Vorlesung "Automaten und Formale Sprachen" Sommersemester 2017

Sebastian Küpper
Übungsleitung: Christina Mika

Das heutige Programm:

- Organisatorisches
  - Vorstellung
  - Ablauf der Vorlesung und der Übungen
  - Prüfung & Klausur
  - Literatur & Folien
- Einführung und Motivation: “Automaten und Formale Sprachen”
- Inhalt der Vorlesung

Wer sind wir?

Dozent: Sebastian Küpper

- Raum LF 261
- E-Mail: `sebastian.kuepper@uni-due.de`

Übungsleitung: Christina Mika

- Raum LF 261
- `christine.mika@uni-due.de`

Web-Seite: `www.ti.inf.uni-due.de/teaching/ss2017/afs/`

Vorlesungstermine

Termin:

- Dienstag, 12:15-13:45 Uhr im Raum LX 1203

Termine der Übungsgruppen

Übungsgruppen:

Gruppe	Tag	Uhrzeit	Raum	Sprache
1	Di	8:00 – 10:00	LE 120	Englisch
2	Di	16:00 – 18:00	LE 103	Deutsch
3	Mi	10:00 – 12:00	LE 120	Deutsch
4	Do	10:00 – 12:00	LK 051	Deutsch
5	Fr	8:00 – 10:00	LC 137	Deutsch
6	Fr	10:00 – 12:00	LC 137	Deutsch

Hinweise zu den Übungen

- Die Übungen beginnen in der dritten Vorlesungswoche am Dienstag, den 2. Mai.
- Bitte versuchen Sie, sich möglichst gleichmäßig auf die Übungen zu verteilen.
- Besuchen Sie die Übungen! Diesen Stoff kann man nur durch regelmäßiges Üben erlernen. Auswendiglernen hilft nicht besonders viel.
- Die Übungsblätter (in Deutsch und Englisch) werden jeweils am Dienstag der Vorwoche ins Netz gestellt. Das erste Übungsblatt wird am 25.4. bereitgestellt.

Hinweise zu den Übungen

- Abgabe der gelösten Aufgaben bis Dienstag der folgenden Woche, 8:00 Uhr.
- Einwurf in den Briefkasten neben dem Raum LF 259 oder Abgabe per Moodle.
- Bitte geben Sie auf Ihrer Lösung *deutlich* die Vorlesung, Ihren Namen, Ihre Matrikelnummer *und* Ihre Gruppennummer an.
- Sie dürfen in *Paaren* abgeben. Bei Abgaben in Paaren sollte die Abgabe nur ein Mal eingerecht, aber mit beiden Namen versehen werden.

Hinweise zu den Übungen

Wir verwenden Moodle, um:

- die Aufgabenblätter zur Verfügung zu stellen,
- die Hausaufgaben elektronisch (nur PDF!) abzugeben *und*
- um Diskussionsforen bereitzustellen.

Moodle-Plattform an der Universität Duisburg-Essen:
`http://moodle.uni-due.de/` (siehe auch Link auf der Webseite)

Bitte legen Sie dort einen Zugang an (falls noch nicht vorhanden) und tragen Sie sich in den Kurs "Automaten und Formale Sprachen 2017" (Sommersemester 2017 $\rightarrow$ Ingenieurwissenschaften $\rightarrow$ Informatik und Angewandte Kognitionswissenschaft) ein. Bitte mit Uni-Kennung anmelden!

Tutorium

In diesem Semester wird es ein Tutorium geben, das wechselweise in Deutsch und in Englisch von Christina Mika gehalten wird. In dem Tutorium können Fragen zur Vorlesung besprochen werden; wenn allerdings im Vorfeld keine Fragen an Christina Mika gesendet werden (per Moodle-Forum oder per Mail), dann entfällt der jeweilige Termin ersatzlos. Das Tutorium findet an den folgenden Terminen statt:

1. Donnerstag, 14:00-16:00 (Deutsch, in geraden Kalenderwochen)
2. Freitag, 14:00-16:00 (Englisch, in ungeraden Kalenderwochen)

Erstmals findet das Tutorium am 4. Mai statt.

Prüfung

Es gibt mehrere Möglichkeiten, die Vorlesung prüfen zu lassen ...

- Klausur (für BAI & ISE & Nebenfach-Studierende).

  Termin: 22. August 2017, 8:30 Uhr
- Mündliche Prüfung (für BAIs, die diese Vorlesung mündlich prüfen lassen; Alternative: mündliche Prüfung in "Rechnernetze und Sicherheit")

  Voraussichtlicher Termin: 21.-25. August 2017

  Im Allgemeinen findet diese mündliche Prüfung als Kombiprüfung/Modulprüfung gemeinsam mit "Berechenbarkeit und Komplexität" statt. Es gibt Ausnahmen für Studierende, die im Sommersemester beginnen und beide Vorlesungen in größerem Abstand hören.

Anmeldung über das Prüfungsamt

Prüfung

Hinweise:

- Die Vorlesung heißt “Automaten und Formale Sprachen”.

  Das Modul, das “Automaten und Formale Sprachen” & “Berechenbarkeit und Komplexität” umfasst, heißt “Theoretische Informatik”.
- Für ISE trägt die Vorlesung *nach der alten Prüfungsordnung* alleine den Namen “Theoretische Informatik”.

Prüfung

Wenn Sie 50% der Übungspunkte erzielt haben, so erhalten Sie einen Bonus für die Prüfung. (Für das Vorrechnen in der Übung gibt es 10 Extrapunkte.)

Auswirkung: Verbesserung um eine Notenstufe; z.B. von 2,3 auf 2,0.

Bonuspunkte aus dem SS 2016 (oder früher) gelten nicht mehr!

Für die mündliche Modulprüfung "Theoretische Informatik" (Kombiprüfung) ist es erforderlich, den Bonus für jede der beiden Vorlesungen ("Automaten und Formale Sprachen" & "Berechenbarkeit & Komplexität") zu erzielen, um eine bessere Note zu erhalten.

Der Bonus ist keine Voraussetzung für die Teilnahme an einer Prüfung.

LuDi - Lern- und Diskussionszentrum

Zur Unterstützung in der Studieneingangsphase bietet das LuDi einen Raum zum gemeinsamen Lernen und Nachfragen, betreut durch studentische Tutoren höherer Semester. Im LuDi erhalten Sie Hausaufgabenhilfe, können Fragestellungen aus Vorlesungen diskutieren und sich gemeinsam in der Klausurphase vorbereiten. Es gibt ein LuDi zu Informatik-nahen Veranstaltungen im *LF 031* zu den folgenden Zeiten:

- Montag 11-14 Uhr
- Mittwoch 12-16 Uhr
- Freitag 11-14 Uhr

Zu allen weiteren Zeiten steht das LuDi als Arbeitsraum zur Verfügung.

LuDi - Lern- und Diskussionszentrum

Weitere Informationen finden Sie unter der URL `https://www.uni-due.de/iw/de/studium/ludi-inko` sowie in der Facebook-Gruppe des LuDi: `http://bit.ly/LuDi-INKO`.
Ein analoges Angebot existiert auch für Mathematik-nahe Veranstaltungen, immer Montags bis Freitags von 10-18 Uhr im BC 520.

Literatur

Die Vorlesung basiert im wesentlichen auf folgendem Buch:

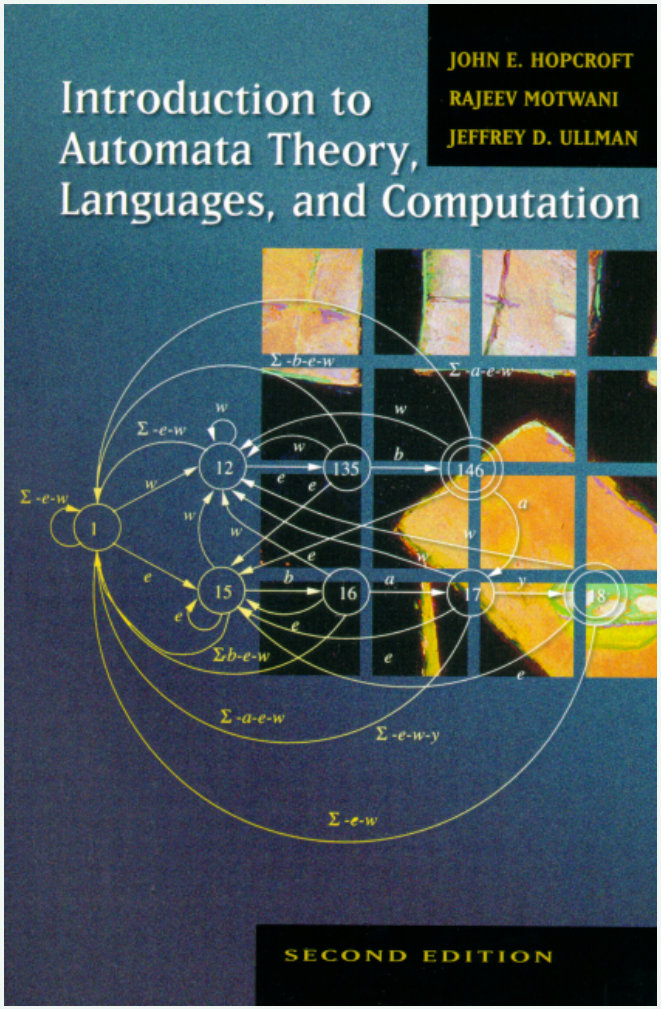
- Uwe Schöning: *Theoretische Informatik – kurzgefaßt*. Spektrum, 2008. (5. Auflage)

Weitere relevante Bücher:

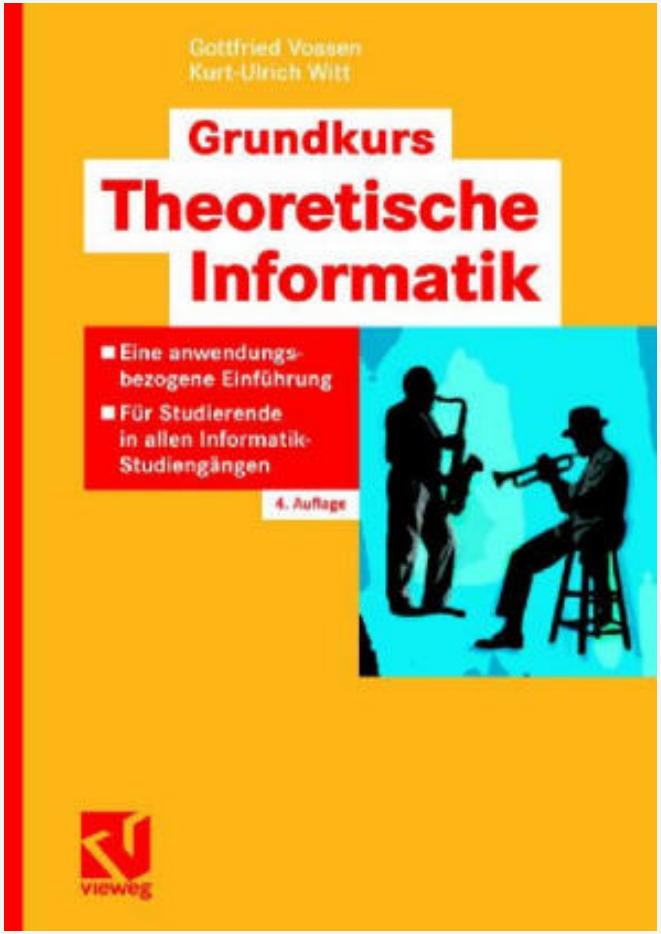
- Neuauflage eines alten Klassikers: Hopcroft, Motwani, Ullman: *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 2001.
- *Auf Deutsch:* Hopcroft, Motwani, Ullman: *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*, Pearson, 2002.
- Vossen, Witt: Grundkurs Theoretische Informatik, vieweg, 2006.
- Asteroth, Baier: Theoretische Informatik, Pearson, 2003.

Literatur

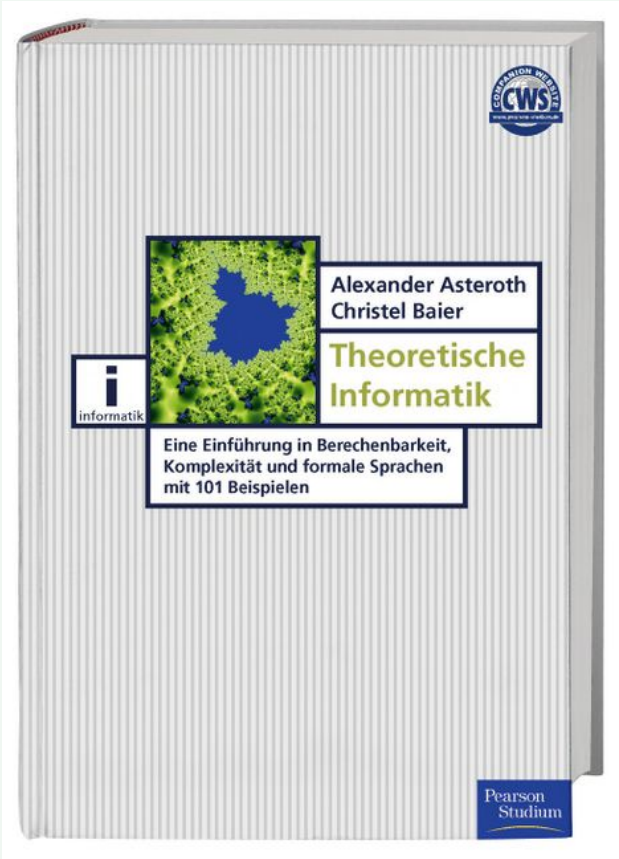

Uwe Schöning
Theoretische Informatik
– kurz gefasst
5. Auflage
HochschulTaschenbuch
Spektrum


JOHN E. HOPCROFT
RAJEEV MOTWANI
JEFFREY D. ULLMAN
Introduction to
Automata Theory,
Languages, and Computation
SECOND EDITION

Literatur


Gottfried Vossen
Kurt-Ulrich Witt
Grundkurs
Theoretische Informatik
Eine anwendungsbezogene Einführung
Für Studierende in allen Informatik-Studiengängen
4. Auflage
vieweg

Literatur


Alexander Asteroth
Christel Baier
Theoretische Informatik
informatik
Eine Einführung in Berechenbarkeit, Komplexität und formale Sprachen mit 101 Beispielen
Pearson Studium

Folien

Folien werden

- auf der Webseite bereitgestellt
- regelmäßig aktualisiert

Die Folien basieren auf den Folien aus dem letzten Jahr. Die behandelten Inhalte werden sich im Vergleich zur vorherigen Veranstaltung nicht wesentlich ändern.
Sie können daher bei Bedarf auf die Folien des Vorjahrs zurückgreifen (`www.ti.inf.uni-due.de/teaching/ss2016/afs/downloads/`).

Unter
`www.ti.inf.uni-due.de/teaching/ss2014/afs/downloads/`
gibt es auch eine englische Übersetzung der Folien aus dem Jahr 2014.

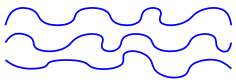
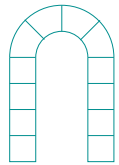
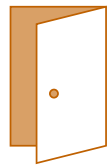
Adventure-Problem

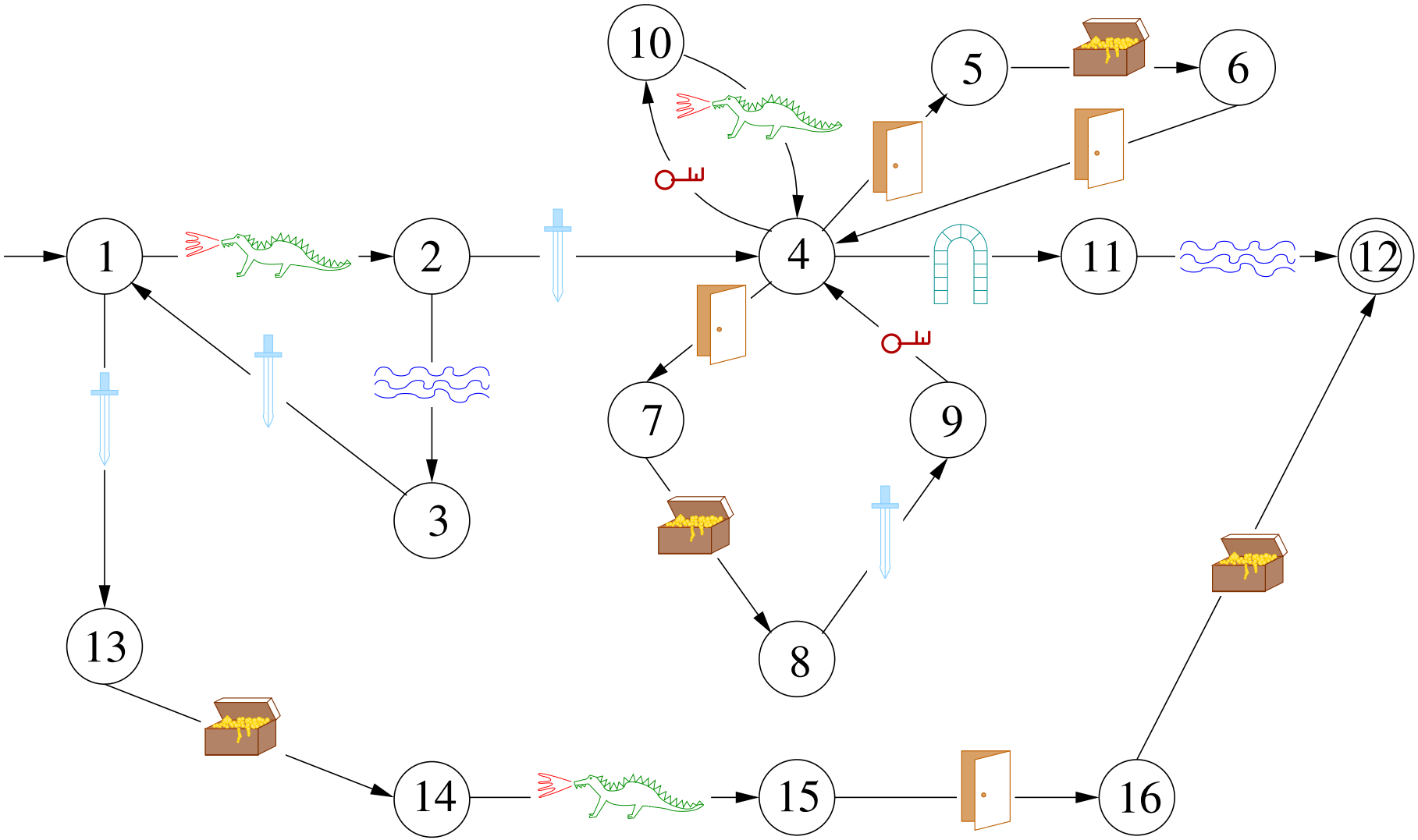
Zum Aufwärmen: wir betrachten das sogenannte Adventure-Problem, bei dem ein Abenteurer/eine Abenteurerin einen Weg durch ein Adventure finden muss.

(Später wird dann erklärt, was das eigentlich mit theoretischer Informatik zu tun hat.)

Adventure-Problem

Adventures bestehen aus Graphen bzw. Automaten, die mit folgenden Symbolen markiert sind:

- Drachen: 
- Schwert: 
- Fluss: 
- Torbogen: 
- Tür: 
- Schlüssel: 
- Schatz: 


10
5
6
1
2
4
11
12
7
9
3
13
8
14
15
16

Adventure-Problem (Level 1)

Natürlich gibt es bestimmte Regeln, die bei einem erfolgreichen Abenteuer zu beachten sind:

Die Schatz-Regel

Man muss mindestens zwei Schätze finden.

Die Tür-Regel

Durch eine Tür kann man nur gehen, wenn man zuvor einen Schlüssel gefunden hat. (Dieser Schlüssel darf aber dann beliebig oft verwendet werden.)

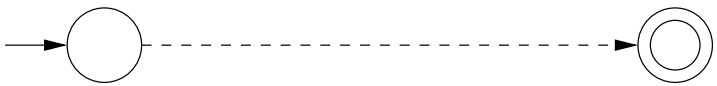
Adventure-Problem (Level 1)

Die Drachen-Regel

Unmittelbar nach der Begegnung mit einem Drachen muss man in einen Fluss springen, da uns der Drache in Brand stecken wird. Dies gilt nicht mehr, sobald man ein Schwert besitzt, mit dem man den Drachen vorher töten kann.

Bemerkung: Drachen, Schätze und Schlüssel werden "nachgefüllt", sobald man das entsprechende Feld verlassen hat.

Gesucht ist der kürzeste Weg, von einem Anfangszustand zu einem Endzustand, der alle diese Bedingungen erfüllt:



Adventure-Problem (Level 2)

Neue Tür-Regel

Die Schlüssel sind magisch und verschwinden sofort, nachdem eine Tür mit ihnen geöffnet wurde. Sobald man eine Tür durchschritten hat, schließt sie sich sofort wieder.

Adventure-Problem (Level 3)

Neue Drachen-Regel

Auch Schwerter werden durch das Drachenblut unbenutzbar, sobald man einen Drachen damit getötet hat. Außerdem werden Drachen sofort wieder "ersetzt".
Es gibt jedoch immer noch die Option, ein Schwert nicht zu benutzen und nach der Begegnung mit dem Drachen in den Fluss zu springen.

Außerdem bleibt die neue Tür-Regel bestehen.

Adventure-Problem (Level 4)

Schlüssel-Regel

Der magische Torbogen kann nur passiert werden, wenn man keinen Schlüssel besitzt.

Schwert-Regel

Ein Fluss kann nur passiert werden, wenn man kein Schwert besitzt (weil man sonst ertrinkt!).

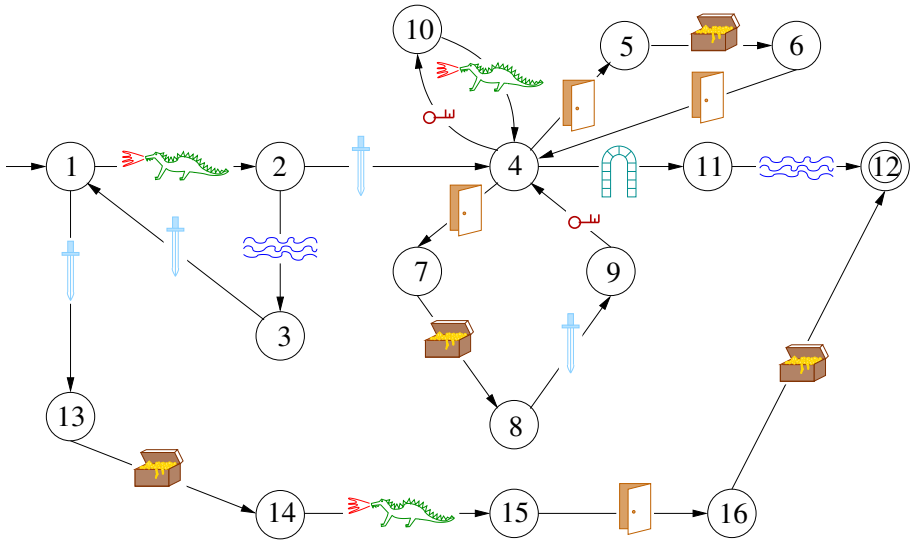
Das Wegwerfen von Schlüsseln oder Schwertern ist nicht erlaubt.

Think-Pair-Share

Wir werden nun eine kleine Aufgabe bearbeiten und dazu das Verfahren Think-Pair-Share verwenden. Wir werden Aufgaben dieser Art regelmäßig durchführen, da sie dazu beitragen können, einen größeren Lernerfolg zu erzielen.
Sie erhalten gleich eine Aufgabenstellung, die zunächst jeder für sich bearbeiten sollte. Nach drei Minuten beginnt die Paararbeitsphase, in der Sie Ihre Ergebnisse mit Ihrem Sitzpartner besprechen. Nach weiteren vier Minuten bitte ich Sie, Ihre Ergebnisse dem Plenum vorzustellen. Gibt es Fragen zu dem Vorgehen?
Bitte suchen Sie sich ein Level (1, 2, 3 oder 4) aus und ermitteln Sie den kürzesten Pfad von einem Start- zu einem Endzustand, der alle Adventure-Regeln des Levels erfüllt.

Level 1: Man muss mindestens zwei Schätze finden. Bevor man eine Tür durchqueren kann, muss man mindestens einen Schlüssel auflesen. Nach einer Begegnung mit einem Drachen muss man unmittelbar in einen Fluss springen, außer man hat irgendwann vorher ein Schwert aufgelesen.

Level 2: Schlüssel verschwinden beim Einsatz an einer Tür; man muss also vor Passieren der n-ten Tür bereits mindestens n Schlüssel gesammelt haben.



Level 3: Zusätzlich kann jedes Schwert nur einmal verwendet werden um einen Drachen zu töten.

Level 4: Ein Torbogen kann nur passiert werden, wenn man keinen Schlüssel besitzt und ein Fluss nur passiert werden, wenn man kein Schwert besitzt.

Adventure-Problem (Level 1)

Fragen (Level 1)

- Gibt es in dem Beispiel eine Lösung? ▸ Adventure

  $\rightsquigarrow$ Ja! Die kürzeste Lösung ist
  $1, 2, 3, 1, 2, 4, 10, 4, 5, 6, 4, 5, 6, 4, 11, 12$ (Länge 16).
- Gibt es ein allgemeines Lösungsverfahren, das – gegeben ein Adventure in Form eines Automaten – immer bestimmen kann, ob es eine Lösung gibt?

  $\rightsquigarrow$ Ja! Wir werden dieses Verfahren noch kennenlernen.

  Um das Verfahren implementieren zu können, benötigen wir auch formale Beschreibungen der Regeln (Tür-Regel, Drachen-Regel, Schatz-Regel).

Adventure-Problem (Level 2)

Fragen (Level 2)

- Gibt es in dem Beispiel eine Lösung? ▸ Adventure

  $\rightsquigarrow$ Ja! Die kürzeste Lösung ist 1, 2, 3, 1, 2, 4, 10, 4, 7, 8, 9, 4, 7, 8, 9, 4, 11, 12. (Länge 18)
- Gibt es ein allgemeines Lösungsverfahren?

  $\rightsquigarrow$ Ja! Wir werden dieses Verfahren noch kennenlernen.
- Warum ist das Problem jetzt schwieriger?

  $\rightsquigarrow$ Wir haben jetzt durch die Schlüssel eine Art Zähler eingeführt.

Adventure-Problem (Level 3)

Fragen (Level 3)

- Gibt es in dem Beispiel eine Lösung? ▸ Adventure

  $\rightsquigarrow$ Ja! Die kürzeste Lösung ist 1, 2, 3, 1, 2, 4, 10, 4, 7, 8, 9, 4, 7, 8, 9, 4, 11, 12. (Länge 18)

- Gibt es ein allgemeines Lösungsverfahren?

  $\rightsquigarrow$ Ja! Dieses Problem ist "gerade noch" lösbar. Eine genaue Laufzeit kann nicht angegeben werden. (Mögliche Lösungen werden in der Vorlesung voraussichtlich nicht behandelt.)

- Warum wird das Problem schwieriger?

  $\rightsquigarrow$ Durch die Schwerter haben wir einen weiteren Zähler hinzubekommen. Weitere Zähler (d.h., drei oder mehr) machen das Problem nicht wesentlich schwieriger.

Adventure-Problem (Level 4)

Fragen (Level 4)

- Gibt es in dem Beispiel eine Lösung? ▸ Adventure

  $\rightsquigarrow$ Ja! Die kürzeste Lösung ist 1, 2, 3, 1, 2, 4, 10, 4, 7, 8, 9, 4, 10, 4, 5, 6, 4, 11, 12. (Länge 19)

- Gibt es ein allgemeines Lösungsverfahren?

