- Verarbeitung von Eingabesymbolen ändert internen Zustand
- Erkannte Sprache: Abarbeitung endet in akzeptierendem Zustand
- Varianten: Deterministisch, nichtdeterministisch mit/ohne  $\epsilon$ -Übergänge
- Umwandlung in deterministische Variante über Teilmengenkonstruktion

- **Reguläre Ausdrücke**

- Algebraische Notation für Sprachen:  $\epsilon$ ,  $\emptyset$ , Symbole von  $\Sigma$ ,  $+$ ,  $\circ$ ,  $*$
- Umwandelbar in  $\epsilon$ -NEAs (iterative Konstruktion)
- DEAs umwandelbar durch Zustandselimination im VNEAs

- **Grammatiken**

- Beschreibung des Aufbaus von Sprachen durch Produktionsregeln
- Erzeugte Sprache: schrittweise Ableitung endet in Terminalworten
- Typ-3 (rechtsslineare) Grammatiken sind äquivalent zu  $\epsilon$ -NEAs
- Direkte Umwandlung zwischen Produktionen und Überführungsfunktion

# EIGENSCHAFTEN REGULÄRER SPRACHEN

- **Abschlußeigenschaften**

- Operationen  $\cup$ ,  $\cap$ ,  $-$ ,  ${}^R$ ,  $\circ$ ,  $*$ ,  $h$ ,  $h^{-1}$  erhalten Regularität von Sprachen
- Verwendbar zum Nachweis von Regularität oder zur Widerlegung

- **Automatische Prüfungen**

- Man kann testen ob eine reguläre Sprache leer ist
- Man kann testen ob ein Wort zu einer regulären Sprache gehört
- Man kann testen ob zwei reguläre Sprachen gleich sind

- **Minimierung von Automaten**

- Ein Automat kann minimiert werden indem man äquivalente Zustände zusammenlegt und unerreichbare Zustände entfernt

- **Pumping Lemma**

- Wiederholt man einen bestimmten Teil ausreichend großer Worte einer regulären Sprache beliebig oft, so erhält man immer ein Wort der Sprache
- Verwendbar zur Widerlegung von Regularität

## Kompliziertere Struktur als reguläre Sprachen

- **Kontextfreie Grammatiken**

- Produktionsregeln ersetzen einzelne Variablen durch beliebige Worte
- Ableitungsbäume beschreiben Struktur von Terminalworten (**Compiler!**)
- Ableitungsbäume entsprechen Links- (oder Rechts-)ableitungen
- Programmiersprachen brauchen eindeutig bestimmmbare Ableitungsbäume

- **Pushdown-Automaten**

- Nichtdeterministischer endlicher Automat mit Stack und  $\epsilon$ -Übergängen
- Erkennung von Worten durch Endzustand oder leeren Stack
- Analyse durch Betrachtung von **Konfigurationsübergängen**
  - Nichtdeterministische PDAs äquivalent zu kontextfreien Grammatiken
    - Umwandlung von **Konfigurationsübergängen in Regeln** und umgekehrt
  - Deterministische PDAs weniger mächtig (nur eindeutige Typ-2 Sprachen)

# EIGENSCHAFTEN KONTEXTFREIER SPRACHEN

- **Abschlußeigenschaften**

- Operationen  $\cup$ ,  $\overset{R}{\cup}$ ,  $\circ$ ,  $*$ ,  $\sigma$ ,  $h^{-1}$  erhalten Kontextfreiheit von Sprachen
- Keine Abgeschlossenheit unter  $\cap$ ,  $-$ ,  $-$

- **Automatische Prüfungen**

- Man kann testen ob eine kontextfreie Sprache leer ist
  - Man kann testen ob ein Wort zu einer kontextfreien Sprache gehört
  - Man kann nicht testen ob zwei kontextfreie Sprachen gleich sind
- Viele wichtige Fragen sind nicht automatisch prüfbar

- **Pumping Lemma**

- Wiederholt man bestimmte Teile ausreichend großer Worte einer kontext-freien Sprache beliebig oft, so erhält man immer ein Wort der Sprache
- Viele einfache Sprachen sind nicht kontextfrei

# JENSEITS VON KONTEXTFREIEN SPRACHEN

## • Turingmaschinen als Maschinenmodell

- Deterministischer endlicher Automat mit unendlichem Speicherband
- Äquivalent zu realen Computern
- Viele gleichmächtige Varianten
- Simulation nichtdeterministischer Maschine ist exponentiell

## • Typ-0 Sprachen

- Keine Einschränkung an Produktionsregeln
- Auch Terminalsymbole und ganze Wörter dürfen ersetzt werden
- Typ-0 Grammatiken sind äquivalent zu Turingmaschinen
  - Umwandlung von Konfigurationsübergängen in Regeln und umgekehrt

## • Entscheidbare Sprachen

- Sprachen von Turingmaschinen, die immer terminieren

## • Typ-1 Sprachen

- Produktionsregeln der Grammatiken dürfen Wörter nicht verkleinern
- Sprachen von Turingmaschinen mit linear beschränktem Band

# ERFRAUEN?