  $\rightsquigarrow$ Nein! Es handelt sich hier um ein sogenanntes unentscheidbares Problem. Wir werden in der Vorlesung "Berechenbarkeit und Komplexität" beweisen, dass es tatsächlich kein Lösungsverfahren gibt.

Adventure-Problem (Level 4)

Fragen (Level 4)

- Warum wird das Problem schwieriger?

  $\rightsquigarrow$ Wir haben jetzt nicht nur zwei Zähler, sondern können diese auch auf Null testen. Damit hat unser Modell bereits eine Mächtigkeit, bei der viele Problemstellungen unentscheidbar werden.

  Man beachte: Computer-Programme sind mindestens so mächtig, denn es ist ganz einfach zwei Zähler einzuführen und ebenso sind Null-Tests möglich!

Adventure-Problem und Formale Sprachen

(Formale) Sprachen

Sprachen = Mengen von Wörtern

Im Beispiel: Menge aller Pfade in einem Adventure; Menge aller zulässigen Pfade in einem Adventure; Menge aller Pfade, die die Tür-Regel erfüllen (unabhängig vom Adventure), ...

Im Allgemeinen: Mengen von Wörtern, die bestimmten Bedingungen genügen (zum Beispiel: Menge aller korrekt geklammerten arithmetischen Ausdrücke; Menge aller syntaktisch korrekter Java-Programme; ...)

Adventure-Problem und Formale Sprachen

Automaten und Formale Sprachen

Sprachen enthalten im Allgemeinen unendliche viele Wörter.

Daher: Man benötigt endliche Beschreibungen für unendliche Sprachen.
Mögliche endliche Beschreibungen sind Automaten (wie im Beispiel), Grammatiken (ähnlich zu Grammatiken für natürliche Sprachen) oder reguläre Ausdrücke.

Es gibt auch Beschreibungen in Worten (Tür-Regel, etc.), aber diese müssen – damit sie eindeutig sind und mechanisch weiterverarbeitet werden können – formalisiert werden.

Adventure-Problem und Formale Sprachen

Fragestellungen

Typische Fragen in diesem Zusammenhang sind:

- Ist eine bestimmte Sprache L leer oder enthält sie ein Wort? $L = \emptyset$?
- Ist ein gegebenes Wort w in der Sprache? $w \in L$?
- Sind zwei Sprachen ineinander enthalten? $L_1 \subseteq L_2$?

Wir betrachten verschiedene Algorithmen, die solche Fragen beantworten können.

Adventure-Problem und Formale Sprachen

Die einzelnen Level des Adventures entsprechen in etwa folgenden Sprachklassen:

- Level 1 $\rightarrow$ reguläre Sprachen
- Level 2 $\rightarrow$ kontextfreie Sprachen
- (Level 3 $\rightarrow$ Petri-Netz-Sprachen)
  Stattdessen: wir behandeln kontextsensitiven Sprachen
- Level 4 $\rightarrow$ Chomsky-0-Sprachen (semi-entscheidbare Sprachen)

Kontextsensitive und semi-entscheidbare Sprachen werden im Detail erst in der Nachfolgervorlesung “Berechenbarkeit und Komplexität” behandelt.

Vom Nutzen der theoretischen Informatik

- Wie kann man unendliche Strukturen (Sprachen) durch endliche Beschreibungen (Automaten, Grammatiken) erfassen?
- Es geht um die Fragen: Was ist berechenbar? Wie sehen die dazugehörigen Algorithmen aus? Was sind wirklich harte Probleme?
- Es gibt zahlreiche Anwendungen, beispielsweise in folgenden Gebieten:
  - Suchen in Texten (reguläre Ausdrücke)
  - Syntax von (Programmier-)Sprachen und Compilerbau
  - Systemverhalten modellieren
  - Programmverifikation

Inhalt der Vorlesung

Automatentheorie und Formale Sprachen

- Sprachen, Grammatiken und Automaten
- Chomsky-Hierarchie (verschiedene Klassen von Sprachen)
- Reguläre Sprachen, kontextfreie Sprachen
- Wie kann man zeigen, dass eine Sprache *nicht* zu einer bestimmten Sprachklasse gehört? (Pumping-Lemma)
- Wortproblem (Gehört ein Wort zu einer bestimmten Sprache?)
- Abschlusseigenschaften (Ist der Schnitt zweier regulärer Sprachen wieder regulär?)

Notation: Mengen und Funktionen

Menge

Menge M von Elementen, wird beschrieben als Aufzählung

$$M = \{0, 2, 4, 6, 8, \dots\}$$

oder als Menge von Elementen mit einer bestimmten Eigenschaft

$$M = \{n \mid n \in \mathbb{N}_0 \text{ und } n \text{ gerade}\}.$$

Allgemeines Format:

$$M = \{x \mid P(x)\}$$

(M ist Menge aller Elemente x, die die Eigenschaft P erfüllen.)

Notation: Mengen und Funktionen

Bemerkungen:

- Die Elemente einer Menge sind ungeordnet, d.h., ihre Ordnung spielt keine Rolle. Beispielsweise gilt:

$$\{1, 2, 3\} = \{1, 3, 2\} = \{2, 1, 3\} = \{2, 3, 1\} = \{3, 1, 2\} = \{3, 2, 1\}$$

- Ein Element kann nicht "mehrfach" in einer Menge auftreten. Es ist entweder in der Menge, oder es ist nicht in der Menge. Beispielsweise gilt:

$$\{1, 2, 3\} \neq \{1, 2, 3, 4\} = \{1, 2, 3, 4, 4\}$$

Notation: Mengen und Funktionen

Element einer Menge

Wir schreiben $a \in M$, falls ein Element a in der Menge M enthalten ist.

Anzahl der Elemente einer Menge

Für eine Menge M gibt $|M|$ die Anzahl ihrer Elemente an.

Teilmengenbeziehung

Wir schreiben $A \subseteq B$, falls jedes Element von A auch in B enthalten ist. Die Relation $\subseteq$ heißt auch *Inklusion*.

Leere Menge

Mit $\emptyset$ oder $\{\}$ bezeichnet man die leere Menge. Sie enthält keine Elemente und ist Teilmenge jeder anderen Menge.

Notation: Mengen und Funktionen

Vereinigung

Die Vereinigung zweier Mengen A, B ist die Menge M, die die Elemente enthält, die in A oder B vorkommen. Man schreibt dafür $A \cup B$.

$$A \cup B = \{x \mid x \in A \text{ oder } x \in B\}$$

Schnitt

Der Schnitt zweier Mengen A, B ist die Menge M, die die Element enthält, die sowohl in A als auch in B vorkommen. Man schreibt dafür $A \cap B$.

$$A \cap B = \{x \mid x \in A \text{ und } x \in B\}$$

Notation: Mengen und Funktionen

Kreuzprodukt

Seien A, B zwei Menge. Die Menge $A \times B$ ist die Menge aller Paare (a, b), wobei das erste Element des Paars aus A, das zweite aus B kommt.

$$A \times B = \{(a, b) \mid a \in A, b \in B\}$$

Beispiel:

$\{1, 2\} \times \{3, 4, 5\} = \{(1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5)\}$

Es gilt: $|A \times B| = |A| \cdot |B|$ (für endliche Menge A, B).

Notation: Mengen und Funktionen

Bemerkungen:

- Wir betrachten nicht nur Paare, sondern auch sogenannte Tupel, bestehend aus mehreren Elementen. Ein Tupel $(a_1, \ldots, a_n)$ bestehend aus n Elementen heißt auch n-Tupel.
- In einem Tupel sind die Element geordnet! Beispielsweise gilt:

$$(1, 2, 3) \neq (1, 3, 2) \in \mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}_0$$

- Ein Element kann "mehrfach" in einem Tupel auftreten. Tupel unterschiedlicher Länge sind immer verschieden. Beispielsweise:

$$(1, 2, 3, 4) \neq (1, 2, 3, 4, 4)$$

Merke: Runde Klammern $(,)$ und geschweifte Klammern $\{,\}$ stehen für ganz verschiedene mathematische Objekte!

Notation: Mengen und Funktionen

Potenzmenge

Sei M eine Menge. Die Menge $\mathcal{P}(M)$ ist die Menge aller Teilmengen von M.

$$\mathcal{P}(M) = \{A \mid A \subseteq M\}$$

Beispiel:
$\mathcal{P}(\{1,2,3\}) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$.

Es gilt: $|\mathcal{P}(M)| = 2^{|M|}$ (für eine endliche Menge M).

Notation: Mengen und Funktionen

Funktion

$$\begin{aligned} f : A &\to B \\ a &\mapsto f(a) \end{aligned}$$

Die Funktion f bildet ein Element $a \in A$ auf ein Element $f(a) \in B$ ab. Dabei ist A der *Definitionsbereich* und B der *Wertebereich*.

Beispiel (Quadratfunktion):

$$f : \mathbb{Z} \to \mathbb{N}_0, \quad f(n) = n^2$$

$$\ldots, -3 \mapsto 9, -2 \mapsto 4, -1 \mapsto 1, 0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 4, 3 \mapsto 9, \ldots$$

Dabei ist $\mathbb{N}_0$ die Menge der natürlichen Zahlen (mit der Null) und $\mathbb{Z}$ die Menge der ganzen Zahlen.

Fragestellungen zu dieser Vorlesungseinheit

Zum Ende einer jeden Vorlesungseinheit betrachten wir im Regelfall drei Fragestellungen, die mithilfe der in dieser Einheit besprochenen Inhalte beantwortet werden sollen. In der darauffolgenden Einheit können zu Beginn mögliche Antworten gesammelt werden. Diese Antworten werden nur in der Vorlesung besprochen und nicht online verfügbar gemacht.

Fragen zur ersten Vorlesungseinheit

- Was sind typische Fragestellungen die wir im Zusammenhang mit formalen Sprachen beantworten wollen und was bedeuten diese Fragestellungen in Bezug auf das Adventure-Beispiel?
- Wie werden Mengen, Tupel und Funktionen notiert?
- Was bedeuten die Operatoren $\in$, $|\cdot|$, $\subseteq$, $\cup$, $\cap$, $\times$, $\mathcal{P}$?

Wörter

Wort

Sei Σ ein Alphabet, d.h., eine endliche Menge von Zeichen. Dann bezeichnet man mit Σ^* die Menge aller Wörter, d.h., die Menge aller (endlichen) Zeichenketten mit Zeichen aus Σ.
Das leere Wort (das Wort der Länge 0) wird mit ε bezeichnet.
Die Menge aller nicht-leeren Wörter über Σ wird mit Σ^+ bezeichnet.
Mit $|w|$ bezeichnen wir die Länge des Wortes w.

Beispiel: Sei $\Sigma = \{a, b, c\}$. Dann sind mögliche Wörter aus Σ^*:

$$\varepsilon, a, b, aa, ab, bc, bbbab, \ldots$$

Ein anderes mögliches Alphabet Σ (mit den Zeichen “Drache”, “Schlüssel”, . . .) haben wir im vorigen Beispiel kennengelernt.

Sprachen

Sprache

Sei Σ ein Alphabet.
Eine (formale) Sprache L über Σ ist eine beliebige Teilmenge von Σ^* ($L \subseteq \Sigma^*$).

Beispiel: sei $\Sigma = \{(,), +, -, *, /, a\}$, so können wir die Sprache *EXPR* der korrekt geklammerten Ausdrücke definieren. Es gilt beispielsweise:

- $(a - a) * a + a/(a + a) - a \in EXPR$
- $(((a))) \in EXPR$
- $((a+) - a(\notin EXPR$

Beispielsprachen

Alphabete und Sprachen:

- $\Sigma_1 = \{(,), +, -, *, /, a\}$
  $L_1 = EXPR = \{w \in \Sigma_1^* \mid w \text{ ist ein arithmetischer Ausdruck}\}$
- $\Sigma_2 = \{\mathrm{a}, \ldots, \mathrm{z}, \ddot{\mathrm{a}}, \ddot{\mathrm{u}}, \ddot{\mathrm{o}}, \ss, ., ,, :, \ldots\}$
  $L_2 = \text{Grammatikalisch korrekte deutsche Sätze}$
- $\Sigma_3 = \text{beliebig}$
  $L_3 = \emptyset,\ L_3' = \{\varepsilon\}$
- Typische Sprachen über dem Alphabet $\Sigma_4 = \{a, b, c\}$:
  - $L_4 = \{w \in \Sigma_4^* \mid w \text{ enthält } aba \text{ als Teilwort}\}$
  - $L_5 = \{a^n b^n \mid n \in \mathbb{N}_0\}$
  - $L_6 = \{a^n b^n c^n \mid n \in \mathbb{N}_0\}$

  (wobei $x^n = \underbrace{x \ldots x}_{n\text{-mal}}$)

Grammatiken (Einführung)

Grammatiken in der Informatik sind – ähnlich wie Grammatiken für natürliche Sprachen – ein Mittel, um alle syntaktisch korrekten Sätze (hier: Wörter) einer Sprache zu erzeugen.

Beispiel: Vereinfachte Grammatik zur Erzeugung natürlichsprachiger Sätze

$$
\begin{aligned}
\langle \text{Satz} \rangle &\to \langle \text{Subjekt} \rangle \langle \text{Prädikat} \rangle \langle \text{Objekt} \rangle \\
\langle \text{Subjekt} \rangle &\to \langle \text{Artikel} \rangle \langle \text{Attribut} \rangle \langle \text{Substantiv} \rangle \\
\langle \text{Artikel} \rangle &\to \varepsilon \\
\langle \text{Artikel} \rangle &\to \text{der} \\
\langle \text{Artikel} \rangle &\to \text{die} \\
\langle \text{Artikel} \rangle &\to \text{das} \\
\langle \text{Attribut} \rangle &\to \varepsilon
\end{aligned}
$$

Grammatiken (Einführung)

$$
\begin{aligned}
\langle \text{Attribut} \rangle &\to \langle \text{Adjektiv} \rangle \\
\langle \text{Attribut} \rangle &\to \langle \text{Adjektiv} \rangle \langle \text{Attribut} \rangle \\
\langle \text{Adjektiv} \rangle &\to \text{kleine} \\
\langle \text{Adjektiv} \rangle &\to \text{bissige} \\
\langle \text{Adjektiv} \rangle &\to \text{große} \\
\langle \text{Substantiv} \rangle &\to \text{Hund} \\
\langle \text{Substantiv} \rangle &\to \text{Katze} \\
\langle \text{Prädikat} \rangle &\to \text{jagt} \\
\langle \text{Objekt} \rangle &\to \langle \text{Artikel} \rangle \langle \text{Attribut} \rangle \langle \text{Substantiv} \rangle
\end{aligned}
$$

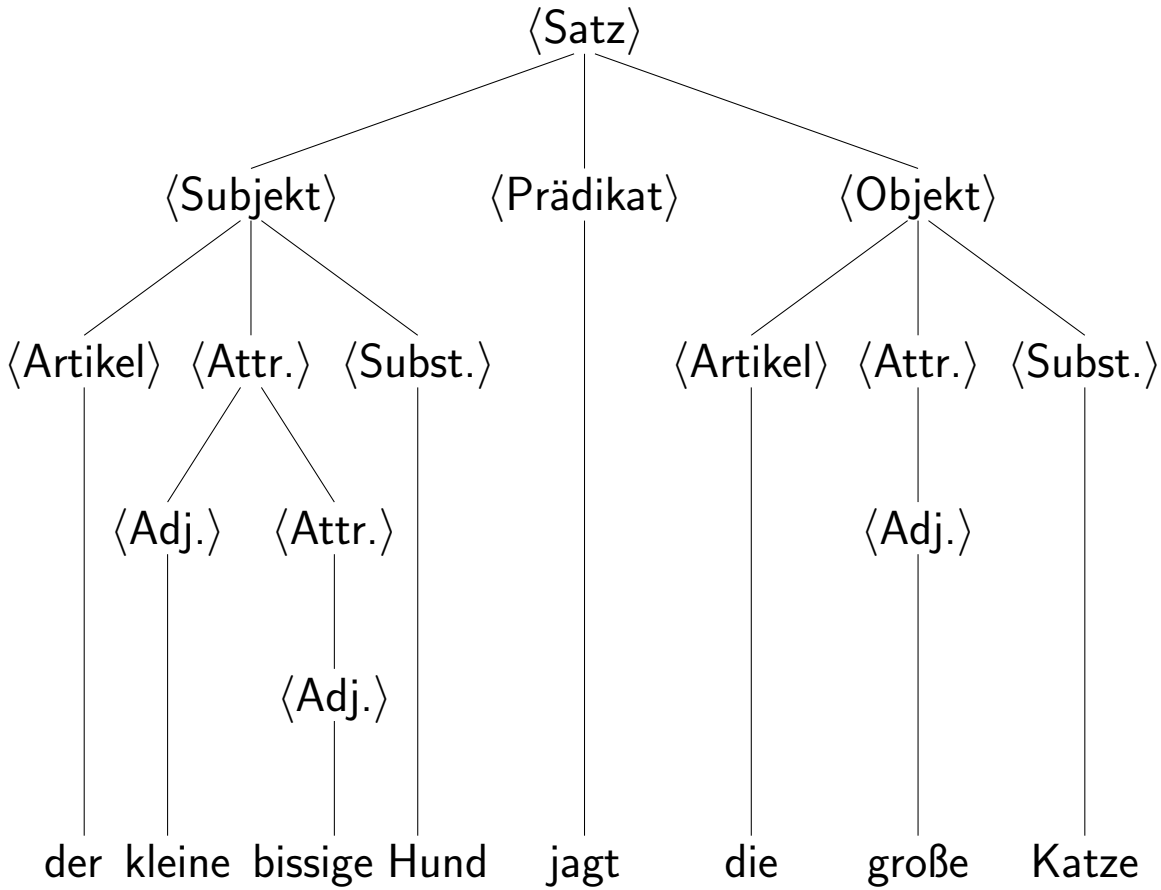
- In spitzen Klammern: Variable, Nicht-Terminale
- Ohne spitze Klammern: Terminale

Grammatiken (Einführung)

Gehört folgender Satz zu der Sprache, die von der Grammatik erzeugt wird?

der kleine bissige Hund jagt die große Katze

Grammatiken (Einführung)



Dieser Baum ist der "Beweis" dafür, dass der Satz in der Sprache vorkommt. Man nennt ihn Syntaxbaum.

Grammatiken (Einführung)

Mit Hilfe dieser (endlichen) Grammatik ist es möglich, unendlich viele Sätze zu erzeugen:

der Hund jagt die kleine kleine kleine . . . Katze

Das heißt, die zu der Grammatik gehörende Sprache (man sagt auch: die von der Grammatik erzeugte Sprache) ist unendlich.

Grammatiken (Definition)

Grammatiken besitzen Regeln der Form

$$\textit{linke Seite} \rightarrow \textit{rechte Seite}$$

Sowohl auf der linken als auch auf auf der rechten Seite können zwei Typen von Symbolen vorkommen:

- Nicht-Terminale (die Variablen, aus denen noch weitere Wortbestandteile abgeleitet werden sollen)
- Terminale (die "eigentlichen" Symbole)

Im vorherigen Beispiel: auf der linken Seite befindet sich immer genau ein Nicht-Terminal (kontextfreie Grammatik).

Es gibt aber allgemeinere Grammatiken. (Es gibt sogar Grammatiken, die auf Bäumen und Graphen statt auf Wörtern arbeiten. Diese werden in der Vorlesung jedoch nicht behandelt.)

Grammatiken (Definition)

Definition (Grammatik)

Eine Grammatik G ist ein 4-Tupel $G = (V, \Sigma, P, S)$, das folgende Bedingungen erfüllt:

- V ist eine endliche Menge von Nicht-Terminalen bzw. Variablen
- Σ ist das endliche Alphabet bzw. die Menge der Terminal(symbol)e. (Es muss gelten: $V \cap \Sigma = \emptyset$, d.h., kein Zeichen ist gleichzeitig Terminal und Nicht-Terminal.)
- P ist eine endliche Menge von Regeln bzw. Produktionen mit $P \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$.
- $S \in V$ ist die Startvariable bzw. das Axiom.

Grammatiken (Definition)

Wie sehen Produktionen aus?

Eine Produktion aus P ist ein Paar (ℓ, r) von Wörtern über $V \cup \Sigma$, das zumeist $\ell \to r$ geschrieben wird. Dabei gilt:

- Sowohl ℓ als auch r bestehen aus Variablen und Terminalsymbolen.
- ℓ darf nicht leer sein. (Eine Regel muss immer zumindest ein Zeichen ersetzen.)

Konventionen:

- Variablen (Elemente aus V) werden mit Großbuchstaben bezeichnet: A, B, C, ..., S, T, ...
- Terminalsymbole (Elemente aus Σ) werden mit Kleinbuchstaben dargestellt: a, b, c, ...

Grammatiken (Beispiel)

Beispiel-Grammatik

$G = (V, \Sigma, P, S)$ mit

- $V = \{S, B, C\}$
- $\Sigma = \{a, b, c\}$
- $P = \{S \to aSBC, S \to aBC, CB \to BC, aB \to ab,$
  $bB \to bb, bC \to bc, cC \to cc\}$

Grammatiken (Ableitungen)

Wie werden die Produktionen eingesetzt, um Wörter aus der Startvariable S zu erzeugen?

Idee: Wenn die Grammatik eine Produktion $\ell \to r$ enthält, dürfen wir ℓ durch r ersetzen.

Beispiel:
Produktion: $CB \to BC$
Ableitungsschritt: $\underbrace{aab}_{x}\underbrace{CB}_{\ell}\underbrace{Bcca}_{y} \Rightarrow \underbrace{aab}_{x}\underbrace{BC}_{r}\underbrace{Bcca}_{y}.$

Grammatiken (Ableitungen)

Definition (Ableitung)

Sei $G = (V, \Sigma, P, S)$ eine Grammatik und seien $u, v \in (V \cup \Sigma)^*$.
Es gilt:

$$u \Rightarrow_G v \ (u \text{ geht unter } G \text{ unmittelbar über in } v),$$

falls u, v folgende Form haben:

$$u = x\ell y \qquad v = xry,$$

wobei $x, y \in (V \cup \Sigma)^*$ und $\ell \to r$ eine Regel in P ist.

Grammatiken (Ableitungen)

Konventionen:

- Wörter aus $(V \cup \Sigma)^*$ werden mit Kleinbuchstaben (aus der hinteren Hälfte des Alphabets) bezeichnet: u, v, w, x, y, z, ...
- Die Konkatenation zweier Wörter u, v wird mit uv bezeichnet. Es gilt $v\varepsilon = \varepsilon v = v$, d.h., das leere Wort ε ist das neutrale Element der Konkatenation.
- Statt $u \Rightarrow_G v$ schreibt man auch $u \Rightarrow v$, wenn klar ist, um welche Grammatik es sich handelt.

Grammatiken (Ableitungen)

Ableitung

Eine Folge von Wörtern $w_0, w_1, w_2, \ldots, w_n \in (V \cup \Sigma)^*$ mit $w_0 = S$ und

$$w_0 \Rightarrow_G w_1 \Rightarrow_G w_2 \Rightarrow_G \cdots \Rightarrow_G w_n$$

heißt Ableitung von w_n (aus S). Dabei darf w_n sowohl Terminale als auch Variablen enthalten und heißt Satzform.

Man schreibt in diesem Fall auch $w_0 \Rightarrow_G^* w_n$, wobei $\Rightarrow_G^*$ die reflexive und transitive Hülle von $\Rightarrow_G$ ist.

Grammatiken und Sprachen

Die von einer Grammatik erzeugte Sprache

Die von einer Grammatik $G = (V, \Sigma, S, P)$ erzeugte Sprache ist

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}.$$

In anderen Worten:

- Die von G erzeugte Sprache besteht genau aus den Satzformen, die nur Terminalsymbole enthalten.
- *Oder:* genau die Wörter, die in mehreren Schritten aus S abgeleitet werden und nur aus Terminalen bestehen, gehören zu $L(G)$.

Grammatiken und Sprachen

Die vorherige Beispielgrammatik G erzeugt die Sprache

$$L(G) = \{a^n b^n c^n \mid n \geq 1\}.$$

Dabei ist $a^n = \underbrace{a \dots a}_{n\text{-mal}}$.

Die Behauptung, dass G wirklich diese Sprache erzeugt, ist nicht einfach nachzuweisen.

Grammatiken und Sprachen

Bemerkung: Ableiten ist kein deterministischer, sondern ein nichtdeterministischer Prozess. Für ein $u \in (V \cup \Sigma)^*$ kann es entweder gar kein, ein oder mehrere v geben mit $u \Rightarrow_G v$.

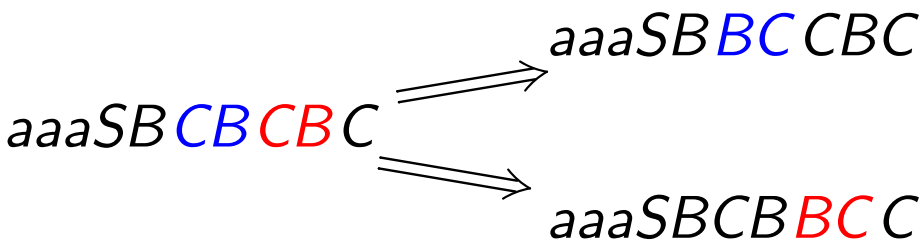
In anderen Worten: $\Rightarrow_G$ ist keine Funktion.

Dieser Nichtdeterminismus kann durch zwei verschiedene Effekte verursacht werden ...

Grammatiken und Sprachen

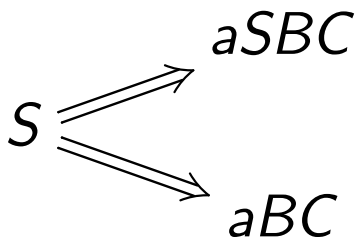
- Eine Regel ist an zwei verschiedenen Stellen anwendbar.

  Beispiel-Grammatik:



- Zwei verschiedene Regeln sind anwendbar (entweder an der gleichen Stelle – wie unten abgebildet – oder an verschiedenen Stellen):

  Beispiel-Grammatik:



Grammatiken und Sprachen

Weitere Bemerkungen:

- Es kann beliebig lange Pfade geben, die nie zu einem Wort aus Terminalsymbolen führen:

$$S \Rightarrow aSBC \Rightarrow aaSBCBC \Rightarrow aaaSBCBCBC \Rightarrow \ldots$$

- Manchmal können Pfade in einer Sackgasse enden, d.h., obwohl noch Variablen in einer Satzform vorkommen, ist keine Regel mehr anwendbar.

$$S \Rightarrow aSBC \Rightarrow aaBCBC \Rightarrow aabCBC \Rightarrow aabcBC \not\Rightarrow$$

Grammatiken und Sprachen

Wir werden oft die folgende kürzere Schreibweise benutzen (die sogenannte Backus-Naur-Form).

Wenn es Regeln

$$u \longrightarrow w_1$$
$$\vdots$$
$$u \longrightarrow w_n$$

gibt, schreiben wir auch

$$u \longrightarrow w_1 \mid \cdots \mid w_n$$

Think-Pair-Share: Grammatiken und Sprachen

Betrachten Sie die folgende Grammatik:

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

mit den folgenden Regeln:

$$S \to aA \mid bB$$
$$A \to aA \mid bA \mid a$$
$$B \to aB \mid bB \mid b$$

Geben Sie ein Wort minimaler Länge an, das von der Grammatik erzeugt wird und überlegen Sie sich, welche Sprache von der Grammatik erzeugt wird. Überlegen Sie zunächst zwei Minuten in Einzelarbeit eine Lösung. Anschließend tauschen Sie sich für weitere zwei Minuten mit ihrem Sitznachbarn aus. Schlussendlich besprechen wir die Lösung im Plenum.

Grammatiken und Sprachen

$\Sigma = \{$  ,  ,  ,  ,  ,  ,  $\}$

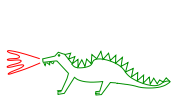
Die Tür-Regel

Durch eine Tür kann man nur gehen, wenn man zuvor einen Schlüssel gefunden hat. (Dieser Schlüssel darf aber dann beliebig oft verwendet werden.)

$G_1 = \big(\{K, N, X\}, \Sigma, P_1, N\big)$, wobei P_1 aus den folgende Produktionen besteht:

$$N \to XN \mid$$ $$K \mid \varepsilon$$

$$K \to XK \mid$$  $$K \mid$$ $$K \mid \varepsilon$$

$$X \to$$  $$\mid$$  $$\mid$$  $$\mid$$  $$\mid$$ 

Grammatiken und Sprachen

$\Sigma = \{$  ,  ,  ,  ,  ,  ,  $\}$

Neue Tür-Regel (Level 2)

Die Schlüssel sind magisch und verschwinden sofort, nachdem eine Tür mit ihnen geöffnet wurde. Sobald man eine Tür durchschritten hat, schließt sie sich sofort wieder.

$G_2 = (\{S, X\}, \Sigma, P_2, S)$, wobei P_2 aus den folgende Produktionen besteht:

$S \to XS \mid$ S  $\mid$ $S \mid SS \mid \varepsilon$

$X \to$  $\mid$  $\mid$  $\mid$  $\mid$ 

Chomsky-Hierarchie

Wir klassifizieren nun Grammatiken nach der Form ihrer Regeln:

Typ 0 – Chomsky-0

Jede Grammatik ist vom Typ 0. (Keine Einschränkung der Regeln.)

Typ 1 – Chomsky-1

Für alle Regeln $\ell \to r$ gilt: $|\ell| \leq |r|$. (Man sagt auch, die Grammatik ist monoton oder kontextsensitiv.)

Typ 2 – Chomsky-2

Eine Typ-1-Grammatik ist vom Typ 2 oder kontextfrei, wenn für alle Regeln $\ell \to r$ gilt, dass $\ell \in V$, d.h., ℓ ist eine einzelne Variable.

D.h., es sind nur Regeln der Form $A \to r$ mit $A \in V$, $r \in (V \cup \Sigma)^*$ erlaubt.

Chomsky-Hierarchie

Typ 3 – Chomsky-3

Eine Typ-2-Grammatik ist vom Typ 3 oder regulär, falls zusätzlich gilt: $r \in \Sigma \cup \Sigma V$, d.h., die rechten Seiten von Regeln sind entweder einzelne Terminale oder ein Terminal gefolgt von einer Variablen.

D.h., es sind nur Regeln der Form $A \to aB$ und $A \to a$ mit $A, B \in V$, $a \in \Sigma$ erlaubt.

Typ-i-Sprache

Eine Sprache $L \subseteq \Sigma^*$ heißt vom Typ i ($i \in \{0, 1, 2, 3\}$), falls es eine Typ-i-Grammatik G gibt mit $L(G) = L$ (d.h., L wird von G erzeugt.)

Solche Sprachen nennt man dann auch semi-entscheidbar bzw. rekursiv aufzählbar (Typ 0), kontextsensitiv (Typ 1), kontextfrei (Typ 2) oder regulär (Typ 3).

Chomsky-Hierarchie

ε-Sonderregelung (Teil 1)

Bei Typ-1-Grammatiken (und damit auch bei regulären und kontextfreien Grammatiken) ist die Regel $S \to \varepsilon$ zunächst nicht zugelassen, wegen $|S| = 1 \not\leq 0 = |\varepsilon|$. Das bedeutet aber: das leere Wort kann nicht abgeleitet werden!

Wir modifizieren daher die Grammatik-Definition für Typ-1-Grammatiken leicht und erlauben $S \to \varepsilon$, falls S das Startsymbol ist und auf keiner rechten Seite vorkommt. Diese Bedingung heißt ε-Sonderregelung.

Chomsky-Hierarchie

ε-Sonderregelung (Teil 2)

Bei *kontextfreien* und *regulären* Grammatiken (Typ 2, Typ 3) ändert sich die Ausdrucksmächtigkeit nicht, wenn man beliebige Produktionen der Form $A \to \varepsilon$ erlaubt:

Durch geeignete Umformungen kann man eine Grammatik die bis auf ε-Ableitungen regulär (kontextfrei) ist, in eine reguläre (kontextfreie) Grammatik transformieren, die die ε-Sonderregel erfüllen. Eine solche Konstruktion existiert im Allgemeinen nicht für alle Typ-1-Grammatiken.

Chomsky-Hierarchie

Bemerkungen:

- Woher kommt der Begriff "kontextsensitiv"?

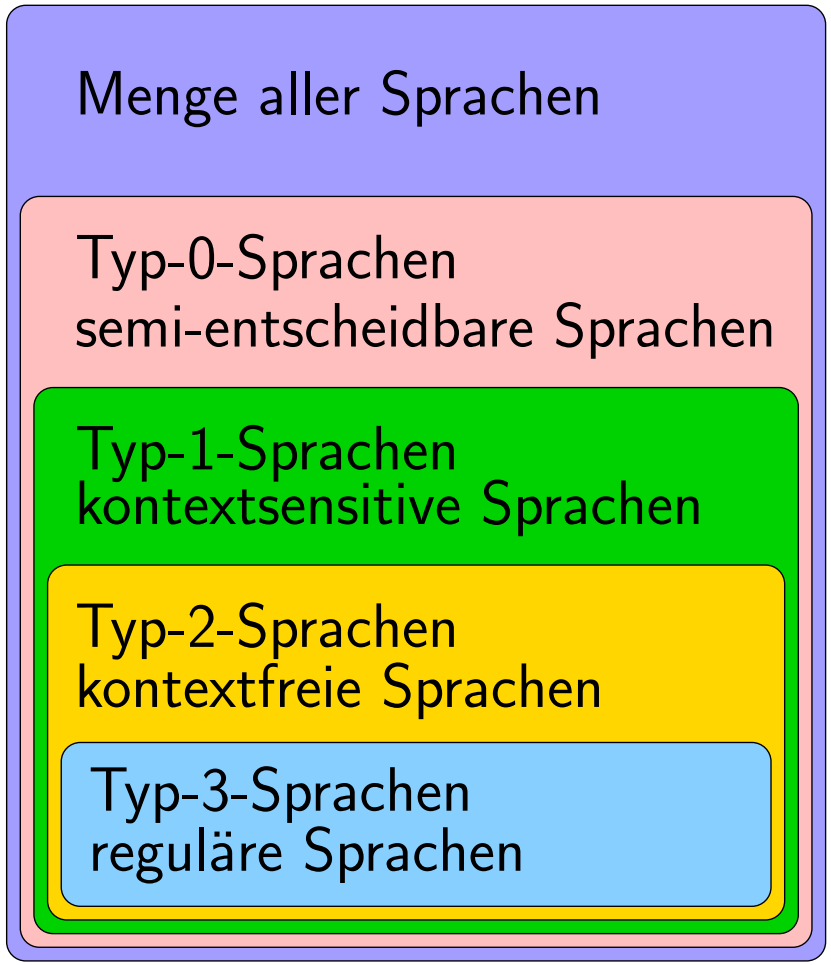
  Bei kontextfreien Sprachen gibt es Regeln der Form $A \to x$, wobei $x \in (\Sigma \cup V)^*$. Das bedeutet: A kann – unabhängig vom Kontext – durch x ersetzt werden.

  Bei den mächtigeren kontextsensitiven Sprachen sind dagegen Regeln der Form $uAv \to uxv$ möglich, mit der Bedeutung: A kann nur in bestimmten Kontexten durch x ersetzt werden.

Chomsky-Hierarchie

Jede Typ-i-Grammatik ist eine Typ-$(i-1)$-Grammatik (für $i \in \{1, 2, 3\}$) $\rightsquigarrow$ die entsprechenden Mengen von Sprachen sind ineinander enthalten.

Außerdem: die Inklusionen sind echt, d.h., es gibt für jedes i eine Typ-$(i-1)$-Sprache, die keine Typ-i-Sprache ist. (Zum Beispiel eine kontextfreie Sprache, die nicht regulär ist.) Das werden wir später zeigen.



Chomsky-Hierarchie

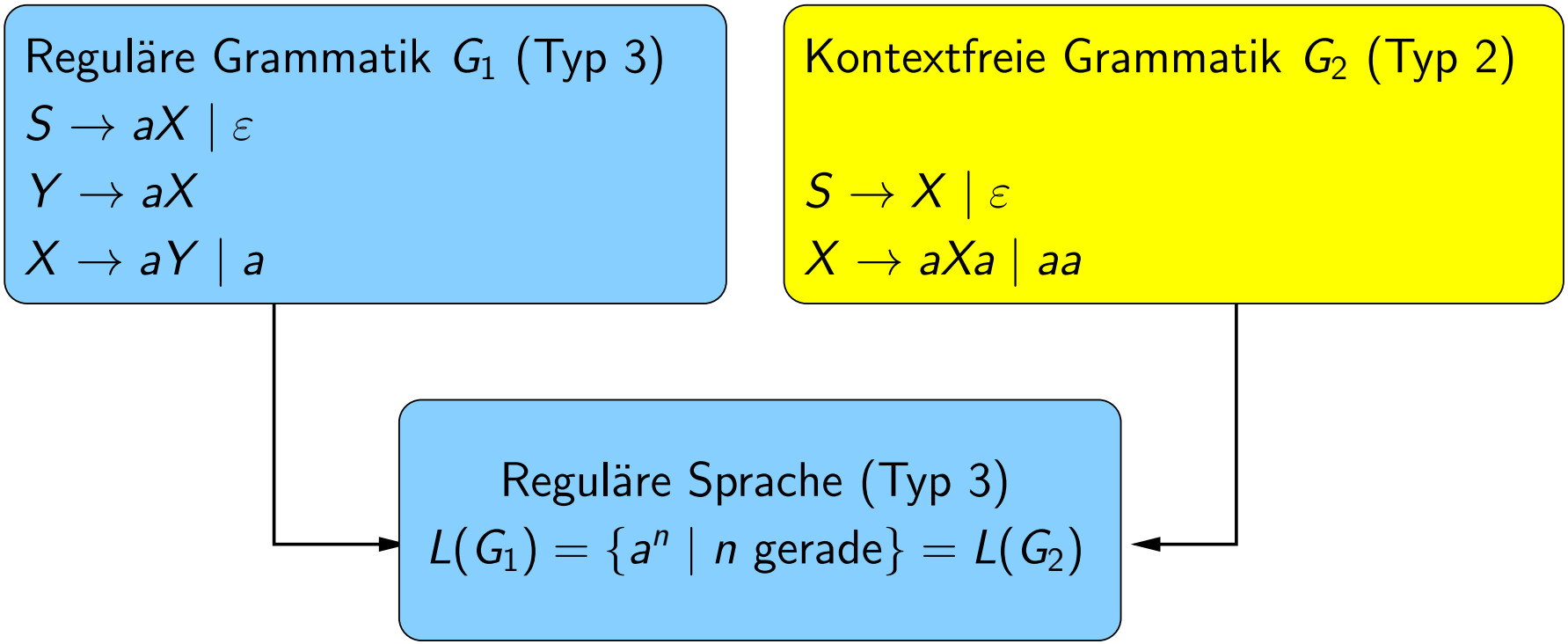
Bemerkungen:

- Für eine Sprache der Chomsky-Hierarchie gibt es immer mehrere Grammatiken, die diese Sprache erzeugen.
- Eine Sprache, die durch eine Grammatik vom Typ i erzeugt wird, hat Typ k für alle $k \leq i$. Sie kann in manchen Fällen aber auch Typ j mit $j > i$ haben.

  Beispielsweise erzeugt die Grammatik G mit den Produktionen $S \to X \mid \varepsilon$, $X \to aXa \mid aa$ die Sprache $L(G) = \{a^n \mid n \in \mathbb{N}_0, n \text{ gerade}\}$.

  Die Grammatik G ist vom Typ 2, aber nicht vom Typ 3. Die Sprache $L(G)$ hat sowohl Typ 2 als auch Typ 3.

Chomsky-Hierarchie



Fragestellungen zu dieser Vorlesungseinheit

Zum Ende einer jeden Vorlesungseinheit betrachten wir im Regelfall drei Fragestellungen, die mithilfe der in dieser Einheit besprochenen Inhalte beantwortet werden sollen. In der darauffolgenden Einheit können zu Beginn mögliche Antworten gesammelt werden.

Fragen zur zweiten Vorlesungseinheit

- Was sind Grammatiken und Sprachen und wie hängen diese beiden Begriffe zusammen?
- Wie sind die vier Hierarchie-Ebenen der Chomsky-Hierarchie definiert; wann ist eine Grammatik und wann ist eine Sprache vom Chomsky Typ i, $i \in \{0, 1, 2, 3\}$?
- Was bedeutet es, dass ein Wort von einer Grammatik erzeugt wird und wie wird in dieser Hinsicht mit Nichtdeterminismus der Ableitung umgegangen?

Wortproblem

Wortproblem

Gegeben eine Grammatik G (von beliebigem Typ) und ein Wort $w \in \Sigma^*$. Entscheide, ob $w \in L(G)$.

Entscheidbarkeit des Wortproblems (Satz)

Das Wortproblem ist entscheidbar für Typ-1-Sprachen (und damit auch für reguläre und kontextfreie Sprachen). Das heißt: es gibt ein Verfahren, das entscheidet, ob $w \in L(G)$ gilt.

Wortproblem für Typ-1-Sprachen

Algorithmus zum Lösen des Wortproblems für Typ-1-Sprachen:
gibt "true" aus genau dann, wenn $w \in L(G)$.

input (G, w)
$T := \{S\}$
repeat
    $T' := T$
    $T := T' \cup \{u \mid |u| \leq |w| \text{ und } u' \Rightarrow u, \text{ für ein } u' \in T'\}$
until $(w \in T)$ or $(T = T')$
return $(w \in T)$

Wortproblem für Typ-1-Sprachen

Wir können die Korrektheit des Algorithmus wie folgt einsehen:

- Da die Grammatik vom Typ 1 ist, können bei einer Ableitung eines Wortes (der Länge größer 1) nur Wörter entstehen, die länger oder gleich lang sind.
- Also müssen Wörter, die die Länge des gesuchten Wortes w übersteigen, nicht weiter exploriert werden – sie können auf keinen Fall mehr zu w abgeleitet werden.
- Daher gilt: Wann immer der Algorithmus „false“ ausgibt, ist $w \notin L(G)$. Offensichtlich gilt auch $w \in L(G)$ wann immer der Algorithmus „true“ ausgibt, da in diesem Fall eine Ableitungsfolge $S \Rightarrow w$ gefunden wurde.

Es bleibt zu zeigen, dass der Algorithmus auch tatsächlich terminiert, da aber nur endlich viele Wörter u mit $|u| \leq |w|$ existieren (sowohl Σ als auch V sind endlich), lässt sich das leicht einsehen.

Wortproblem für Typ-1-Sprachen

Beispiel:

- Grammatik G: $S \to aX \mid bX$, $X \to cS \mid d$
- Wort $w = acbdc$
- Entstehende Folge von Mengen von Satzformen:
  1. $T = \{S\}$
  2. $T = \{S, aX, bX\}$
  3. $T = \{S, aX, bX, acS, ad, bcS, bd\}$
  4. $T = \{S, aX, bX, acS, ad, bcS, bd, acaX, acbX, bcaX, bcbX\}$
  5. $T = \{S, aX, bX, acS, ad, bcS, bd, acaX, acbX, bcaX, bcbX,$
     $acacS, acad, acbcS, acbd, bcacS, bcad, bcbcS, bcbd\}$

  Nach dem fünften Schritt bricht der Algorithmus ab, da nur noch Wörter entstehen, die länger als w sind.
  Es gilt: $w \notin T$, daraus folgt $w \notin L(G)$.

Syntaxbäume und Eindeutigkeit

Wir beschränken uns im Folgenden auf kontextfreie Grammatiken.

Wir betrachten folgende (eindeutige) Beispiel-Grammatik zur Erzeugung von korrekt geklammerten arithmetischen Ausdrücken:

$$G = (\{E, T, F\}, \{(,), a, +, *\}, P, E)$$

mit folgender Produktionenmenge P (in abkürzender Backus-Naur-Form):

$$\begin{aligned} E &\rightarrow T \mid E + T \\ T &\rightarrow F \mid T * F \\ F &\rightarrow a \mid (E) \end{aligned}$$

Think-Pair-Share: Eindeutigkeit

$$G = (\{E, T, F\}, \{(,), a, +, *\}, P, E)$$

$$\begin{aligned} E &\to T \mid E + T \\ T &\to F \mid T * F \\ F &\to a \mid (E) \end{aligned}$$

Zeigen Sie, dass der Ausdruck $a * (a + a)$ mit G ableitbar ist.
Erarbeiten Sie zunächst drei Minuten in Einzelarbeit eine Lösung. Anschließend tauschen Sie sich für weitere drei Minuten mit ihrem Sitznachbarn aus. Schlussendlich besprechen wir die Lösung im Plenum.

Syntaxbäume und Eindeutigkeit

Für die meisten Wörter der von G erzeugten Sprache gibt es mehrere mögliche Ableitungen:

$$E \Rightarrow T \Rightarrow T * F \Rightarrow F * F \rightarrow a * F \Rightarrow a * (E)$$
$$\Rightarrow a * (E + T) \Rightarrow a * (T + T) \Rightarrow a * (F + T)$$
$$\Rightarrow a * (a + T) \Rightarrow a * (a + F) \Rightarrow a * (a + a)$$

$$E \Rightarrow T \Rightarrow T * F \Rightarrow T * (E) \rightarrow T * (E + T)$$
$$\Rightarrow T * (E + F) \Rightarrow T * (E + a) \Rightarrow T * (T + a)$$
$$\Rightarrow T * (F + a) \Rightarrow T * (a + a) \Rightarrow F * (a + a) \Rightarrow a * (a + a)$$

Die erste Ableitung ist eine sogenannte Linksableitung (immer so weit links wie möglich ableiten), die zweite eine Rechtsableitung (so weit rechts wie möglich ableiten).

Syntaxbäume und Eindeutigkeit

Syntaxbaum aufbauen

Wir bilden nun aus beiden Ableitungen den Syntaxbaum, indem wir

- Die Wurzel des Baums mit der Startvariable der Grammatik beschriften.
- Bei jeder Regelanwendung der Form $A \to z$ zu A $|z|$ Kinder hinzufügen, die mit den Zeichen von z beschriftet sind.

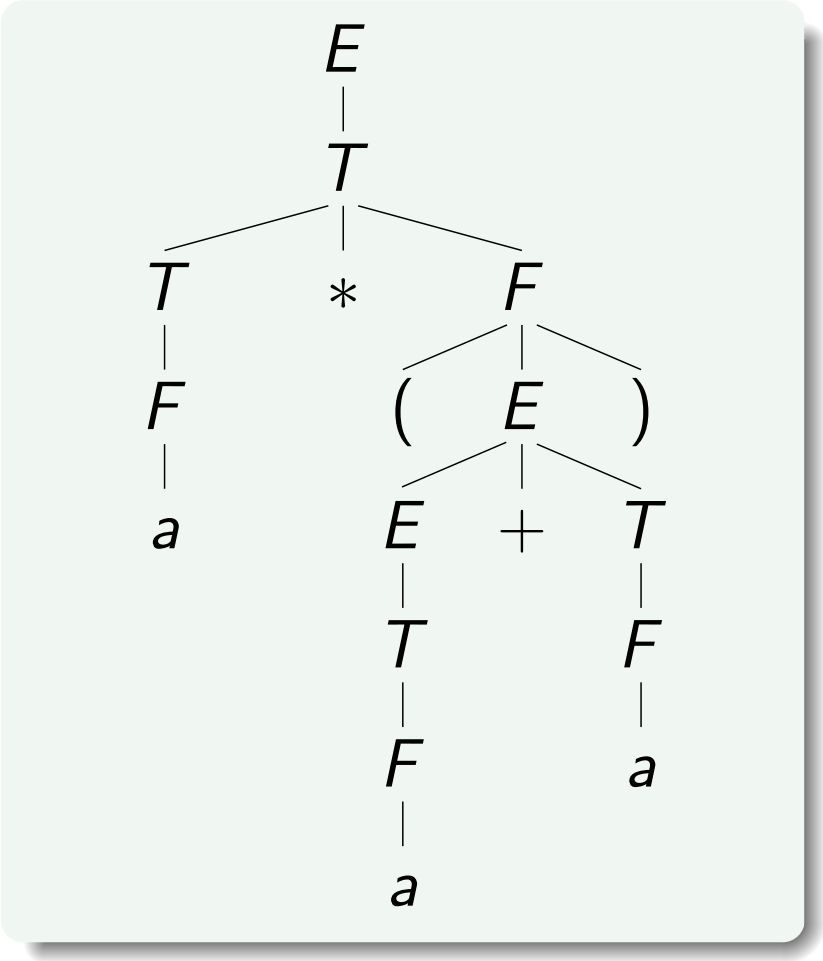
Syntaxbäume lassen sich für alle Ableitungen von kontextfreien Grammatiken aufbauen.

Syntaxbäume und Eindeutigkeit

Dabei erhalten wir in beiden Fällen den gleichen Syntaxbaum.

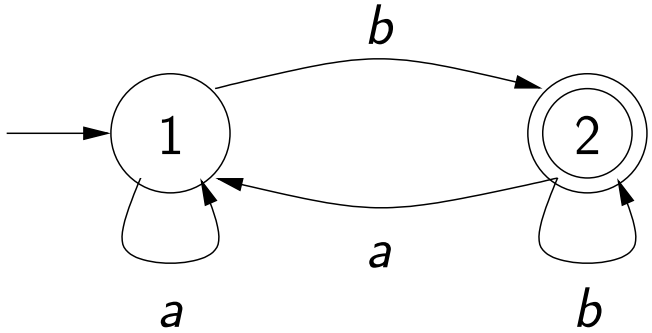
Man sagt, eine Grammatik ist eindeutig, wenn es für jedes Wort in der erzeugten Sprache genau einen Syntaxbaum gibt
$\iff$ es gibt für jedes Wort genau eine Linksableitung
$\iff$ es gibt für jedes Wort genau eine Rechtsableitung.

Ansonsten heißt die Grammatik mehrdeutig.



Endliche Automaten

In diesem Abschnitt beschäftigen wir uns mit regulären Sprachen, aber zunächst unter einem anderen Blickwinkel. Statt Typ-3-Grammatiken betrachten wir zustandsbasierte Automatenmodelle, die man auch als "Spracherzeuger" bzw. "Sprachakzeptierer" betrachten kann.



Deterministische endliche Automaten

Deterministischer endlicher Automat (Definition)

Ein (deterministischer) endlicher Automat M ist ein 5-Tupel $M = (Z, \Sigma, \delta, z_0, E)$, wobei

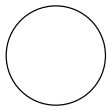
- Z die Menge der Zustände,
- Σ das Eingabealphabet (mit $Z \cap \Sigma = \emptyset$),
- $z_0 \in Z$ der Startzustand,
- $E \subseteq Z$ die Menge der Endzustände *und*
- $\delta\colon Z \times \Sigma \to Z$ die Überführungsfunktion (oder Übergangsfunktion) ist.

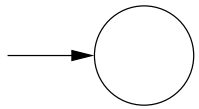
Z, Σ müssen endliche Mengen sein.

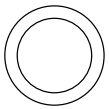
Abkürzung: DFA (deterministic finite automaton)

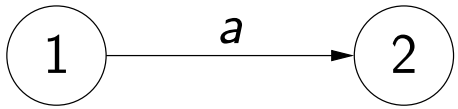
Deterministische endliche Automaten

Graphische Notation:

Zustand: 

Startzustand: 

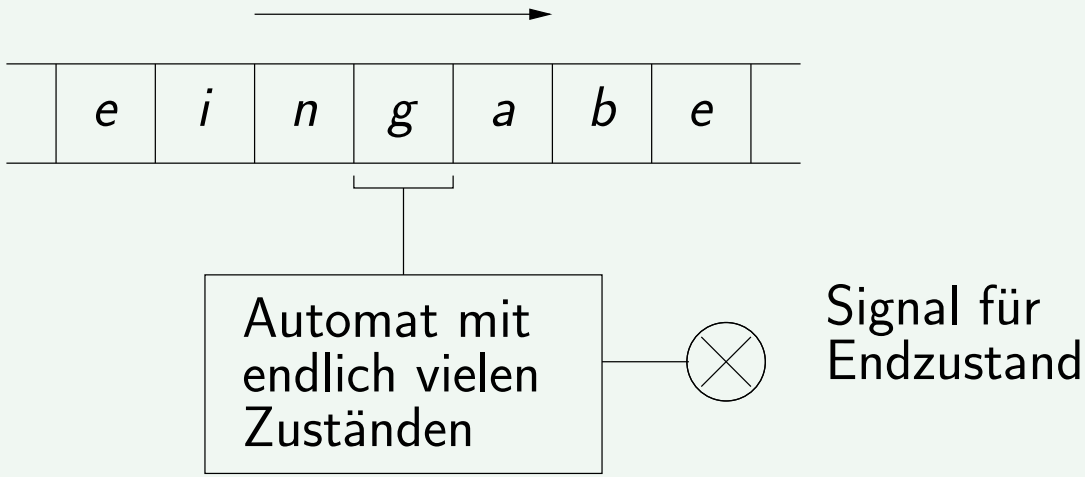
Endzustand: 

Übergang $\delta(1, a) = 2$: 

Deterministische endliche Automaten

Woher kommt der Name “endlicher Automat”?

Vorstellung von einer Maschine, die sich in endlich vielen Zuständen befinden kann, die eine Eingabe (von links nach rechts) liest und signalisiert, sobald die Eingabe akzeptiert ist.



Deterministische endliche Automaten

Analogie zum Fahrkartenautomat: ein Fahrkartenautomat kann sich in folgenden Zuständen befinden:

- Keine Eingabe
- Fahrtziel ausgewählt
- Geld eingegeben
- Fahrkarte wurde ausgegeben

(Das ist eine vereinfachte Darstellung, da ein Fahrkartenautomat auch mitzählen muss, wieviel Geld bereits eingeworfen wurde. Dafür würde man jedoch (idealisiert, unter Missachtung der maximalen Kapazität eines Fahrkartenautomaten) unendlich viele Zustände benötigen.)

Deterministische endliche Automaten

Die bisherige Übergangsfunktion δ liest nur ein Zeichen auf einmal ein. Wir verallgemeinern sie daher zu einer Übergangsfunktion $\hat{\delta}$, die die Übergänge für ganze Wörter ermittelt.

Mehr-Schritt-Übergänge

Zu einem gegebenen DFA $M = (Z, \Sigma, \delta, z_0, E)$ definieren wir eine Funktion $\hat{\delta}\colon Z \times \Sigma^* \to Z$ induktiv wie folgt:

$$\begin{aligned} \hat{\delta}(z, \varepsilon) &= z \\ \hat{\delta}(z, ax) &= \hat{\delta}(\delta(z, a), x) \end{aligned}$$

mit $z \in Z$, $x \in \Sigma^*$ und $a \in \Sigma$.

Deterministische endliche Automaten

Akzeptierte Sprache

Die einem DFA M akzeptierte Sprache ist

$$T(M) = \{x \in \Sigma^* \mid \hat{\delta}(z_0, x) \in E\}.$$

In anderen Worten:

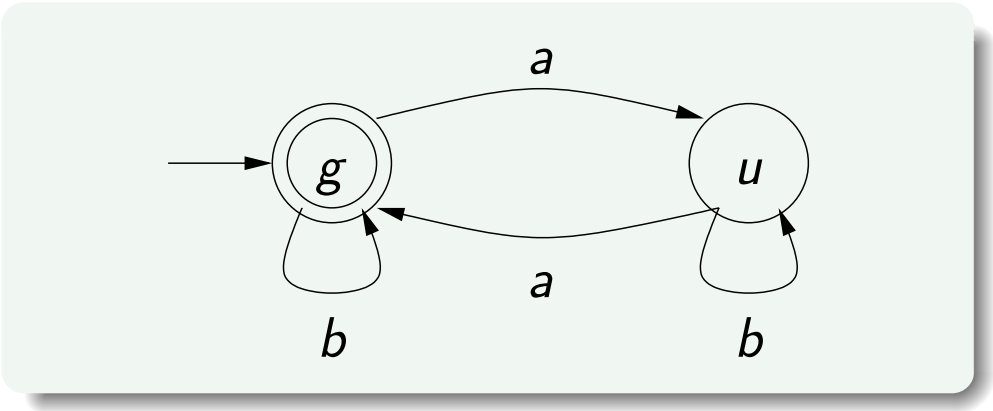
Die Sprache kann man dadurch erhalten, indem man allen Pfaden vom Anfangszustand zu einem Endzustand folgt und dabei alle Zeichen auf den Übergängen aufsammelt.

Deterministische endliche Automaten

Beispiel 1: Wir suchen einen endlichen Automaten, der folgende Sprache L akzeptiert:

$$L = \{w \in \{a, b\}^* \mid \#_a(w) \text{ gerade}\}.$$

Dabei ist $\#_a(w)$ die Anzahl der a's in w.

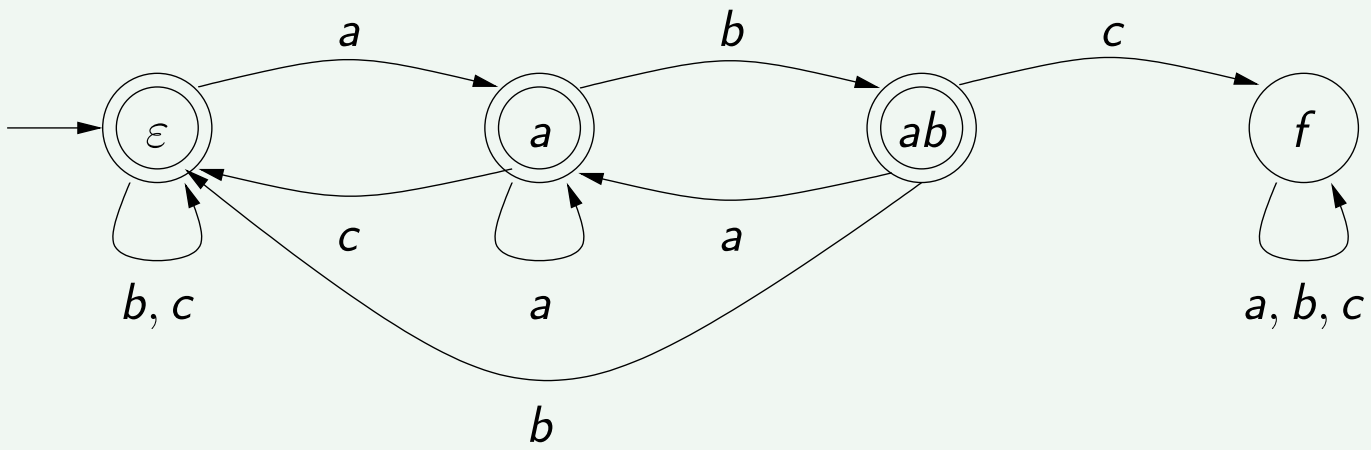


Bedeutung der Zustände:
g – gerade Anzahl a's; u – ungerade Anzahl a's

Deterministische endliche Automaten

Beispiel 2: Wir suchen einen endlichen Automaten, der folgende Sprache L akzeptiert:

$$L = \{w \in \{a, b, c\}^* \mid \text{das Teilwort } abc \text{ kommt in } w \text{ nicht vor}\}.$$



Bedeutung der Zustände:

ε – kein Präfix von abc gelesen; a – letztes gelesenes Zeichen war ein a; ab – zuletzt ab gelesen; f – abc kam im bereits gelesenen Wort vor (Fangzustand, Fehlerzustand)

Think-Pair-Share: DFA

Geben Sie einen DFA für die Sprache

$$L = \{w \in \{a, b, c\} \mid \text{auf jedes } a \text{ folgt ein } b\}$$

an.
Erarbeiten Sie zunächst drei Minuten in Einzelarbeit eine Lösung. Anschließend tauschen Sie sich für weitere drei Minuten mit ihrem Sitznachbarn aus. Schlussendlich besprechen wir die Lösung im Plenum.

Fragestellungen zu dieser Vorlesungseinheit

Zum Ende einer jeden Vorlesungseinheit betrachten wir im Regelfall drei Fragestellungen, die mithilfe der in dieser Einheit besprochenen Inhalte beantwortet werden sollen. In der darauffolgenden Einheit können zu Beginn mögliche Antworten gesammelt werden.

Fragen zur dritten Vorlesungseinheit

- Wie kann man für eine beliebige Typ-1 Grammatik G entscheiden, ob ein Wort w von G erzeugt wird, also ob $w \in L(G)$ gilt?
- Was sind Syntaxbäume für kontextfreie Grammatiken und wie hängen diese mit dem Begriff der Eindeutigkeit (einer Grammatik) zusammen?
- Was ist ein deterministischer endlicher Automat?

Deterministische endliche Automaten

DFAs $\rightarrow$ Reguläre Sprachen (Satz)

Jede von einem endlichen Automaten akzeptierte Sprache ist regulär.

Beweisidee: Ein endlicher Automat $M = (Z, \Sigma, \delta, z_0, E)$ wird in eine Grammatik $G = (V, \Sigma, P, S)$ umgewandelt, wobei $V = Z$, $S = z_0$ und P folgende Produktionen enthält:

$$\text{falls} \quad \delta(z_1, a) = z_2, \quad \text{dann gilt} \quad (z_1 \rightarrow az_2) \in P$$

Falls zusätzlich $z_2 \in E$, dann gilt $(z_1 \rightarrow a) \in P$.

Außerdem gilt $(z_0 \rightarrow \varepsilon) \in P$, falls $z_0 \in E$.

Deterministische endliche Automaten

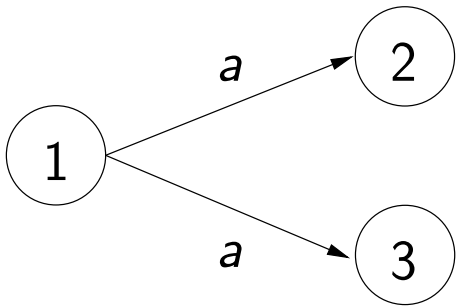
Bemerkungen:

- Bei der Konstruktion kann die Regel $z_0 \to \varepsilon$ hinzugefügt werden und die Variable z_0 gleichzeitig auf einer rechten Seite auftreten, was eigentlich ein Verstoß gegen die ε-Sonderregelung ist. Bei regulären (und auch kontextfreien Grammatiken) kann die Grammatik jedoch immer so umgeformt werden, dass die Bedingungen der ε-Sonderregelung wieder erfüllt sind.
- Es gilt auch die umgekehrte Aussage: jede reguläre Sprache kann von einem endlichen Automaten akzeptiert werden. (Dazu später mehr.)

Nichtdeterministische endliche Automaten

Im Gegensatz zu Grammatiken gibt es bei DFAs keine nichtdeterministischen Effekte. Das heißt, sobald das nächste Zeichen eingelesen wurde, ist klar, welcher Zustand der Folgezustand ist.

Aber: In vielen Fällen ist es natürlicher, wenn man auch nichtdeterministische Übergänge zulässt. Das führt auch oft zu kleineren Automaten.



Nichtdeterministische endliche Automaten

Definition: Nichtdeterministischer endlicher Automat

Ein nichtdeterministischer endlicher Automat M ist ein 5-Tupel $M = (Z, \Sigma, \delta, S, E)$, wobei

- Z die Menge der Zustände,
- Σ das Eingabealphabet (mit $Z \cap \Sigma = \emptyset$),
- $S \subseteq Z$ die Menge der Startzustände,
- $E \subseteq Z$ die Menge der Endzustände *und*
- $\delta\colon Z \times \Sigma \to \mathcal{P}(Z)$ die Überführungsfunktion (oder Übergangsfunktion) ist.

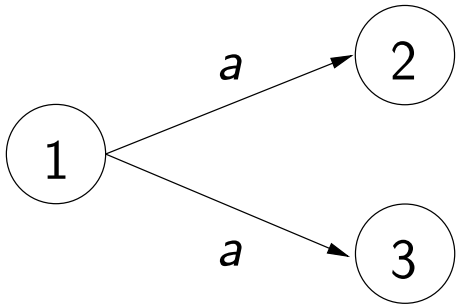
Z, Σ müssen endliche Mengen sein.

Abkürzung: NFA (nondeterministic finite automaton)

Nichtdeterministische endliche Automaten

Dabei ist $\mathcal{P}(Z)$ die Potenzmenge von Z, d.h., die Menge aller Teilmengen von Z. (Diese Menge wird manchmal auch mit 2^Z bezeichnet.)

Beispiel: $\delta(1, a) = \{2, 3\}$



Nichtdeterministische endliche Automaten

Die Übergangsfunktion δ kann wieder zu einer Mehr-Schritt-Übergangsfunktion erweitert werden:

Mehr-Schritt-Übergänge

Zu einem gegebenen NFA $M = (Z, \Sigma, \delta, S, E)$ definieren wir eine Funktion $\hat{\delta}\colon \mathcal{P}(Z) \times \Sigma^* \to \mathcal{P}(Z)$ induktiv wie folgt:

$$\begin{aligned} \hat{\delta}(Z', \varepsilon) &= Z' \\ \hat{\delta}(Z', ax) &= \bigcup_{z \in Z'} \hat{\delta}(\delta(z, a), x) \end{aligned}$$

mit $Z' \subseteq Z$, $x \in \Sigma^*$ und $a \in \Sigma$.

Nichtdeterministische endliche Automaten

Akzeptierte Sprache

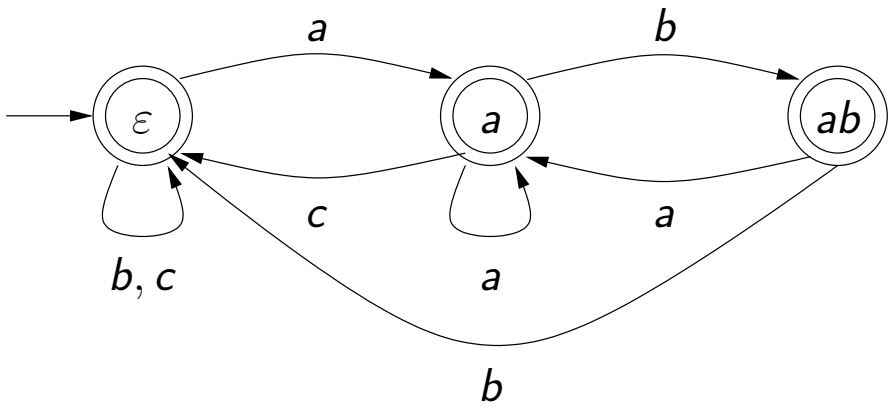
Die einem NFA M akzeptierte Sprache ist

$$T(M) = \{x \in \Sigma^* \mid \hat{\delta}(S, x) \cap E \neq \emptyset\}.$$

In anderen Worten: ein Wort w wird akzeptiert, genau dann wenn es einen Pfad von einem Anfangszustand zu einem Endzustand gibt, dessen Übergänge mit den Zeichen von w markiert sind. (Es könnte auch mehrere solche Pfade geben.)

Nichtdeterministische endliche Automaten

Beispiel 1: bei nicht-deterministischen Automaten darf auch $\delta(z, a) = \emptyset$ für ein $a \in \Sigma$ gelten, das heißt, es muss nicht für jedes Alphabetsymbol immer einen Übergang geben und der sogenannte "Fangzustand" kann weggelassen werden.

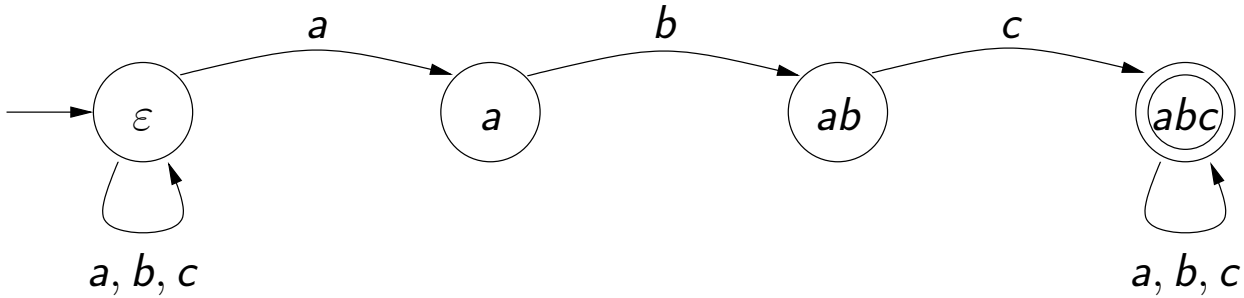


Nichtdeterministische endliche Automaten

Beispiel 2: gesucht ist ein nicht-deterministischer Automat, der die Sprache

$$L = \{w \in \{a, b, c\}^* \mid \text{das Teilwort } abc \text{ kommt in } w \text{ vor}\}$$

akzeptiert.



Dieser Automat entscheidet zu einem bestimmten Zeitpunkt nicht-deterministisch, dass jetzt das Teilwort *abc* beginnt.

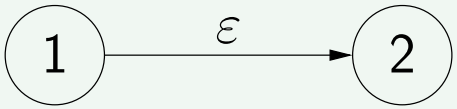
Nichtdeterministische endliche Automaten

Andere Interpretation: jedes Mal, wenn eine nicht-deterministische Verzweigung möglich ist, werden mehrere "Paralleluniversen" erzeugt, in denen verschiedene Kopien der Maschine die verschiedenen möglichen Pfade erkunden. Das Wort wird akzeptiert, wenn es in einem dieser Paralleluniversen akzeptiert wird.

Nichtdeterministische endliche Automaten

ε-Kanten

Es gibt auch nichtdeterministische Automaten mit sogenannten ε-Kanten (spontante Übergänge, bei denen kein Alphabetsymbol eingelesen wird). Diese werden jedoch in der Vorlesung im Allgemeinen nicht benutzt.

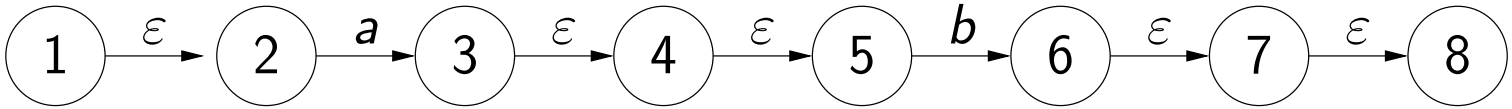


Neue Übergangsfunktion: $\delta\colon Z \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Z)$

Im Beispiel: $\delta(1, \varepsilon) = \{2\}$.

Nichtdeterministische endliche Automaten

Neue Mehr-Schritt-Übergangsfunktion: $\hat{\delta}\colon \mathcal{P}(Z) \times \Sigma^* \to \mathcal{P}(Z)$.
Dabei dürfen zwischen dem Einlesen der Zeichen beliebig viele ε-Übergänge gemacht werden.



$$\hat{\delta}(\{1\}, ab) = \{6, 7, 8\}$$

Think-Pair-Share: ε-Übergänge

Zeigen Sie:

Äquivalenz von NFAs mit und ohne ε-Übergängen

Jeder NFA mit ε-Übergängen kann in einen NFA ohne ε-Übergänge umgewandelt werden, ohne die Anzahl der Zustände zu erhöhen.

Erarbeiten Sie zunächst vier Minuten in Einzelarbeit eine Lösung. Anschließend tauschen Sie sich für weitere vier Minuten mit ihrem Sitznachbarn aus. Schlussendlich besprechen wir die Lösung im Plenum. Es reicht, wenn Sie die notwendige Konstruktion angeben.

Lösungsvorschlag zur Think-Pair-Share-Aufgabe

Beweisidee:

- Für jeden Übergang $\delta(z, a) \ni z'$ so dass es einen ε-Übergang von z' zu einem Zustand z'' gibt, füge z'' zu $\delta(z, a)$ hinzu.
- Wiederhole diesen Schritt, bis sich hierdurch keine Änderung mehr ergibt.
- Für alle Startzustände, die einen ε-Übergang zu einem Zustand z haben, füge z zu der Startzustandsmenge hinzu.
- Entferne alle ε-Übergänge

NFAs, DFAs und reguläre Grammatiken

NFAs $\rightarrow$ DFAs (Satz)

Jede von einem NFA akzeptierbare Sprache ist auch von einem DFA akzeptierbar.

Idee: Wir lassen die verschiedenen "Paralleluniversen" von einem Automaten simulieren. Dieser merkt sich, in welchen Zuständen er sich gerade befindet.

Das heißt, die Zustände dieses Automaten sind Mengen von Zuständen des ursprünglichen Automaten. Man nennt diese Konstruktion daher auch Potenzmengenkonstruktion.

NFAs, DFAs und reguläre Grammatiken

Potenzmengenkonstruktion:

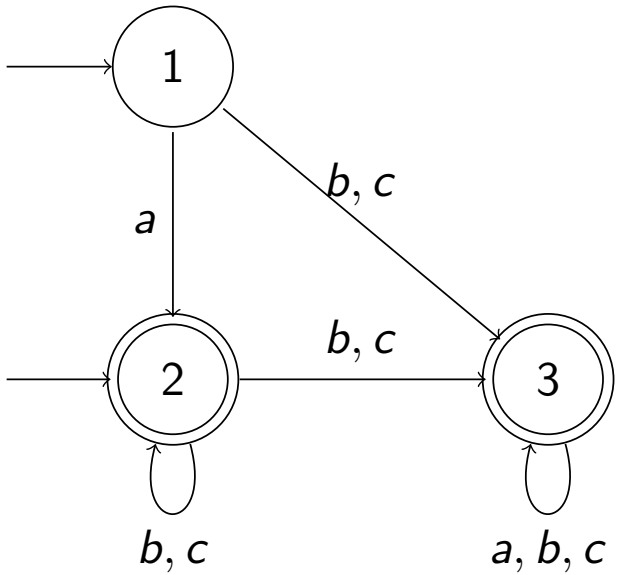
Gegeben sei ein nicht-deterministischer endlicher Automat $M = (Z, \Sigma, \delta, S, E)$. Daraus konstruieren wir einen deterministischen endlichen Automaten $M' = (\mathcal{Z}, \Sigma, \delta', z_0', E')$ mit:

$$\begin{aligned} \mathcal{Z} &= \mathcal{P}(Z) \\ \delta'(Z', a) &= \hat{\delta}(Z', a),\ Z' \subseteq Z \\ z_0' &= S \\ E' &= \{Z' \subseteq Z \mid Z' \cap E \neq \emptyset\} \end{aligned}$$

Dabei entspricht der Zustand $Z' = \emptyset$ einem Fangzustand.

Beispiel zur Potenzmengenkonstruktion

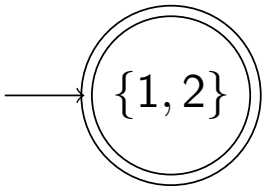
Wir betrachten folgendes Beispiel für einen NFA (welche Sprache akzeptiert der NFA?):



Wir wollen nun mithilfe der Potenzmengenkonstruktion diesen Automaten in einen DFA umwandeln.
Hierzu werden wir den Automaten ausgehend vom Startzustand konstruieren, auf diese Weise können wir nicht erreichbare Zustände auslassen.

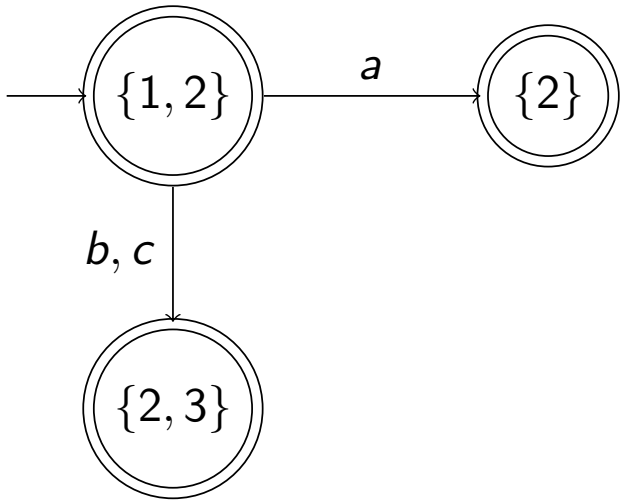
Der Startzustand

Der Startzustand ist die Menge aller Startzustände des NFA, in diesem Fall $\{1, 2\}$. Der Zustand ist ein Endzustand, weil 2 ein Endzustand ist.



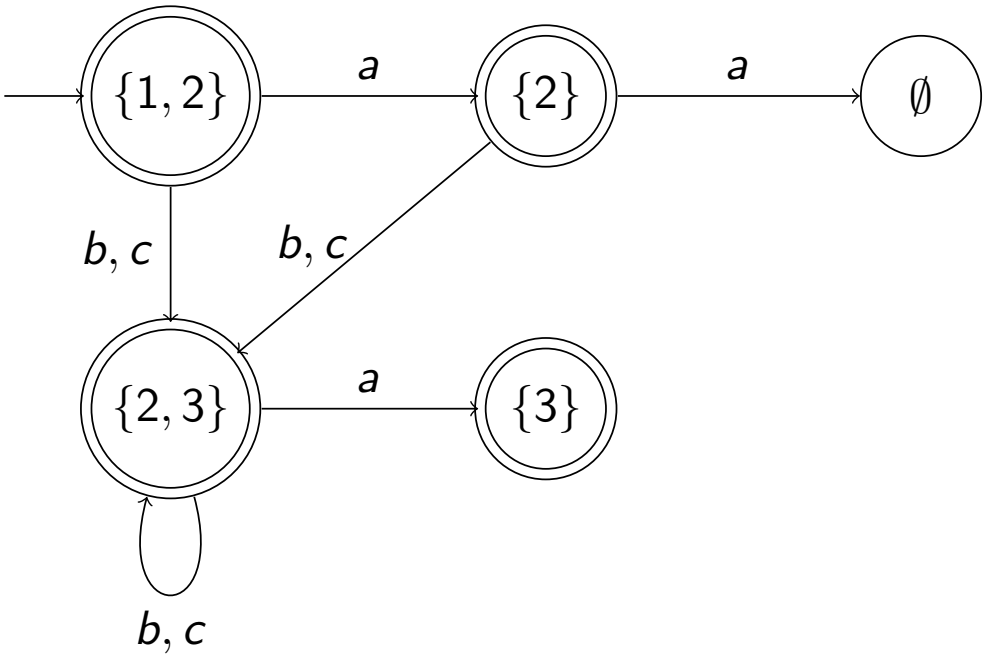
Übergänge vom Startzustand aus

Mit einem a erreichen wir von Zustand 1 aus Zustand 2, wohingegen kein Übergang a-Übergang von Zustand 2 aus existiert. Von Zustand 1 aus kann man mit b oder c kann man den Zustand 3 erreichen, von Zustand 2 aus den Zustand 2 oder den Zustand 3. Insgesamt kann man von $\{1, 2\}$ also mit b, c den Zustand $\{2, 3\}$ erreichen.



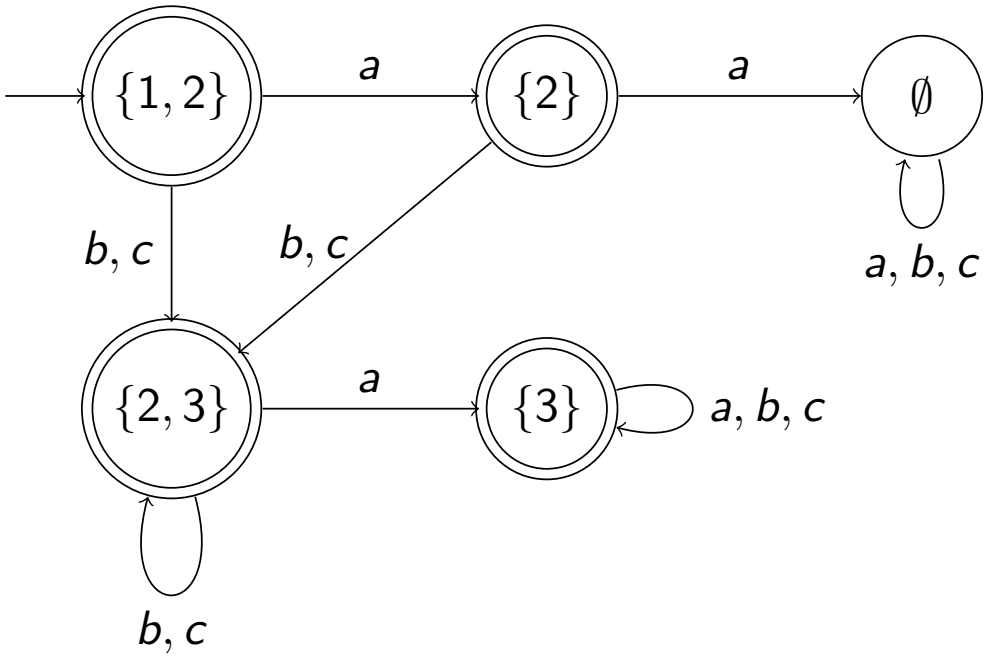
Übergänge von $\{2\}$, $\{2,3\}$ aus

Die Übergänge von Zustand $\{2\}$ sind unmittelbar ablesbar: Es gibt keinen Übergang mit a, es wird also $\emptyset$ erreicht, mit b und c können Zustände 2 und 3 erreicht werden. Von Zustand $\{2,3\}$ aus kann mit a der Zustand $\{3\}$ erreicht werden, da von 3 mit a ein Übergang zu 3 möglich ist und weitere a-Übergänge in 2 und 3 nicht existieren. Mit b, c ist ein Übergang zu $\{2,3\}$ möglich, da von 2 aus b, c-Übergänge zu 2 und 3 existieren.



Übergänge von $\{3\}$, $\emptyset$ aus

Die Übergänge von Zustand $\{3\}$ sind unmittelbar ablesbar: Für alle Eingabezeichen erreicht man wieder Zustand $\{3\}$. Für $\emptyset$ gilt hier und in jedem anderen Fall: Für jedes Alphabetsymbol bleibt man im Zustand $\emptyset$.



Nicht erreichbare Zustände

Durch diese Konstruktion haben wir alle erreichbaren Zustände des Potenzmengenautomaten erzeugt. Allerdings gibt es auch einige nicht erreichbare Zustände, die wir auf diese Weise nicht erzeugt haben (und in aller Regel auch nicht erzeugen wollen):
$\{1\}, \{1,3\}, \{1,2,3\}$.

NFAs, DFAs und reguläre Grammatiken

Bemerkungen zur Potenzmengenkonstruktion:

Wegen $|\mathcal{P}(Z)| = 2^{|Z|}$ hat der DFA exponentiell mehr Zustände als der dazugehörige NFA. Evtl. kann er aber noch verkleinert werden (z.B. durch Entfernen nicht-erreichbarer Zustände).

In vielen Fällen ist der kleinste DFA, der eine Sprache akzeptiert, tatsächlich exponentiell größer als der kleinste NFA. Ein Beispiel hierfür ist die folgende Sprache:

$$L_k = \{x \in \{0,1\}^* \mid |x| \geq k, \text{das } k\text{-letzte Zeichen von } x \text{ ist } 0\}$$

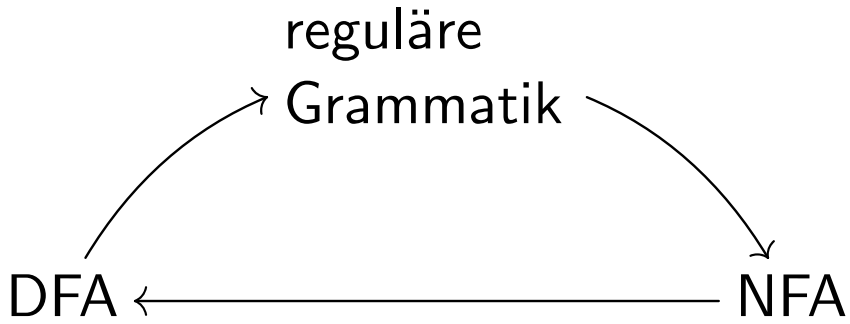
L_k wird durch einen NFA mit $k+1$ Zuständen erkannt und man kann zeigen, dass der kleinste DFA, der L_k erkennt, mindestens 2^k Zustände haben muss.

NFAs, DFAs und reguläre Grammatiken

Wir können nun

- NFAs in DFAs umwandeln
- DFAs in reguläre Grammatiken umwandeln

Es fehlt noch die Richtung "reguläre Grammatik $\rightarrow$ NFA", dann haben wir die Äquivalenz aller dieser Formalismen gezeigt.



NFAs, DFAs und reguläre Grammatiken

Reguläre Grammatiken $\rightarrow$ NFAs (Satz)

Zu jeder regulären Grammatik G gibt es einen NFA M mit $L(G) = T(M)$.

NFAs, DFAs und reguläre Grammatiken

Umwandlung reguläre Grammatik $\rightarrow$ NFA:

Gegeben sei eine reguläre Grammatik $G = (V, \Sigma, P, S)$, die die ε-Sonderregelung erfüllt. Wir erstellen einen NFA $M = (Z, \Sigma, \delta, S', E)$ mit

$$
\begin{aligned}
Z &= V \cup \{X\},\ X \notin V \\
S' &= \{S\} \\
E &= \begin{cases} \{S, X\} & \text{falls } (S \rightarrow \varepsilon) \in P \\ \{X\} & \text{falls } (S \rightarrow \varepsilon) \notin P \end{cases} \\
B &\in \delta(A, a) \quad \text{falls } (A \rightarrow aB) \in P \\
X &\in \delta(A, a) \quad \text{falls } (A \rightarrow a) \in P
\end{aligned}
$$

NFAs, DFAs und reguläre Grammatiken

Zwischenzusammenfassung

Wir haben verschiedene Modelle zur Beschreibung regulärer Sprachen kennengelernt:

- Reguläre Grammatiken: Schaffen die Verbindung zur Chomsky-Hierarchie. Werden zur Erzeugung von Sprachen eingesetzt. Sind weniger gut dazu geeignet, um zu entscheiden, ob sich ein bestimmtes Wort in der Sprache befindet.
- NFAs: Erlauben oft kleine, kompakte Darstellungen von Sprachen. Sind, wegen ihres Nichtdeterminismus, genauso wie Grammatiken weniger gut für die Lösung des Wortproblems geeignet. Besitzen aber eine intuitive graphische Notation.

NFAs, DFAs und reguläre Grammatiken

Zwischenzusammenfassung

Wir haben verschiedene Modelle zur Beschreibung regulärer Sprachen kennengelernt:

- DFAs: Können gegenüber äquivalenten NFAs exponentiell größer werden. Sobald man jedoch einen DFA gegeben hat, erlaubt dieser eine effiziente Lösung des Wortproblems (einfach den Übergängen des Automaten nachlaufen und überprüfen, ob ein Endzustand erreicht wird).

Alle Modelle benötigen jedoch relativ viel Schreibaufwand und Platz für die Notation. Gesucht wird also eine kompaktere Repräsentation: sogenannte reguläre Ausdrücke.

Reguläre Ausdrücke

Regulärer Ausdruck

Ein regulärer Ausdruck α ist von einer der folgenden Formen:

- $\emptyset$
- ε
- a mit $a \in \Sigma$
- $\alpha\beta$
- $(\alpha|\beta)$
- $(\alpha)^*$

wobei α, β reguläre Ausdrücke sind.

Bemerkung: Statt $(\alpha|\beta)$ wird oft auch $(\alpha + \beta)$ geschrieben.

Reguläre Ausdrücke

Nach der Festlegung der Syntax regulärer Ausdrücke, müssen wir auch deren Bedeutung festlegen, d.h., welcher reguläre Ausdruck steht für welche Sprache?

Sprache eines regulären Ausdrucks

- $L(\emptyset) = \emptyset$
- $L(\varepsilon) = \{\varepsilon\}$
- $L(a) = \{a\}$
- $L(\alpha\beta) = L(\alpha)L(\beta)$, wobei $L_1 L_2 = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$ für zwei Sprachen L_1, L_2.
- $L(\alpha|\beta) = L(\alpha) \cup L(\beta)$
- $L((\alpha)^*) = (L(\alpha))^*$, wobei $L^* = \{w_1 \dots w_n \mid n \in \mathbb{N}_0, w_i \in L\}$ für eine Sprache L

Reguläre Ausdrücke

Bemerkungen zum $*$-Operator: $L^* = \{w_1 \dots w_n \mid n \in \mathbb{N}_0, w_i \in L\}$

- Dieser Operator wird oft Kleenesche Hülle genannt. Nur durch ihn kann man unendliche Sprachen erzeugen.
- L^* enthält immer das leere Wort ε (siehe Definition).
- Beispiel für die Anwendung des $*$-Operators:
  $L = \{a, bb, cc\}$

  $\rightsquigarrow L^* =$
  $\{\varepsilon, a, bb, cc, aa, abb, acc, bba, bbbb, bbcc, cca, ccbb, cccc, \dots\}$

  Alle Kombinationen beliebiger Länge sind möglich.

Reguläre Ausdrücke

Beispiele für reguläre Ausdrücke über dem Alphabet $\Sigma = \{a, b\}$.

Beispiel 1: Sprache aller Wörter, die mit a beginnen und mit bb enden

$$\alpha = a(a|b)^*bb$$

Beispiel 2: Sprache aller Wörter, die das Teilwort aba enthalten.

$$\alpha = (a|b)^*aba(a|b)^*$$

Beispiel 3: Sprache aller Wörter, die gerade viele a's enthalten.

$$\alpha = (b^*ab^*a)^*b^* \qquad \text{oder} \qquad \alpha = (b \mid ab^*a)^*$$

Fragestellungen zu dieser Vorlesungseinheit

Zum Ende einer jeden Vorlesungseinheit betrachten wir im Regelfall drei Fragestellungen, die mithilfe der in dieser Einheit besprochenen Inhalte beantwortet werden sollen. In der darauffolgenden Einheit können zu Beginn mögliche Antworten gesammelt werden.

Fragen zur vierten Vorlesungseinheit

- Was unterscheidet NFA und DFA?
- Wie transformiert man einen NFA in einen äquivalenten DFA?
- Wie bildet man reguläre Ausdrücke?

Reguläre Ausdrücke

Reguläre Ausdrücke $\rightarrow$ NFAs

Zu jedem regulären Ausdruck γ gibt es einen NFA M mit $L(\gamma) = T(M)$.

Reguläre Ausdrücke

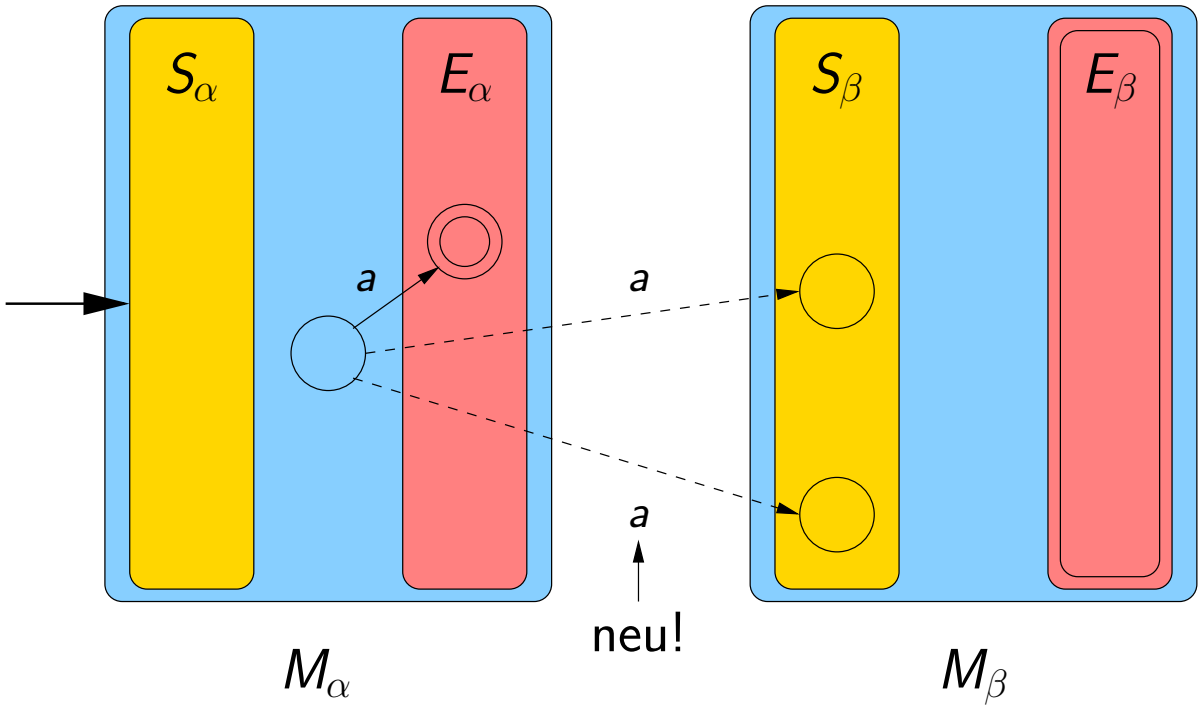
Beweis durch Induktion über den Aufbau von γ.

Für $\gamma = \emptyset$, $\gamma = \varepsilon$, $\gamma = a$ gibt es offensichtlich entsprechende Automaten.

Sei nun $\gamma = \alpha\beta$. Dann gibt es Automaten M_α, M_β mit $T(M_\alpha) = L(\alpha)$ und $T(M_\beta) = L(\beta)$. Wir schalten diese Automaten nun wie folgt hintereinander zu einem Automaten M:

- M hat als Zustände die Vereinigung beider Zustandsmengen, die gleichen Startzustände wie M_α und die gleichen Endzustände wie M_β. (Falls $\varepsilon \in L(\alpha)$, so sind auch die Startzustände von M_β Startzustände von M.)
- Alle Übergänge von M_α bzw. M_β bleiben erhalten.
- Alle Zustände, die einen Übergang zu einem Endzustand von M_α haben, erhalten zusätzlich genauso beschriftete Übergänge zu allen Startzuständen von M_β.

Reguläre Ausdrücke



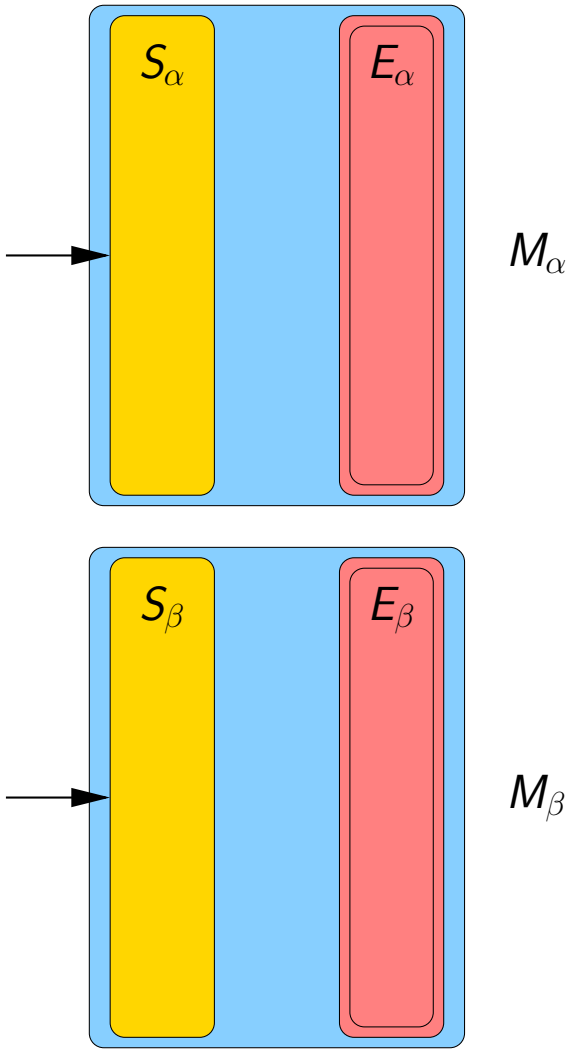
Es gilt $T(M) = T(M_\alpha)T(M_\beta) = L(\alpha)L(\beta)$

Reguläre Ausdrücke

Sei nun $\gamma = (\alpha \mid \beta)$. Dann gibt es Automaten M_α, M_β mit $T(M_\alpha) = L(\alpha)$ und $T(M_\beta) = L(\beta)$. Wir bauen nun aus diesen zwei Automaten einen Vereinigungsautomaten M:

- M hat als Zustände die Vereinigung beider Zustandsmengen. Ebenso ergeben sich die Startzustände als Vereinigung der Startzustandsmengen und die Endzustände als Vereinigung der Endzustandsmengen.
- Alle Übergänge von M_α bzw. M_β bleiben erhalten.

Reguläre Ausdrücke



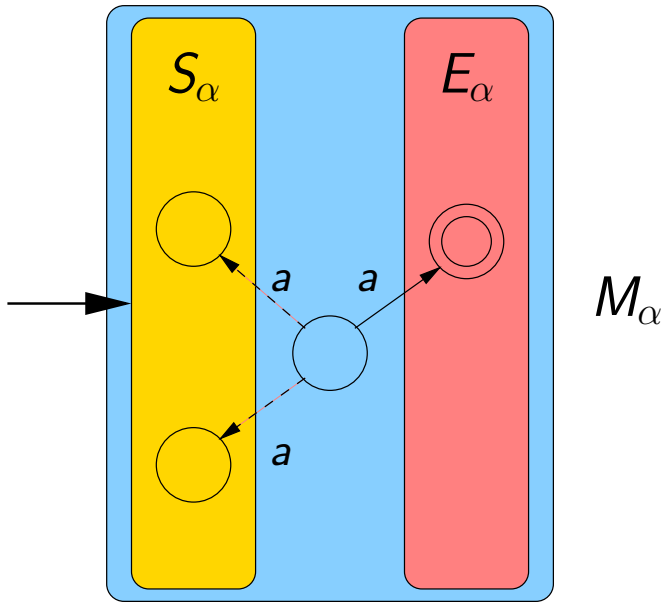
Es gilt $T(M) = T(M_\alpha) \cup T(M_\beta) = L(\alpha) \cup L(\beta)$

Reguläre Ausdrücke

Sei nun $\gamma = (\alpha)^*$. Dann gibt es einen Automaten M_α mit $T(M_\alpha) = L(\alpha)$. Wir bauen aus diesem Automaten nun wie folgt einen Automaten M:

- Alle Zustände, Start- und Endzustände sowie Übergänge bleiben erhalten.
- Zusätzlich erhalten alle Zustände, die einen Übergang zu einem Endzustand von M_α haben, genauso beschriftete Übergänge zu allen Startzuständen von M_α (Rückkopplung).
- Falls $\varepsilon \notin T(M_\alpha)$, so gibt es einen weiteren Zustand, der sowohl Start- als auch Endzustand ist. (Damit auch das leere Wort erkannt wird.)

Reguläre Ausdrücke



evtl. zusätzl. Zustand

Es gilt $T(M) = (T(M_\alpha))^* = (L(\alpha))^*$.

Reguläre Ausdrücke

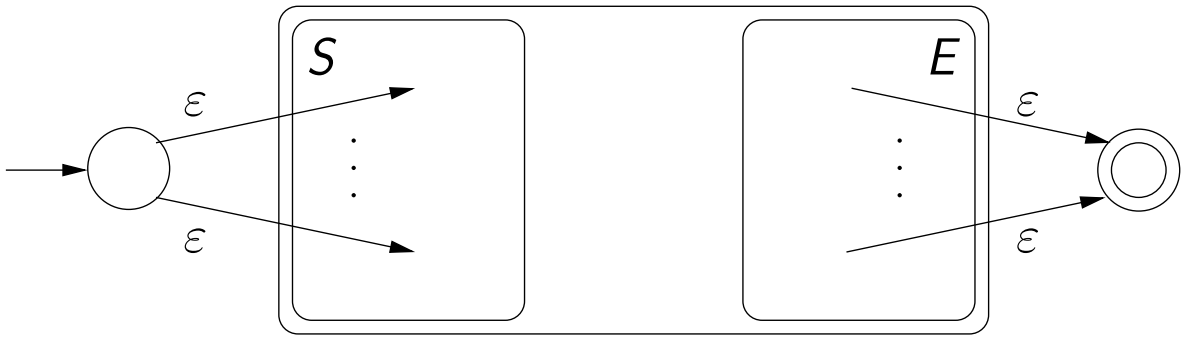
NFAs $\rightarrow$ Reguläre Ausdrücke

Zu jedem NFA M gibt es einen regulären Ausdruck γ mit $T(M) = L(\gamma)$.

Reguläre Ausdrücke

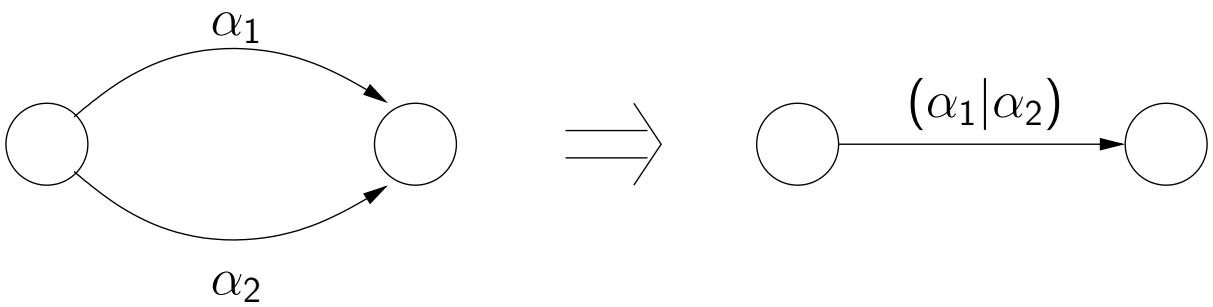
Wir verwenden das folgende Zustandseliminations-Verfahren, das einen NFA M in einen regulären Ausdruck verwandelt. Dabei erhält man als Zwischenzustände Automaten, deren Übergänge nicht mit Alphabetsymbolen, sondern mit regulären Ausdrücken beschriftet sind.

Zunächst führen wir einen neuen Startzustand und einen neuen Endzustand ein und verbinden die bisherigen Start- bzw. Endzustände mit den neuen Zuständen durch ε-Kanten.

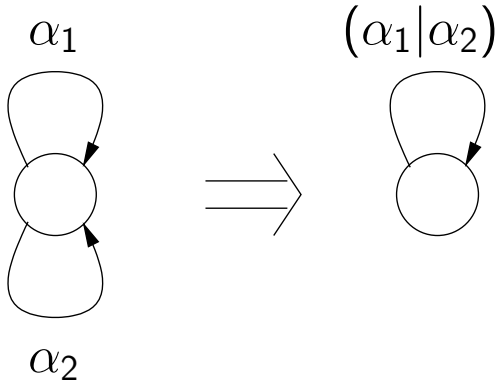


Reguläre Ausdrücke

Transformations-Regeln: Zwei parallel verlaufende Übergänge mit den Beschriftungen α_1 und α_2 können zu einer einzigen mit der Beschriftung $(\alpha_1 \mid \alpha_2)$ verschmolzen werden (Regel V).

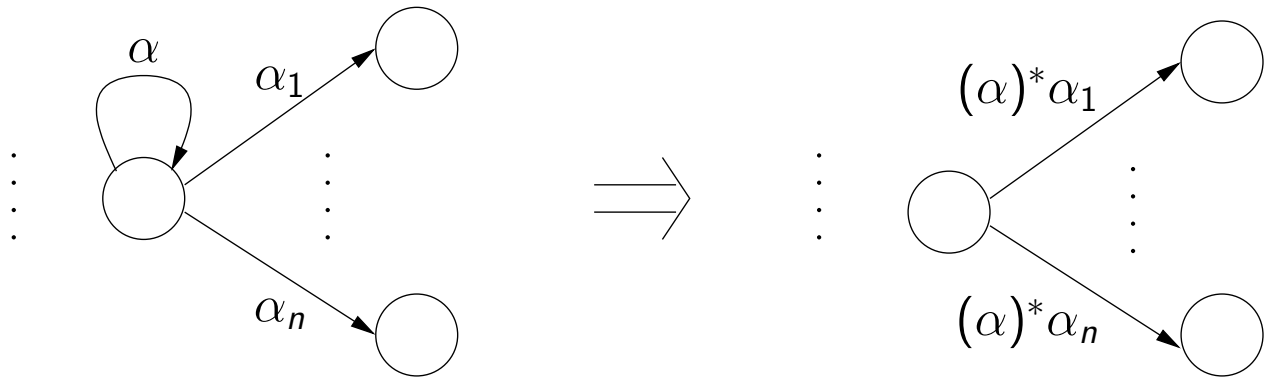


Gleiches gilt im Fall, wenn ein Zustand zwei Schleifen besitzt.



Reguläre Ausdrücke

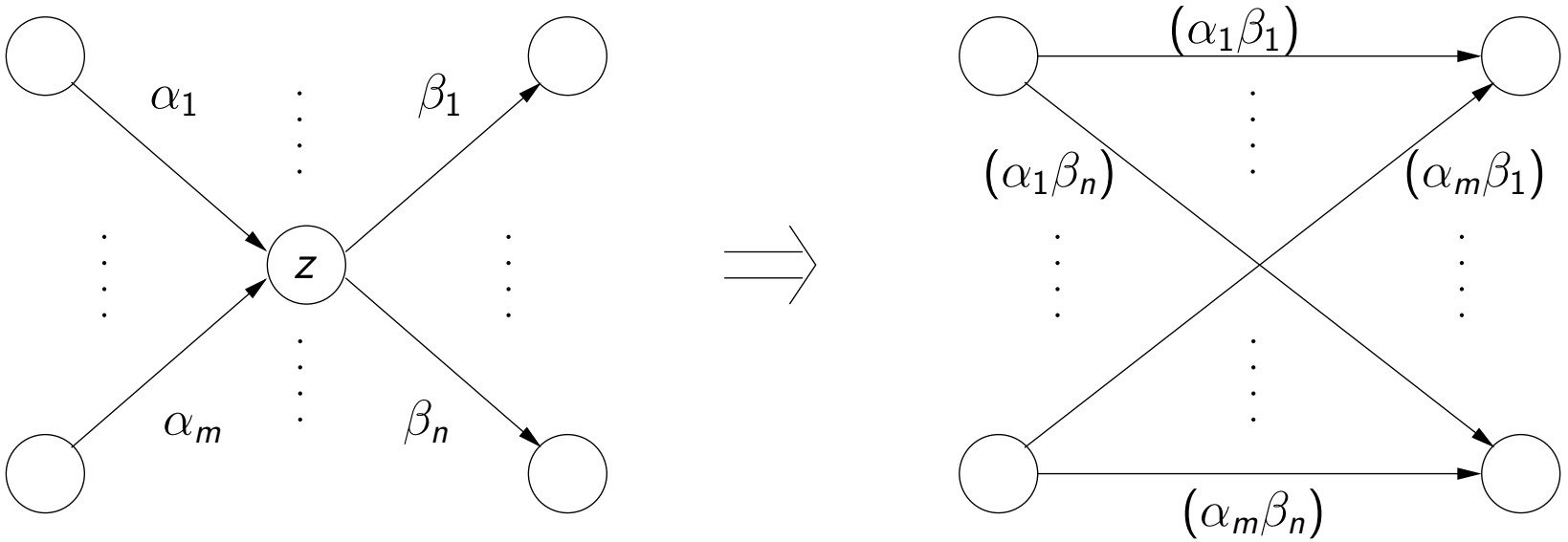
Schleifen werden entfernt, indem man ihre Beschriftung α (mit einem $*$ versehen) mit auf die nachfolgenden Kanten setzt. (Regel S).



Nur zulässig, wenn es sich dabei um die einzige Schleife des Zustands handelt.

Reguläre Ausdrücke

Ein Zustand z wird eliminiert, indem man die Zustände, von denen aus Kanten nach z hineinführen, und Zustände, in die Kanten von z aus hineinführen, geeignet miteinander verbindet (Regel E). Hierbei ergibt jedes Paar von eingehender und ausgehender Kante eine neue Kante.



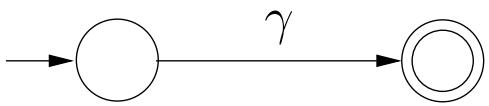
Reguläre Ausdrücke

Die Anwendung von Regel E ist nur zulässig, wenn:

- sich keine Schleife am zu entfernenden Zustand befindet *und*
- es mindestens eine nach z hineinführende und eine aus z herausführende Kante gibt.

Reguläre Ausdrücke

Sobald keine Regel mehr anwendbar ist, haben wir im Allgemeinen folgende Situation (plus evtl. zusätzliche Sackgassen):

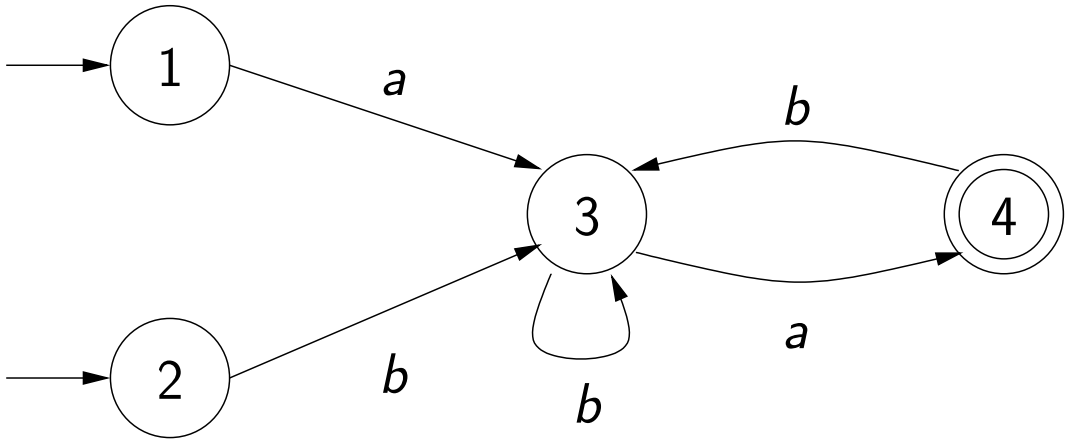


Dann ist γ der gesuchte reguläre Ausdruck.

Falls es keine Kante zwischen Anfangs- und Endzustand gibt:
$\gamma = \emptyset$.

Reguläre Ausdrücke

Beispiel: Umwandlung des folgenden nicht-deterministischen Automaten in einen regulären Ausdruck



Ergebnis: $(\varepsilon a|\varepsilon b)(b^*ab)^*b^*a\varepsilon$

Reguläre Ausdrücke

Wozu sind reguläre Ausdrücke in der Praxis nützlich?

- Suchen und Ersetzen in Editoren
- Pattern-Matching und Verarbeitung großer Texte und Datenmengen, z.B., beim Data-Mining
  (Tools: Stream-Editor `grep`, `sed`, `awk`, `perl`, ...)
- Übersetzung von Programmiersprachen:
  Lexikalische Analyse – Umwandlung einer Folge von Zeichen (das Programm) in eine Folge von Tokens, in der bereits die Schlüsselwörter, Bezeichner, Daten, etc. identifiziert sind.
  (Tools: `lex`, `flex`, ...)

Abschlusseigenschaften

Abgeschlossenheit (Definition)

Gegeben sei eine Menge M und ein binärer Operator $\otimes : M \times M \to M$.
Man sagt, eine Menge $M' \subseteq M$ ist unter $\otimes$ abgeschlossen, wenn für zwei beliebige Elemente $m_1, m_2 \in M'$ gilt: $m_1 \otimes m_2 \in M'$.

Wir betrachten hier Abschlusseigenschaften für die Menge aller regulärer Sprachen. Die interessante Frage ist:

> Falls L_1, L_2 regulär sind, sind dann auch $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 L_2$, $\overline{L_1} = \Sigma^* \backslash L_1$ (Komplement) und L_1^* regulär?

Kurze Antwort: Die regulären Sprachen sind unter allen diesen Operationen abgeschlossen.

Abschlusseigenschaften

Warum sind Abschlusseigenschaften interessant?

Sie sind vor allem dann interessant, wenn sie konstruktiv verwirklicht werden können, das heißt, wenn man – gegeben Automaten für L_1 und L_2 – auch einen Automaten beispielsweise für den Schnitt von L_1 und L_2 konstruieren kann.

Damit hat man dann mit Automaten eine Datenstruktur für unendliche Sprachen, die man maschinell weiterverarbeiten kann.

Abschlusseigenschaften

Abschluss unter Vereinigung

Wenn L_1 und L_2 reguläre Sprachen sind, dann ist auch $L_1 \cup L_2$ regulär.

Begründung: den (nicht-deterministischen) Automaten für $L_1 \cup L_2$ kann man mit denselben Methoden bauen wie den Automaten für $L(\alpha|\beta)$ bei der Umwandlung von regulären Ausdrücken in NFAs.

Think-Pair-Share: Abschlusseigenschaften

Abschluss unter Komplement

Wenn L eine reguläre Sprache ist, dann ist auch $\overline{L} = \Sigma^* \setminus L$ regulär.

Bemerkung: bei Bildung des Komplements muss immer festgelegt werden, bezüglich welcher Obermenge das Komplement gebildet werden soll. Hier ist das die Menge Σ^* aller Wörter über dem Alphabet Σ, das gerade betrachtet wird.

Wir untersuchen nun, wieso diese Abschlusseigenschaft erfüllt ist. Sei dazu ein DFA $M = (Z, \Sigma, \delta, z_0, E)$ für L gegeben. Wie kann man auf dieser Grundlage einen DFA M' konstruieren, der die Komplementsprache $\Sigma^* \setminus L$ akzeptiert?
Erarbeiten Sie zunächst fünf Minuten in Einzelarbeit eine Lösung. Anschließend tauschen Sie sich für weitere fünf Minuten mit ihrem Sitznachbarn aus. Schlussendlich besprechen wir die Lösung im Plenum. Es reicht, wenn Sie die Konstruktion angeben.

Lösungsvorschlag zur Think-Pair-Share-Aufgabe

Begründung: Aus einem DFA $M = (Z, \Sigma, \delta, z_0, E)$ für L gewinnt man leicht einen DFA M' für $\overline{L}$ indem man die End- und Nicht-Endzustände vertauscht. D.h. $M' = (Z, \Sigma, \delta, z_0, Z \backslash E)$.

Dann gilt:

$$w \in L \iff \hat{\delta}(z_0, w) \in E \iff \hat{\delta}(z_0, w) \notin Z \backslash E \iff w \notin \overline{L}.$$

Abschlusseigenschaften

Abschluss unter Produkt/Konkatenation

Wenn L_1 und L_2 reguläre Sprachen sind, dann ist auch $L_1 L_2$ regulär.

Begründung: den (nicht-deterministischen) Automaten für $L_1 L_2$ kann man mit denselben Methoden bauen wie den Automaten für $L(\alpha\beta)$ bei der Umwandlung von regulären Ausdrücken in NFAs.

Abschlusseigenschaften

Abschluss unter der Stern-Operation

Wenn L eine reguläre Sprache ist, dann ist auch L^* regulär.

Begründung: den (nicht-deterministischen) Automaten für L^* kann man mit denselben Methoden bauen wie den Automaten für $L((\alpha)^*)$ bei der Umwandlung von regulären Ausdrücken in NFAs.

Abschlusseigenschaften

Abschluss unter Schnitt

Wenn L_1 und L_2 reguläre Sprachen sind, dann ist auch $L_1 \cap L_2$ regulär.

Begründung 1: Es gilt $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ und wir wissen bereits, dass reguläre Sprachen und Komplement und Vereinigung abgeschlossen sind.

Abschlusseigenschaften

Begründung 2: Es gibt noch eine andere direktere Konstruktion für den Schnitt. Dabei werden die zwei Automaten für L_1 und L_2 miteinander synchronisiert und quasi "parallelgeschaltet". Dies erfolgt durch das Bilden des Kreuzprodukts.

Seien $M_1 = (Z_1, \Sigma, \delta_1, S_1, E_1)$, $M_2 = (Z_2, \Sigma, \delta_2, S_2, E_2)$ NFAs mit $T(M_1) = L_1$ und $T(M_2) = L_2$. Dann akzeptiert folgender Automat M die Sprache $L_1 \cap L_2$:

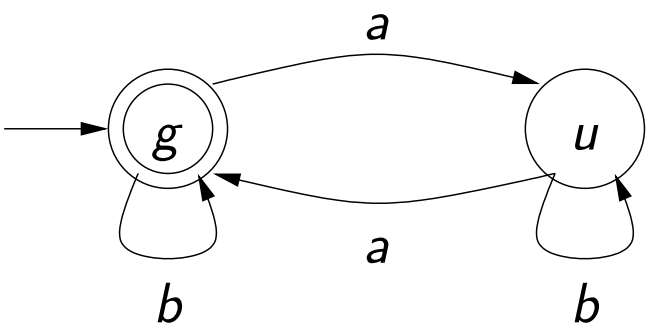
$$M = (Z_1 \times Z_2, \Sigma, \delta, S_1 \times S_2, E_1 \times E_2),$$

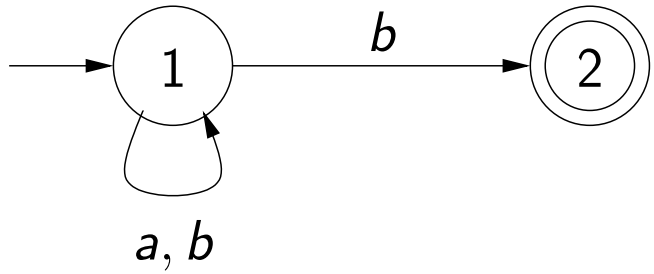
wobei $\delta((z_1, z_2), a) = \{(z_1', z_2') \mid z_1' \in \delta_1(z_1, a), z_2' \in \delta_2(z_2, a))\}$.

M akzeptiert ein Wort w genau dann, wenn sowohl M_1 als auch M_2 das Wort w akzeptieren.

Abschlusseigenschaften

Beispiel für ein Kreuzprodukt: bilde das Kreuzprodukt der folgenden zwei Automaten:





Abschlusseigenschaften

Wir betrachten eine Anwendung des Kreuzprodukts auf Adventures. Wiederholung der Regeln für Level 1:

Die Schatz-Regel

Man muss mindestens zwei Schätze finden.

Die Tür-Regel

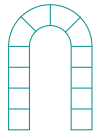
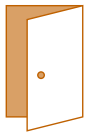
Durch eine Tür kann man nur gehen, wenn man zuvor einen Schlüssel gefunden hat. (Dieser Schlüssel darf aber dann beliebig oft verwendet werden.)

Abschlusseigenschaften

Die Drachen-Regel

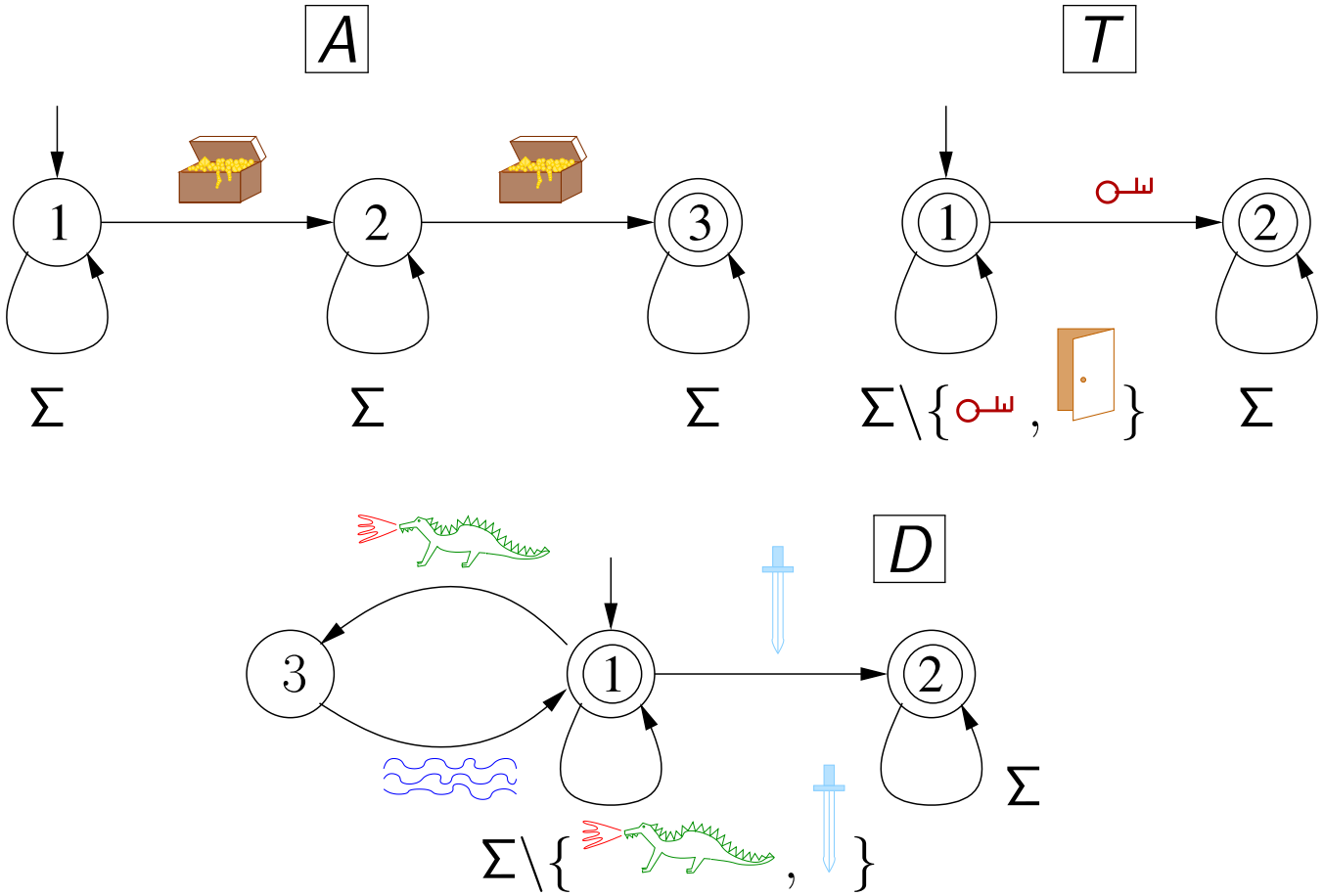
Unmittelbar nach der Begegnung mit einem Drachen muss man in einen Fluss springen, da uns der Drache in Brand stecken wird. Dies gilt nicht mehr, sobald man ein Schwert besitzt, mit dem man den Drachen vorher töten kann.

Alphabetsymbole:

- Drachen (D): 
- Schwert (W): 
- Fluss (F): 
- Torbogen (B): 
- Tür (T): 
- Schlüssel (L): 
- Schatz (A): 

Abschlusseigenschaften

Man kann diese Regeln durch folgende endliche Automaten beschreiben:



Abschlusseigenschaften

Gegeben sei ein Automat M, der eine Adventure-Karte beschreibt. Sei

- $L_M = T(M)$ die Sprache aller Pfade durch M von einem Anfangs- zu einem Endzustand,
- $L_A = T(A)$ die Menge aller Pfade, die die Schatz-Regel erfüllen,
- $L_T = T(T)$ die Menge aller Pfade, die die Tür-Regel erfüllen *und*
- $L_D = T(D)$ die Menge aller Pfade, die die Drachen-Regel erfüllen.

Außerdem sei A_M die Menge aller Pfade durch die Adventure-Karte, die alle Bedingungen erfüllen. Offensichtlich gilt:

$$A_M = L_M \cap L_A \cap L_T \cap L_D$$

Abschlusseigenschaften

Damit haben wir ein Verfahren, um das Adventure-Problem (Level 1) zu lösen, d.h., um zu überprüfen, ob ein Adventure eine Lösung hat:

1. Bilde nacheinander das Kreuzprodukt der vier Automaten M, A, T, D (das Kreuzprodukt ist assoziativ und daher die Reihenfolge gleichgültig).
2. Überprüfe, ob der dadurch entstehende Automat mindestens ein Wort akzeptiert, d.h., ob es einen Pfad von einem Anfangs- zu einem Endzustand gibt.

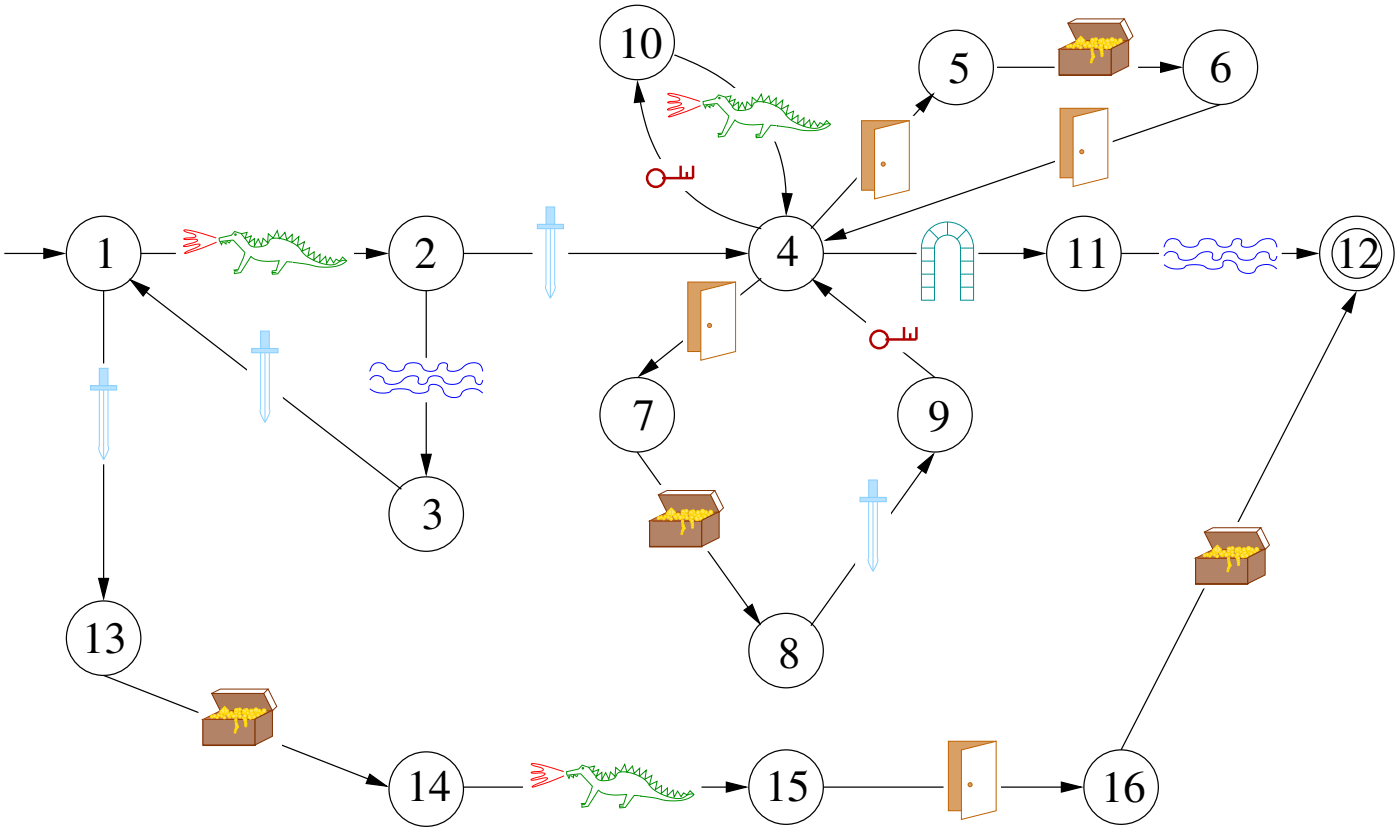
Dies kann automatisch erfolgen, beispielsweise mit dem Tool Grail zur Manipulation endlicher Automaten:
`http://www3.cs.stonybrook.edu/~algorith/implement/grail/implement.shtml`

Abschlusseigenschaften

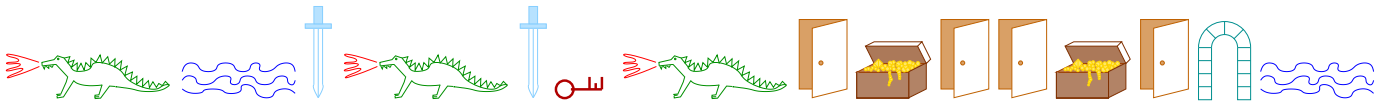
Kürzeste Lösungen, ermittelt mit Grail (Befehle `fmcross`, `fmenum`):

```
> fmcross a.aut < t.aut > at.aut
> fmcross at.aut < d.aut > atd.aut
> fmcross m.aut < atd.aut > loesung.aut
> fmenum loesung.aut
DFWDWLDTATTATBF
DFWDWLDTATTAWLBF
DFWDWLDTAWLTATBF
DFWDWLDLDTATTATBF
...
```

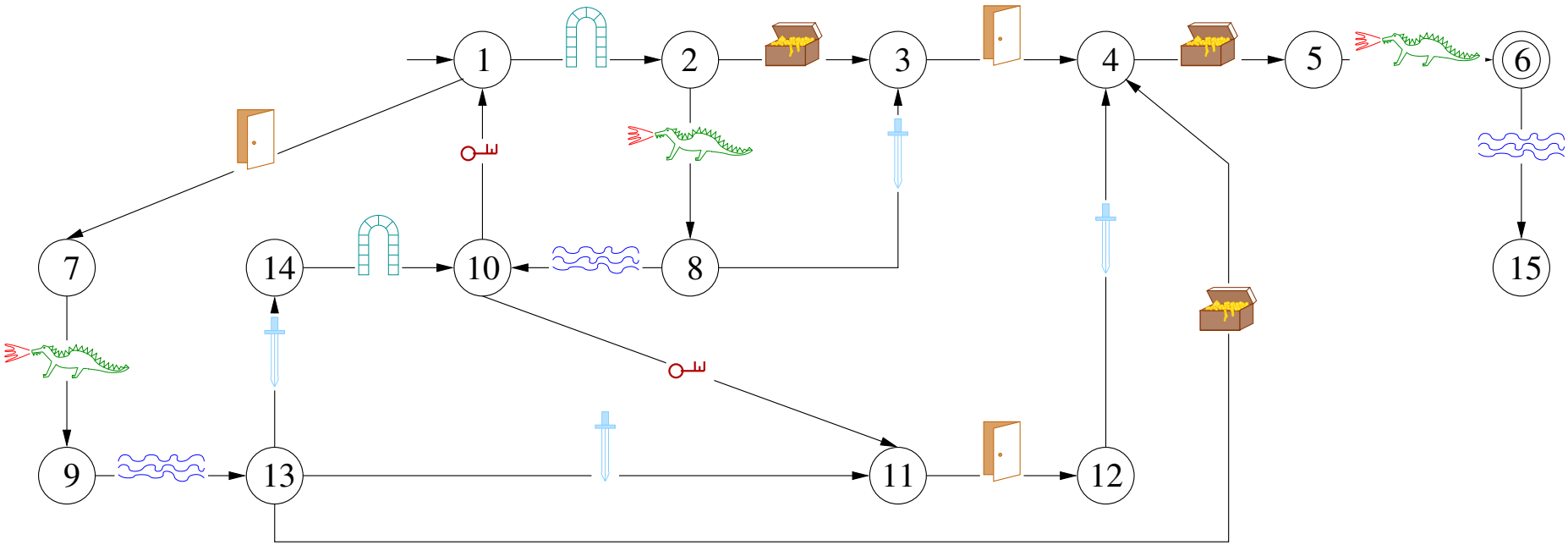
Abschlusseigenschaften



DFWDWLDTATTATBF =



Abschlusseigenschaften



BDFLTDFWBLBATAD =

Ausblick

Weitere interessante Fragen

- Wie kann man zeigen, dass eine Sprache *nicht* regulär ist?

  Beispiel: Die Sprache $\{a^n b^n c^n \mid n \geq 1\}$, die bereits als Beispiel auftauchte, scheint nicht regulär zu sein. Wie kann man das zeigen?
- Wenn eine Sprache regulär ist, wie groß ist dann der kleinste Automat, der die Sprache akzeptiert? Gibt es überhaupt *den* kleinsten Automaten?

Fragestellungen zu dieser Vorlesungseinheit

Zum Ende einer jeden Vorlesungseinheit betrachten wir im Regelfall drei Fragestellungen, die mithilfe der in dieser Einheit besprochenen Inhalte beantwortet werden sollen. In der darauffolgenden Einheit können zu Beginn mögliche Antworten gesammelt werden.

Fragen zur fünften Vorlesungseinheit

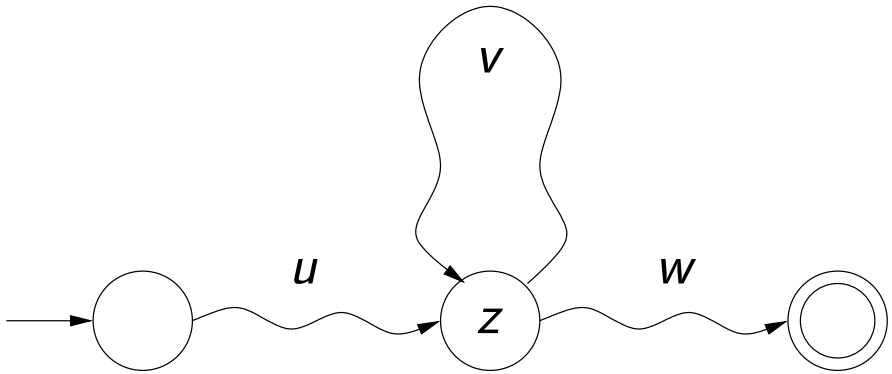
- Wie wandelt man einen NFA in einen regulären Ausdruck um?
- Wie wandelt man einen regulären Ausdruck in einen NFA um?
- Wie konstruiert man den Automaten für die Komplementsprache?

Das Pumping-Lemma

Wie beweist man, dass eine Sprache L *nicht* regulär ist?

Idee: Man versucht auszunutzen, dass eine reguläre Sprache von einem Automat mit endlich vielen Zuständen akzeptiert werden muss. Das bedeutet auch: wenn ein Wort $x \in L$ ausreichend lang ist – nämlich mindestens so viele Zeichen lang ist wie es Zustände im Automaten gibt – so besucht man damit beim Durchlauf durch den Automaten mindestens einen Zustand z zweimal (Taubenschlag Prinzip).

Das Pumping-Lemma



Die dadurch entstehende Schleife kann nun mehrfach (oder gar nicht) durchlaufen werden, dadurch wird das Wort $x = uvw$ "aufgepumpt" und man stellt fest, dass uv^2w, uv^3w, ... sowie uw ebenfalls in L liegen müssen.

Bemerkung: Es gilt $v^i = \underbrace{v \dots v}_{i\text{-mal}}$.

Das Pumping-Lemma

Außerdem kann man für u, v, w folgende Eigenschaften verlangen, wobei n die Anzahl der Zustände des Automaten ist.

1. $|v| \geq 1$: Die Schleife ist auf jeden Fall nicht trivial und enthält zumindest einen Übergang.
2. $|uv| \leq n$: Spätestens nach n Alphabetsymbolen wird der Zustand z das zweite Mal erreicht.

Die Idee bei dieser Einschränkung ist, dass man „die erste“ Schleife im Automaten identifiziert, die vollendet wird.

Das Pumping-Lemma

Pumping-Lemma, *uvw*-Theorem (Satz)

Sei L eine reguläre Sprache. Dann gibt es eine Zahl n, so dass sich alle Wörter $x \in L$ mit $|x| \geq n$ zerlegen lassen in $x = uvw$, so dass folgende Eigenschaften erfüllt sind:

1. $|v| \geq 1$,
2. $|uv| \leq n$ *und*
3. für alle $i = 0, 1, 2, \ldots$ gilt: $uv^i w \in L$.

Dabei ist n die Anzahl der Zustände eines Automaten, der L erkennt. Dieses Lemma spricht jedoch nicht über Automaten, sondern nur über die Eigenschaften der Sprache. Daher ist es dazu geeignet, Aussagen über Nicht-Regularität zu machen.

Das Pumping-Lemma

Wie kann man das Pumping-Lemma dazu nutzen, um zu zeigen, dass eine Sprache nicht regulär ist?

Aussage des Pumping-Lemmas mit logischen Operatoren:

L regulär
$$\rightarrow \quad \exists n \quad \forall x \in L, |x| \geq n \; \exists u, v, w, x = uvw \quad \forall i \quad (uv^i w \in L)$$

Das ist logisch äquivalent zu

$$\forall n \quad \exists x \in L, |x| \geq n \quad \forall u, v, w, x = uvw \quad \exists i \quad (uv^i w \notin L)$$
$\rightarrow L$ ist nicht regulär

$$A \rightarrow B \equiv \neg B \rightarrow \neg A \text{ und } \neg \forall x \exists y F \equiv \exists x \forall y \neg F.$$

Das Pumping-Lemma

Pumping-Lemma (alternative Formulierung)

Sei L eine Sprache. Angenommen, wir können für jede Zahl n ein Wort $x \in L$ mit $|x| \geq n$ wählen, so dass folgendes gilt: für alle Zerlegungen $x = uvw$ mit

1. $|v| \geq 1$,
2. $|uv| \leq n$

gibt es eine Zahl i mit $uv^i w \notin L$. Dann ist L nicht regulär.

D.h., wir müssen zeigen, dass es für jedes n (für jede mögliche Anzahl von Zuständen) ein Wort gibt, das mindestens so lang wie n ist und das keine "pumpbare" Zerlegung hat.

Pumping-Lemma

"Kochrezept" für das Pumping-Lemma

Gegeben sei eine Sprache L (Beispiel: $\{a^k b^k \mid k \geq 0\}$). Wir wollen zeigen, dass sie nicht regulär ist.

1. Nehme eine *beliebige* Zahl n an. Diese Zahl darf nicht frei gewählt werden.
2. Wähle ein Wort $x \in L$ mit $|x| \geq n$. Damit das Wort auch wirklich mindestens die Länge n hat, empfiehlt es sich, dass n (beispielsweise als Exponent) im Wort auftaucht.

   Beispiel: $x = a^n b^n$

Pumping-Lemma

"Kochrezept" für das Pumping-Lemma

3. Betrachte nun *alle* möglichen Zerlegungen $x = uvw$ mit den Einschränkungen $|v| \geq 1$ und $|uv| \leq n$.

   Beispiel: hier gibt es nur eine mögliche Zerlegung $u = a^j$, $v = a^\ell$, $w = a^m b^n$ mit $j + \ell + m = n$ und $\ell \geq 1$.

4. Wähle für *jede* dieser Zerlegungen ein i (das kann jedes Mal ein anderes i sein), so dass $uv^i w \notin L$. (In vielen Fällen sind $i = 0$ und $i = 2$ eine gute Wahl.)

   Beispiel: wähle $i = 2$, dann gilt $uv^2w = a^{j+2\ell+m}b^n \notin L$, da $j + 2\ell + m \neq n$.

Pumping-Lemma

Als weiteres Beispiel betrachten wir das Adventure, Level 2.
Wiederholung der Regeln:

Die Schatz-Regel

Man muss mindestens zwei Schätze finden.

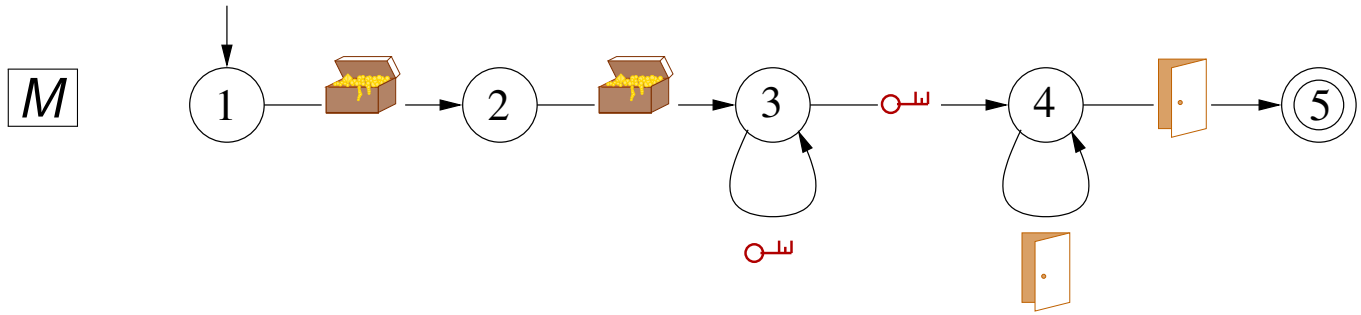
Die Drachen-Regel

Unmittelbar nach der Begegnung mit einem Drachen muss man in einen Fluss springen, da uns der Drache in Brand stecken wird. Dies gilt nicht mehr, sobald man ein Schwert besitzt, mit dem man den Drachen vorher töten kann.

Neue Tür-Regel

Die Schlüssel sind magisch und verschwinden sofort, nachdem eine Tür mit ihnen geöffnet wurde. Sobald man eine Tür durchschritten hat, schließt sie sich sofort wieder.

Pumping-Lemma



Wir betrachten folgende Sprache A_M:

$$\begin{aligned} A_M &= \{w \mid w \text{ entspricht einem Pfad durch das oben} \\ &\qquad \text{angegebene Adventure, d.h., } w \in T(M), \text{ und} \\ &\qquad \text{erfüllt alle Regeln für Level 2}\} \\ &= \{\, A^2 L^k T^m \mid k \geq m \geq 1 \,\} \end{aligned}$$

$L =$ Schlüssel $\qquad T =$ Tür $\qquad A =$ Schatz

Pumping-Lemma

Wir zeigen nun, dass A_M nicht regulär ist.

1. Gegeben sei eine beliebige Zahl n.
2. Wir wählen als Wort $x = A^2 L^n T^n \in A_M$.
3. Sei nun $x = uvw$ eine beliebige Zerlegung von x mit $|v| \geq 1$ und $|uv| \leq n$. Dann enthält v nur Schätze (A) oder Schlüssel (L) (aber keine Türen T).
4. Wir machen nun folgende Fallunterscheidung:
   - v enthält zumindest einen Schatz: dann enthält uv^0w höchstens noch einen Schatz und kann nicht in A_M liegen, da die Schatz-Regel verletzt ist. ($i = 2$ ist hier auch möglich.)
   - v enthält zumindest einen Schlüssel: dann enthält uv^0w weniger als n Schlüssel und kann nicht in A_M liegen, da es für jede der n Türen vorher mindestens einen Schlüssel geben muss.

Pumping-Lemma

Falsche Anwendung des Pumping-Lemmas

"Wenn L die Pumping-Eigenschaft erfüllt (d.h., es gibt ein n, so dass alle Wörter länger als n pumpbar sind), dann ist L regulär."
Dieses Argument ist nicht korrekt.

Es gibt nicht-reguläre Sprachen, die trotzdem die Pumping-Eigenschaft erfüllen.

Beispiel für nicht-reguläre Sprache mit der Pumping-Eigenschaft

$$L = \{a^k b^m c^m \mid k, m \geq 0\} \cup \{b^i c^j \mid i, j \geq 0\}.$$

- L erfüllt die Pumping-Eigenschaft: Sei $n \in \mathbb{N}$ und ein Wort $w \in L$ mit $|x| > n$ beliebig gegeben. Führe eine Fallunterscheidung danach durch, ob x mit a beginnt. Falls ja, wähle $u = \varepsilon$, $v = a$, w ist der Rest des Wortes, dann ist $uv^i w \in \{a^k b^m c^m \mid k, m \geq 0\}$. Anderenfalls wähle $u = \varepsilon$, $v = b$, falls x mit b beginnt, sonst $v = c$ und w is der Rest des Wortes. Dann ist $uv^i w \in \{b^i c^j \mid i, h \geq 0\}$.

Beispiel für nicht-reguläre Sprache mit der Pumping-Eigenschaft

$$L = \{a^k b^m c^m \mid k, m \geq 0\} \cup \{b^i c^j \mid i, j \geq 0\}.$$

- Aber L ist nicht regulär. (Argumentation mit Hilfe von Abschluss regulärer Sprachen unter Schnitt: $L \cap L(a(b \mid c)^*) = \{ab^m c^m \mid m \geq 0\}$ und für diese Sprache kann man mit dem Pumping-Lemma zeigen, dass sie nicht regulär ist.)

Äquivalenzrelationen und Minimalautomat

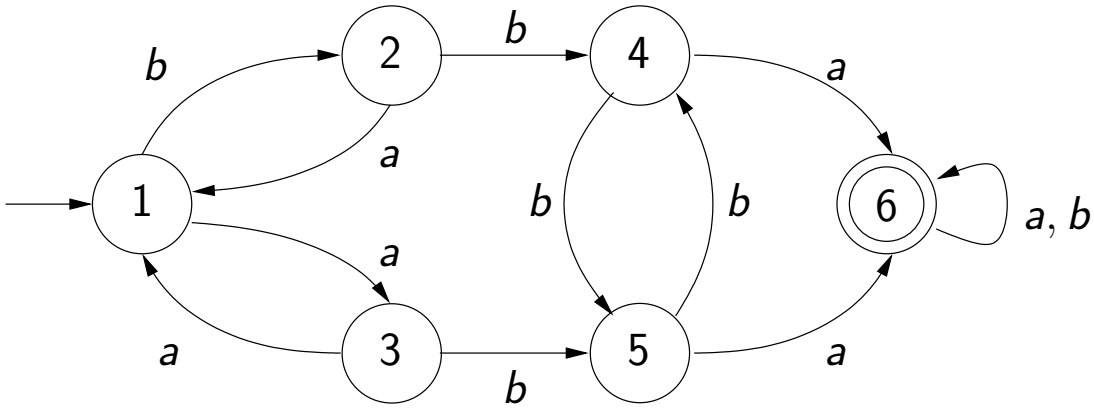
Minimalautomat

Wir beschäftigen uns nun mit folgenden Fragen:

- Gibt es zu jeder regulären Sprache immer *den* kleinsten deterministischen/nicht-deterministischen Automat?
- Kann man direkt aus der Sprache die Anzahl der Zustände des minimalen Automaten ablesen?
- Wie bestimmt man den minimalen Automat?

Äquivalenzrelationen und Minimalautomat

Wir betrachten folgenden Automaten M:



Feststellung: für die Zustände 4, 5 gilt

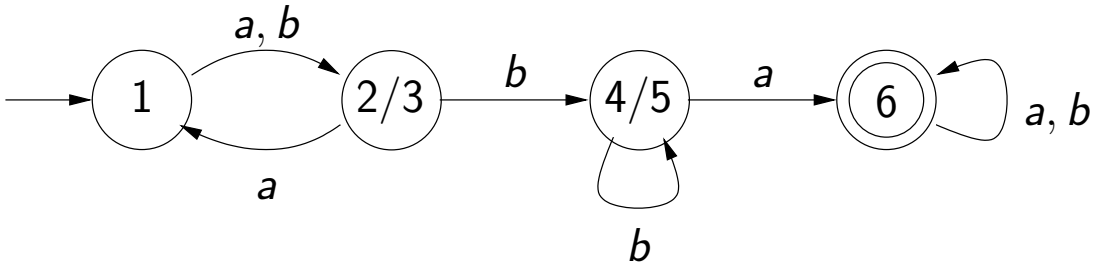
- mit einem Wort, das ein a enthält, landet man von dort aus immer im Zustand 6 (Endzustand)
- mit einem Wort, das kein a enthält, landet man von dort aus immer im Zustand 4 bzw. 5 (kein Endzustand)

Daraus folgt: 4 und 5 sind erkennungsäquivalent und können zu einem Zustand verschmolzen werden.

Äquivalenzrelationen und Minimalautomat

Ebenso: die Zustände 2 und 3 sind erkennungsäquivalent

Entstehender Automat M':



Jetzt sind keine Zustände mehr erkennungsäquivalent und sie können daher nicht weiter verschmolzen werden $\rightsquigarrow$ der Automat M' ist minimal für diese Sprache.

Äquivalenzrelationen und Minimalautomat

Erkennungsäquivalenz (Definition)

Gegeben sei ein DFA M. Zwei Zustände z_1, z_2 heißen erkennungsäquivalent genau dann, wenn für jedes Wort $w \in \Sigma^*$ gilt:

$$\hat{\delta}(z_1, w) \in E \iff \hat{\delta}(z_2, w) \in E.$$

Äquivalenzrelationen und Minimalautomat

Was ist eine Äquivalenzrelation?

Wir beginnen zunächst mit der Definition einer Relation:

Relation

Eine (zweistellige) Relation R auf einer Menge M ist eine Teilmenge $R \subseteq M \times M$.
Statt $(m_1, m_2) \in R$ schreibt man manchmal auch $m_1 \, R \, m_2$.

Äquivalenzrelationen und Minimalautomat

Äquivalenzrelation

Eine Äquivalenzrelation R auf einer Menge M ist eine Relation $R \subseteq M \times M$, die folgende Eigenschaften erfüllt:

- R ist reflexiv, d.h., es gilt $(m, m) \in R$ für alle $m \in M$.
- R ist symmetrisch, d.h., falls $(m_1, m_2) \in R$, so auch $(m_2, m_1) \in R$.
- R ist transitiv, d.h., aus $(m_1, m_2) \in R$ und $(m_2, m_3) \in R$ folgt $(m_1, m_3) \in R$.

Typische Beispiele für Äquivalenzrelationen auf natürlichen Zahlen: Gleichheit, Gleichheit modulo k, ...

Äquivalenzrelationen und Minimalautomat

Äquivalenzklasse

Sei R eine Äquivalenzrelation auf M und $m \in M$. Die Äquivalenzklasse $[m]_R$ von m ist folgende Menge:

$$[m]_R = \{n \in M \mid (n, m) \in R\}$$

Manchmal schreibt man auch nur $[m]$, wenn klar ist, welche Relation gemeint ist.

Äquivalenzrelationen und Minimalautomat

Eigenschaften von Äquivalenzklassen

Sei R eine Äquivalenzrelation auf M und $m_1, m_2 \in M$.
Dann gilt entweder

$$[m_1]_R = [m_2]_R$$

oder

$$[m_1]_R \cap [m_2]_R = \emptyset.$$

Außerdem gilt:

$$M = \bigcup_{m \in M} [m]_R.$$

D.h., zwei Äquivalenzklassen sind entweder gleich oder vollständig disjunkt. Außerdem überdecken sie M vollständig.
Man sagt auch: die Äquivalenzklassen bilden eine Partition von M.

Äquivalenzrelationen und Minimalautomat

Jedem Wort $x \in \Sigma^*$ kann man in einem deterministischen Automat einen eindeutigen Zustand $z = \hat{\delta}(z_0, x)$ zuordnen. Daher kann die Definition der Erkennungsäquivalenz auf Wörter aus Σ^* und Sprachen (anstatt Automaten) ausgedehnt werden.

Myhill-Nerode-Äquivalenz (Definition)

Gegeben sei eine Sprache L und Wörter $x, y \in \Sigma^*$.
Wir definieren eine Äquivalenzrelation R_L mit $x \, R_L \, y$ genau dann wenn

$$\text{für alle } z \in \Sigma^* \text{ gilt } (xz \in L \iff yz \in L).$$

Das ist gleichbedeutend damit, dass $\hat{\delta}(z_0, x)$ und $\hat{\delta}(z_0, y)$ erkennungsäquivalent sind, und zwar für einen beliebigen Automaten M, der L akzeptiert.

Äquivalenzrelationen und Minimalautomat

Beispiel 1 für Myhill-Nerode-Äquivalenz:

$$\text{Sprache } L = \{w \in \{a,b\}^* \mid \#_a(w) \text{ gerade}\}$$

Es gibt folgende Äquivalenzklassen:

- $[\varepsilon] = \{w \in \{a,b\}^* \mid \#_a(w) \text{ gerade}\} = L$
  (Äquivalenzklasse von ε)
- $[a] = \{w \in \{a,b\}^* \mid \#_a(w) \text{ ungerade}\} = \{a,b\}^* \setminus L$
  (Äquivalenzklasse von a)

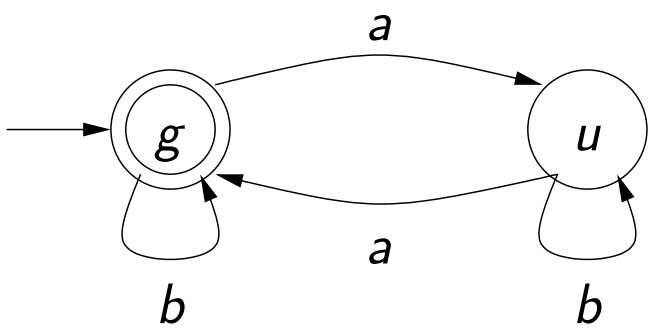
Beispiel: ε und aa sind äquivalent, denn

- wird an beide ein Wort mit gerade vielen a's angehängt, so bleiben sie in der Sprache
- wird an beide ein Wort mit ungerade vielen a's angehängt, so fallen sie aus der Sprache heraus

Äquivalenzrelationen und Minimalautomat

$L = \{w \in \{a, b\}^* \mid \#_a(w) \text{ gerade}\}$

Automat:



Äquivalenzrelationen und Minimalautomat

Beispiel 2 für Myhill-Nerode-Äquivalenz:

Sprache

$L = \{w \in \{a, b, c\}^* \mid$ das Teilwort abc kommt in w nicht vor$\}$

Es gibt folgende Äquivalenzklassen:

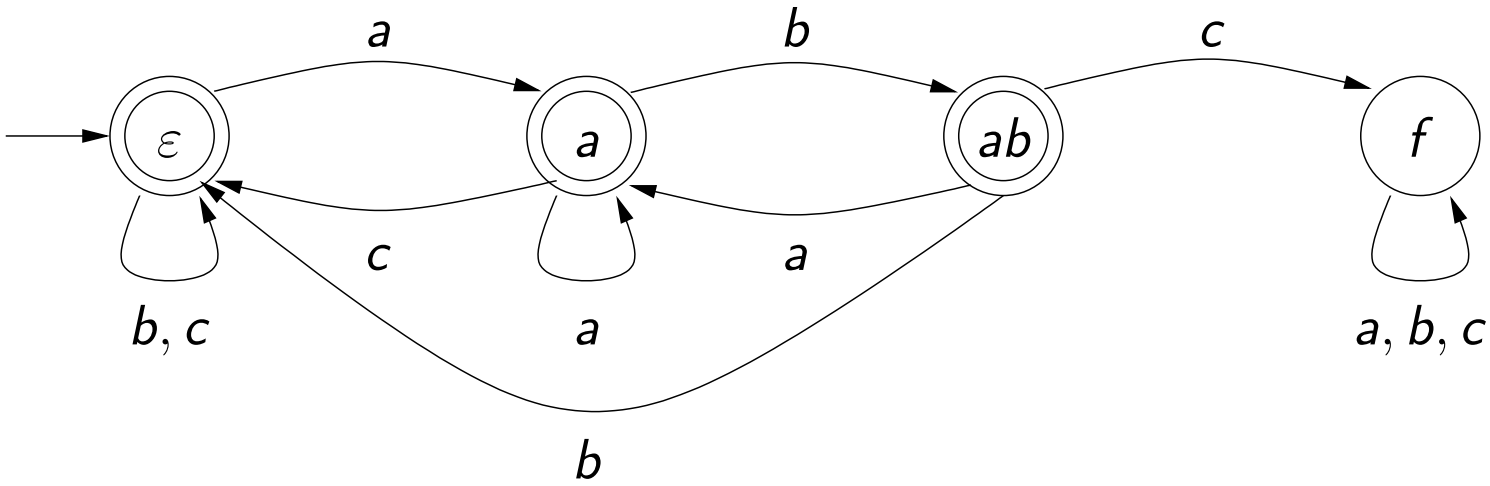
- $[\varepsilon] = \{w \in \{a, b, c\}^* \mid$
  w endet nicht auf a oder ab und enthält abc nicht$\}$
- $[a] = \{w \in \{a, b, c\}^* \mid$ w endet auf a und enthält abc nicht$\}$
- $[ab] = \{w \in \{a, b, c\}^* \mid$
  w endet auf ab und enthält abc nicht$\}$
- $[abc] = \{w \in \{a, b, c\}^* \mid$ w enthält $abc\}$ (Fangzustand)

Beispiel: a und ab sind nicht äquivalent, denn wird an beide ein c angehängt, so ist ac noch in der Sprache, abc ist es aber nicht.

Äquivalenzrelationen und Minimalautomat

$L = \{w \in \{a, b, c\}^* \mid$ das Teilwort abc kommt in w nicht vor$\}$

Automat:



Äquivalenzrelationen und Minimalautomat

Myhill-Nerode-Äquivalenz und Regularität (Satz)

Eine Sprache $L \subseteq \Sigma^*$ ist genau dann regulär, wenn R_L endlich viele Äquivalenzklassen hat.

Äquivalenzrelationen und Minimalautomat

R_L hat endlich viele Äquivalenzklassen $\Rightarrow$ L regulär:

Wir nehmen zunächst an, dass R_L endlich viele Äquivalenzklassen hat und konstruieren einen endlichen Automaten $M = (Z, \Sigma, \delta, z_0, E)$ für L, der wie folgt definiert ist:

$$
\begin{aligned}
Z &= \{[w]_{R_L} \mid w \in \Sigma^*\} \quad \text{(Menge der Äquivalenzklassen)} \\
z_0 &= [\varepsilon]_{R_L} \\
E &= \{[w]_{R_L} \mid w \in L\} \\
\delta([w]_{R_L}, a) &= [wa]_{R_L}
\end{aligned}
$$

Äquivalenzrelationen und Minimalautomat

L regulär $\Rightarrow$ R_L hat endlich viele Äquivalenzklassen:

Sei nun M ein DFA mit $T(M) = L$. Dann definieren wir eine Äquivalenzrelation R_M mit

$$x \, R_M \, y \iff \hat{\delta}(z_0, x) = \hat{\delta}(z_0, y) \quad \text{für } x, y \in \Sigma^*.$$

Die Anzahl der Äquivalenzklassen von R_M ist gleich der Anzahl der Zustände von M, d.h., sie ist endlich.

Äquivalenzrelationen und Minimalautomat

Man kann zeigen, dass aus $x\,R_M\,y$ immer $x\,R_L\,y$ folgt: dazu nehmen wir ein beliebiges $z \in \Sigma^*$. Dann gilt nämlich

$$xz \in L \iff \hat{\delta}(z_0, xz) \in E \iff \hat{\delta}(\hat{\delta}(z_0, x), z) \in E$$
$$\iff \hat{\delta}(\hat{\delta}(z_0, y), z) \in E \iff \hat{\delta}(z_0, yz) \in E \iff yz \in L.$$

Also setzt R_M höchstens so viel Elemente in Beziehung wie R_L und hat damit mehr (oder gleich viele) Äquivalenzklassen wie R_L. Daraus folgt aber, dass R_L nur endlich viele Äquivalenzklassen hat.

Äquivalenzrelationen und Minimalautomat

Man kann den obigen Satz dazu nutzen, um zu zeigen, dass ein Sprache nicht regulär ist. Dazu muss man nur unendlich viele Wörter aus Σ^* aufzählen und zeigen, dass sie in verschiedenen Äquivalenzklassen sind.

Beispiel 3 für Myhill-Nerode-Äquivalenz:

$$\text{Sprache } L = \{a^k b^k \mid k \geq 0\}$$

Betrachte die Wörter a, aa, aaa, $\ldots$, a^i, $\ldots$

Es gilt: $\neg(a^i \, R_L \, a^j)$ für $i \neq j$, denn $a^i b^i \in L$ und $a^j b^i \notin L$.

Kochrezept für Myhill-Nerode-Beweise

Um den Satz von Myhill-Nerode zu verwenden, um zu beweisen, dass eine Sprache L nicht regulär ist, geht man wie folgt vor:

- Identifiziere eine unendliche Klasse an Wörtern $w_1, w_2, \ldots$, die jeweils eine eigene Äquivalenzklasse repräsentieren.
- Zeige für alle i, j: $[w_i] = [w_j] \Rightarrow i = j$.

Es ist nicht notwendig, alle Äquivalenzklassen zu identifizieren oder für jedes Wort in Σ^* anzugeben, in welcher Äquivalenzklasse es liegt. Es reicht, unendlich viele Wörter zu identifizieren, die paarweise nicht äquivalent sind.

Äquivalenzrelationen und Minimalautomat

Wir verwenden Myhill-Nerode-Äquivalenz nun um zwei zuvor mit dem Pumping-Lemma untersuchte Sprachen zu analysieren:

Beispiel 4 für Myhill-Nerode-Äquivalenz (Adventure):

$$\text{Sprache } L = \{A^2 L^k T^m \mid k \geq m \geq 1\}$$

Betrachte die Wörter AA, AAL, $AALL$, $\ldots$, AAL^i, $\ldots$

Es gilt: $\neg(AAL^i \, R_L \, AAL^j)$ für $i \neq j$. O.B.d.A sei $i > j$, dann $AAL^i T^i \in L$ und $AAL^j T^i \notin L$.

Äquivalenzrelationen und Minimalautomat

Wir verwenden Myhill-Nerode-Äquivalenz nun um zwei zuvor mit dem Pumping-Lemma untersuchte Sprachen zu analysieren:

Beispiel 5 für Myhill-Nerode-Äquivalenz (nicht reguläre Sprache mit Pumping-Eigenschaft):

Sprache

$$L = \{a^k b^m c^m \mid k, m \geq 0\} \cup \{b^i c^j \mid i, j \geq 0\}.$$

Betrachte die Wörter a, ab, abb, $\ldots$, ab^i, $\ldots$

Es gilt: $\neg(ab^i \, R_L \, ab^j)$ für $i \neq j$. Sei $i \neq j$, dann $ab^i c^i \in L$ und $ab^j c^i \notin L$.

Think-Pair-Share: Nicht-Regularität

Betrachten Sie die (nicht-reguläre) Sprache

$$L = \{a^n b^m \mid n < m\}.$$

Zeigen Sie, wahlweise mit dem Pumping-Lemma (das ist in diesem Fall möglich) oder Myhill-Nerode-Äquivalenzklassen, dass L nicht regulär ist.
Erarbeiten Sie zunächst vier Minuten in Einzelarbeit eine Lösung. Anschließend tauschen Sie sich für weitere vier Minuten mit ihrem Sitznachbarn aus. Schlussendlich besprechen wir die Lösung im Plenum.

Fragestellungen zu dieser Vorlesungseinheit

Zum Ende einer jeden Vorlesungseinheit betrachten wir im Regelfall drei Fragestellungen, die mithilfe der in dieser Einheit besprochenen Inhalte beantwortet werden sollen. In der darauffolgenden Einheit können zu Beginn mögliche Antworten gesammelt werden.

Fragen zur sechsten Vorlesungseinheit

- Wie kann man mit Hilfe des Pumping Lemmas für reguläre Sprachen zeigen, dass eine Sprache nicht regulär ist?
- Wie kann man mit Hilfe der Myhill-Nerode-Äquivalenz zeigen, dass eine Sprache nicht regulär ist?
- Wie kann man mit Hilfe der Myhill-Nerode-Äquivalenz zeigen, dass eine Sprache regulär ist?

Äquivalenzrelationen und Minimalautomat

Der DFA, der aus den Äquivalenzklassen einer regulären Sprache L konstruiert werden kann, ist der (eindeutige) minimale deterministische Automat für L. Wie kann man ihn aus einem, nicht notwendigerweise minimalen, DFA erhalten, ohne die Äquivalenzklassen zu konstruieren?

Lösung: wir starten mit dem DFA und verschmelzen alle erkennungsäquivalenten Zustände.

Dabei legen wir zunächst fest, welche Zustände auf jeden Fall nicht erkennungsäquivalent sind (die Endzustände und Nicht-Endzustände) und finden weitere nicht erkennungsäquivalente Zustände.

Äquivalenzrelationen und Minimalautomat

Algorithmus Minimalautomat

Eingabe: DFA M (Zustände, die vom Startzustand aus nicht erreichbar sind, sind bereits entfernt)

Ausgabe: Mengen von erkennungsäquivalenten Zuständen

1. Stelle eine Tabelle aller Zustandspaare $\{z, z'\}$ mit $z \neq z'$ auf.
2. Markiere alle Paare $\{z, z'\}$ mit $z \in E$ und $z' \notin E$ (oder umgekehrt)
   (z, z' sind sicherlich nicht erkennungsäquivalent.)

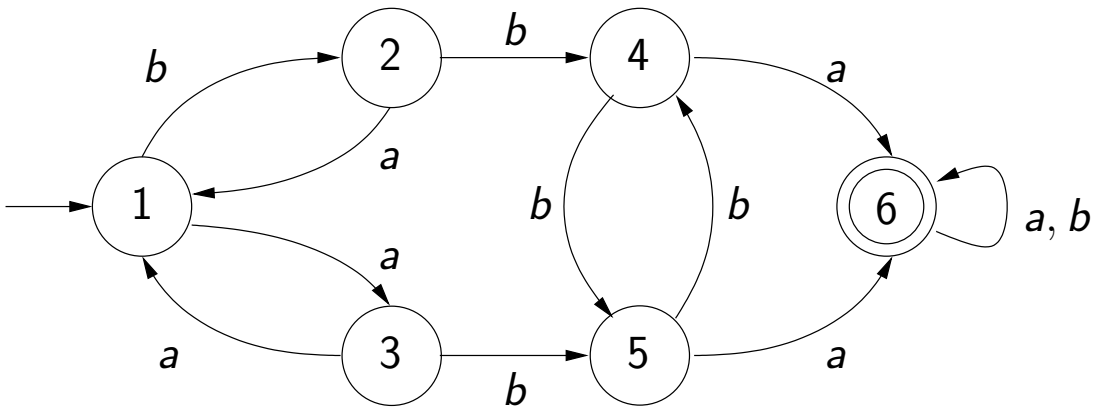
Äquivalenzrelationen und Minimalautomat

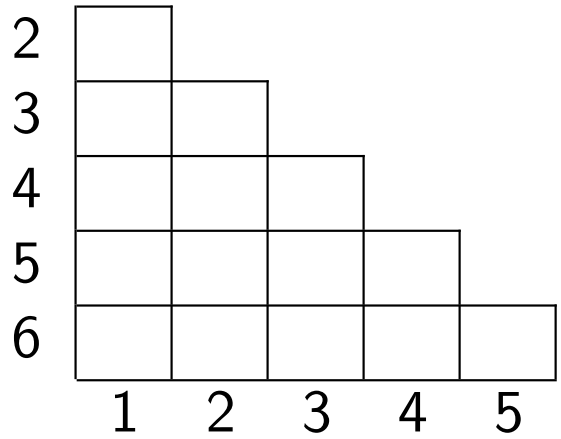
Algorithmus Minimalautomat

3. Für jedes noch unmarkierte Paar $\{z, z'\}$ und jedes $a \in \Sigma$ teste, ob $\{\delta(z, a), \delta(z', a)\}$ bereits markiert ist. Wenn ja: markiere auch $\{z, z'\}$.
   (Von z, z' gibt es Übergänge zu nicht erkennungsäquivalenten Zuständen, sie können daher nicht erkennungsäquivalent sein.)
4. Wiederhole den vorherigen Schritt, bis sich keine Änderung in der Tabelle mehr ergibt.
5. Für alle jetzt noch unmarkierten Paare $\{z, z'\}$ gilt: z und z' sind erkennungsäquivalent.

Äquivalenzrelationen und Minimalautomat

Durchführung des Minimierungs-Algorithmus am Beispiel des folgenden Automaten:

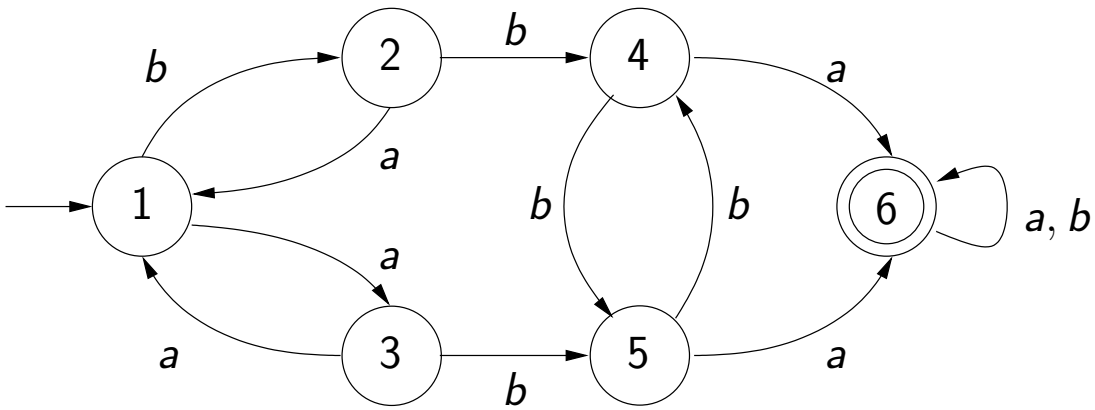




Erstelle eine Tabelle aller Zustandspaare

Äquivalenzrelationen und Minimalautomat

Durchführung des Minimierungs-Algorithmus am Beispiel des folgenden Automaten:

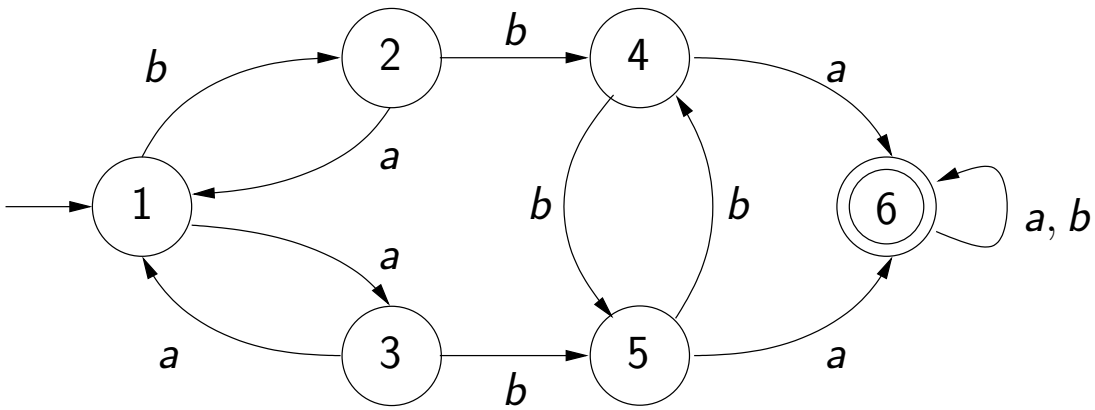


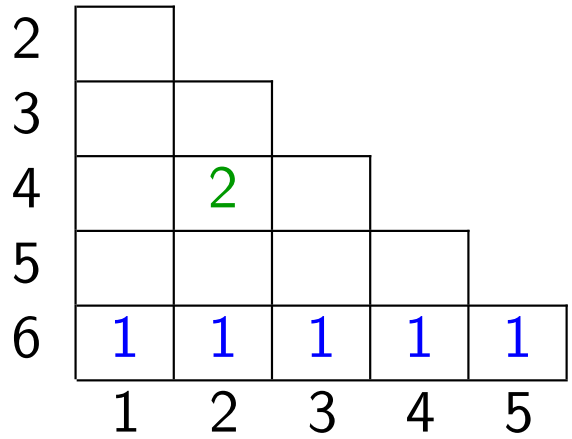
2					
3					
4					
5					
6	1	1	1	1	1
	1	2	3	4	5

(1) Markiere Paare von Endzuständen und Nicht-Endzuständen

Äquivalenzrelationen und Minimalautomat

Durchführung des Minimierungs-Algorithmus am Beispiel des folgenden Automaten:

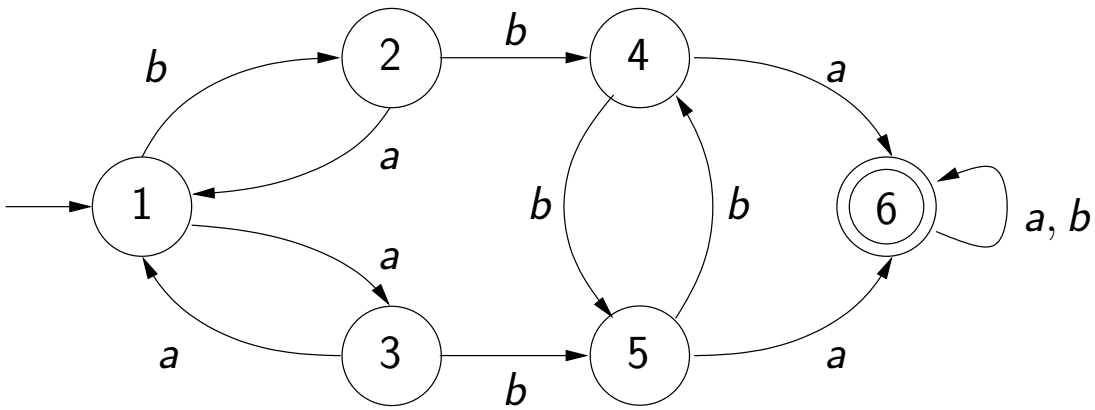


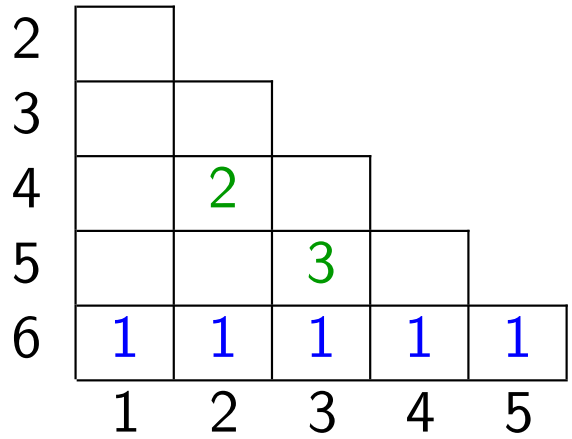


(2) Markiere $\{2,4\}$ wegen $\delta(2,a)=1$, $\delta(4,a)=6$ und $\{1,6\}$ markiert

Äquivalenzrelationen und Minimalautomat

Durchführung des Minimierungs-Algorithmus am Beispiel des folgenden Automaten:

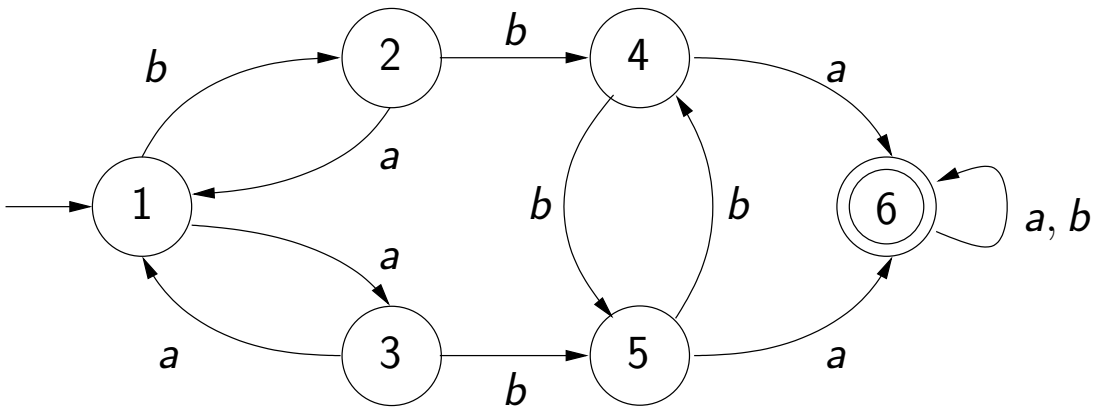




(3) Markiere $\{3, 5\}$ wegen $\delta(3, a) = 1$, $\delta(5, a) = 6$ und $\{1, 6\}$ markiert

Äquivalenzrelationen und Minimalautomat

Durchführung des Minimierungs-Algorithmus am Beispiel des folgenden Automaten:

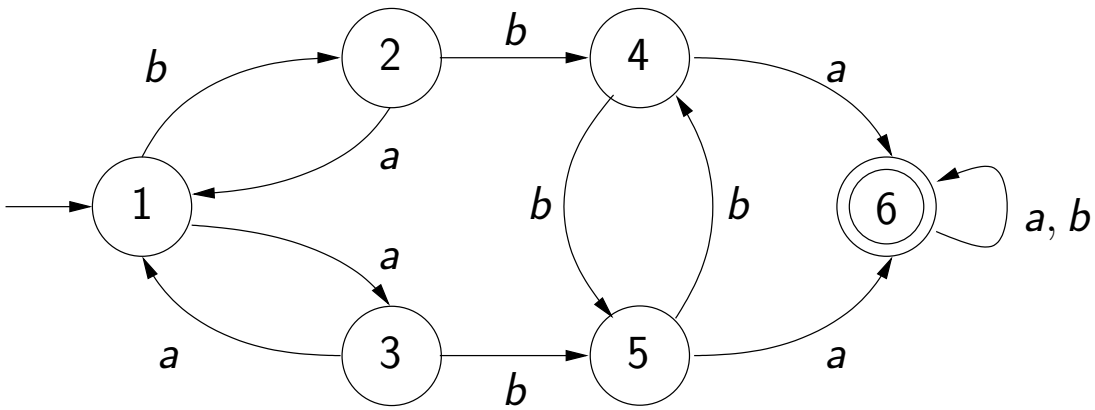


2					
3					
4		2			
5		4	3		
6	1	1	1	1	1
	1	2	3	4	5

(4) Markiere $\{2,5\}$ wegen $\delta(2,a)=1$, $\delta(5,a)=6$ und $\{1,6\}$ markiert

Äquivalenzrelationen und Minimalautomat

Durchführung des Minimierungs-Algorithmus am Beispiel des folgenden Automaten:

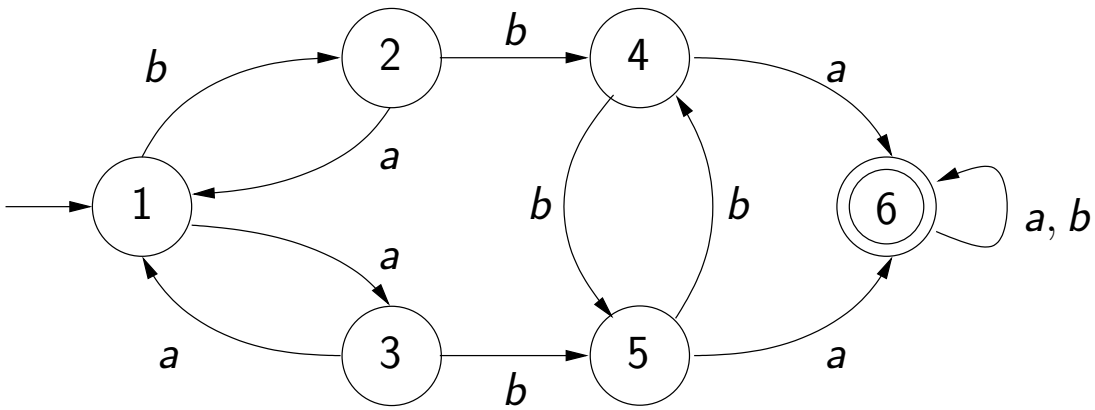


2					
3					
4		2	5		
5		4	3		
6	1	1	1	1	1
	1	2	3	4	5

(5) Markiere $\{3,4\}$ wegen $\delta(3,a) = 1$, $\delta(4,a) = 6$ und $\{1,6\}$ markiert

Äquivalenzrelationen und Minimalautomat

Durchführung des Minimierungs-Algorithmus am Beispiel des folgenden Automaten:

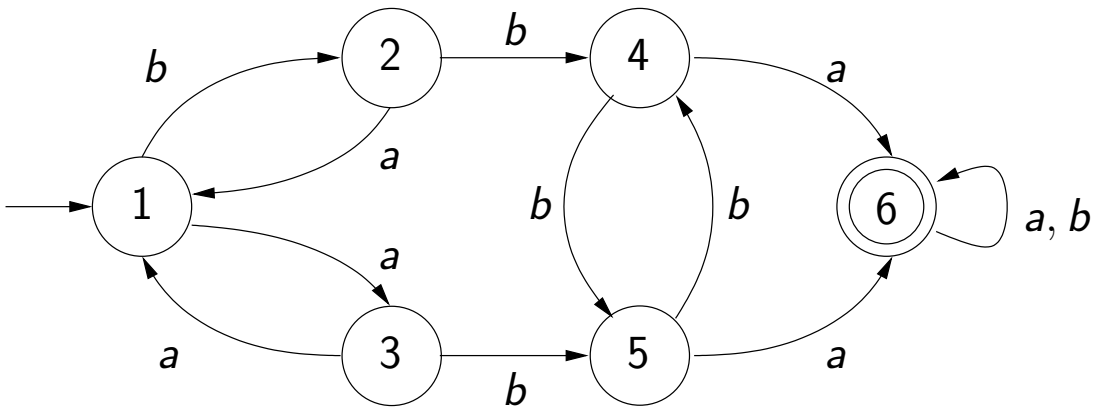


2					
3					
4		2	5		
5	6	4	3		
6	1	1	1	1	1
	1	2	3	4	5

(6) Markiere $\{1, 5\}$ wegen $\delta(1, a) = 3$, $\delta(5, a) = 6$ und $\{3, 6\}$ markiert

Äquivalenzrelationen und Minimalautomat

Durchführung des Minimierungs-Algorithmus am Beispiel des folgenden Automaten:

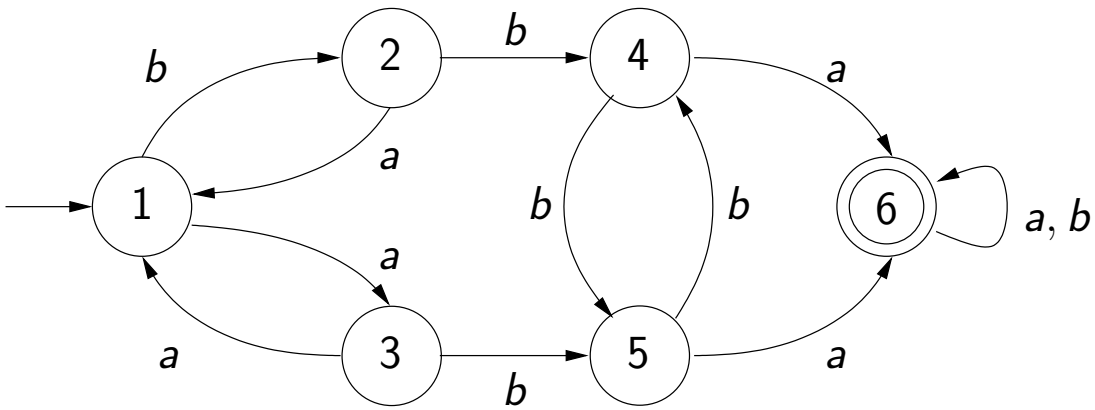


2					
3					
4	7	2	5		
5	6	4	3		
6	1	1	1	1	1
	1	2	3	4	5

(7) Markiere $\{1, 4\}$ wegen $\delta(1, a) = 3$, $\delta(4, a) = 6$ und $\{3, 6\}$ markiert

Äquivalenzrelationen und Minimalautomat

Durchführung des Minimierungs-Algorithmus am Beispiel des folgenden Automaten:

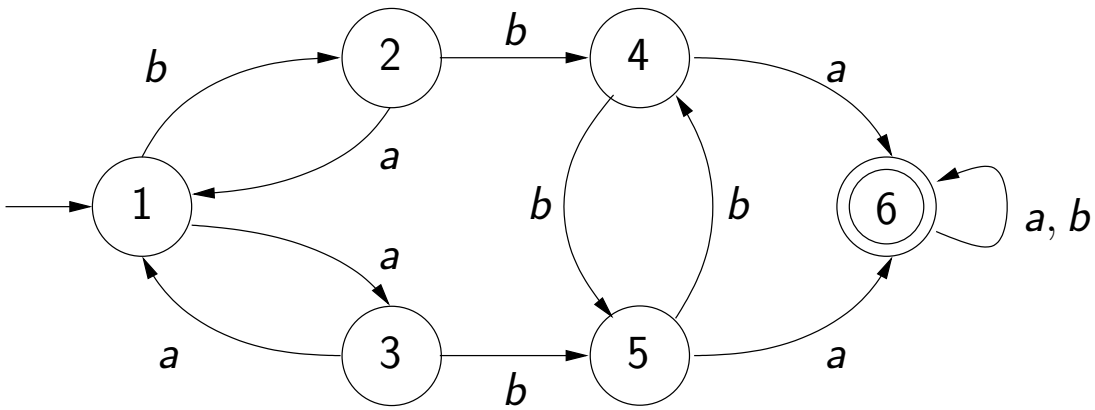


2					
3	8				
4	7	2	5		
5	6	4	3		
6	1	1	1	1	1
	1	2	3	4	5

(8) Markiere $\{1, 3\}$ wegen $\delta(1, b) = 2$, $\delta(3, b) = 5$ und $\{2, 5\}$ markiert

Äquivalenzrelationen und Minimalautomat

Durchführung des Minimierungs-Algorithmus am Beispiel des folgenden Automaten:

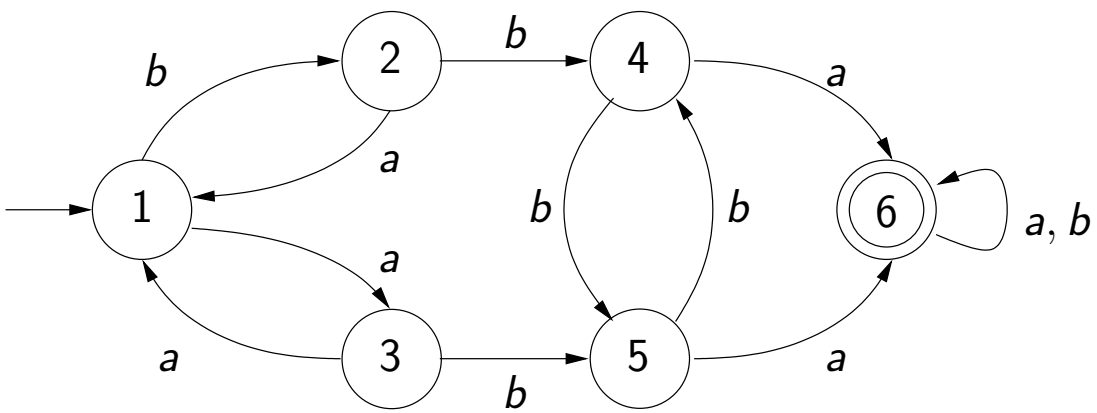


2	9				
3	8				
4	7	2	5		
5	6	4	3		
6	1	1	1	1	1
	1	2	3	4	5

(9) Markiere $\{1, 2\}$ wegen $\delta(1, b) = 2$, $\delta(2, b) = 4$ und $\{2, 4\}$ markiert

Äquivalenzrelationen und Minimalautomat

Durchführung des Minimierungs-Algorithmus am Beispiel des folgenden Automaten:



2	9				
3	8				
4	7	2	5		
5	6	4	3		
6	1	1	1	1	1
	1	2	3	4	5

Die verbleibenden Zustandspaare $\{2, 3\}$ und $\{4, 5\}$ können nicht mehr markiert werden $\rightsquigarrow$ sie sind erkennungsäquivalent und können verschmolzen werden.

Äquivalenzrelationen und Minimalautomat

Hinweise für die Durchführung des Minimierungs-Algorithmus:

- Die Tabelle möglichst so aufstellen, dass jedes Paar nur genau einmal vorkommt! Also bei Zustandsmenge $\{1, \ldots, n\}$:

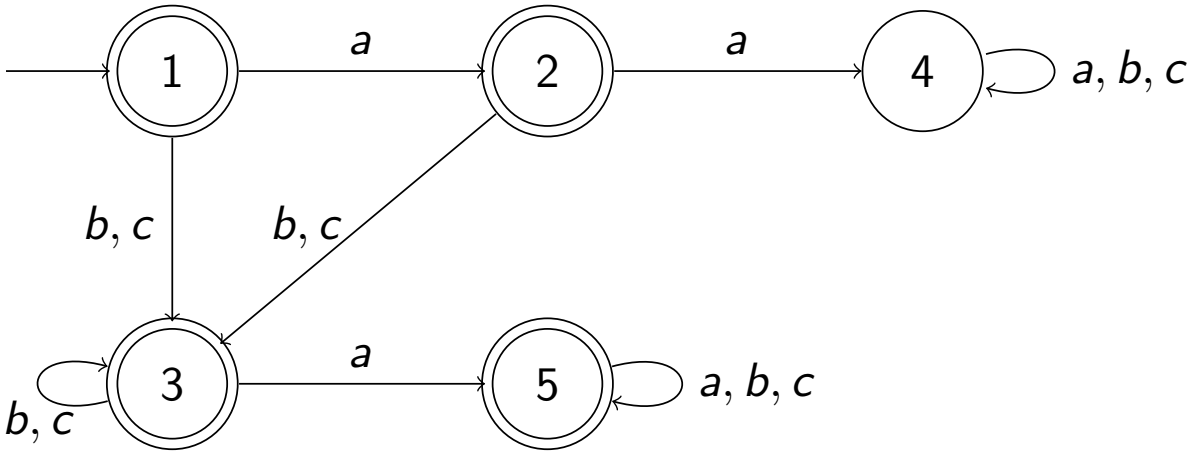
  $2, \ldots, n$ vertikal und $1, \ldots, n-1$ horizontal notieren.

- Bitte angeben, welche Zustände in welcher Reihenfolge und warum markiert wurden!

  (Im Buch von Schöning werden nur Sternchen ($*$) verwendet, aber daraus werden bei der Korrektur die Reihenfolge und die Gründe für die Markierung nicht ersichtlich.)

Think-Pair-Share: Minimalautomat

Minimieren Sie den folgenden Automaten über dem Alphabet $\{a, b, c\}$. Geben Sie außerdem die Myhill-Nerode-Äquivalenzrelation auf dem Automaten an. Beachten Sie, dass Sie hierzu das Ergebnis der Minimierung verwenden können.



Erarbeiten Sie zunächst sechs Minuten in Einzelarbeit eine Lösung. Anschließend tauschen Sie sich für weitere sechs Minuten mit ihrem Sitznachbarn aus. Schlussendlich besprechen wir die Lösung im Plenum.

Lösung der Think-Pair-Share-Aufgabe

2	2			
3	3	2		
4	1	1	1	
5	3	2		1
	1	2	3	4

Demzufolge ist die Myhill-Nerode-Äquivalenzrelation durch die folgenden Äquivalenzklassen gegeben:

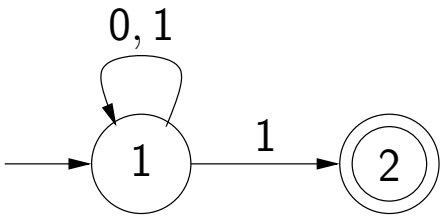
$$[\varepsilon] = \{1\}, [a] = \{2\}, [b] = \{3, 5\}, [aa] = \{4\}.$$

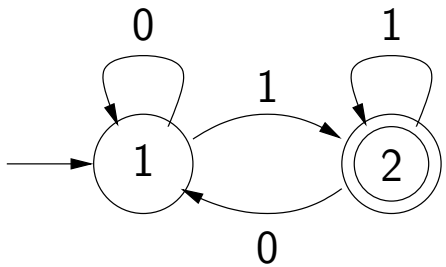
Äquivalenzrelationen und Minimalautomat

Für nicht-deterministische Automaten kann man folgende Aussagen treffen:

- Es gibt nicht *den* minimalen NFA, sondern es kann mehrere geben.

  Folgende zwei minimale NFAs erkennen $L((0|1)^*1)$ und haben zwei Zustände. (Mit nur einem Zustand kann diese Sprache nicht erkannt werden.)





Äquivalenzrelationen und Minimalautomat

- Gegeben ein DFA M. Dann hat ein minimaler NFA, der $T(M)$ erkennt, immer höchstens so viel Zustände wie M. (Denn M selbst ist schon ein NFA.)

  Außerdem: der minimale NFA kann exponentiell kleiner sein als der minimale DFA.

  Siehe Beispielsprachen:

  $L_k = \{x \in \{0,1\}^* \mid |x| \geq k, \text{das } k\text{-letzte Zeichen von } x \text{ ist } 0\}$.

Entscheidbarkeit

Wir diskutieren nun, ob es Verfahren gibt, um die folgenden Fragestellungen bzw. Probleme für reguläre Sprachen zu entscheiden. Dabei nehmen wir an, dass reguläre Sprachen als DFAs, NFAs, Grammatiken oder reguläre Ausdrücke gegeben sind.

Probleme

- Wortproblem: Gegeben eine reguläre Sprache L und $w \in \Sigma^*$. Gilt $w \in L$?
- Leerheitsproblem: Gegeben eine reguläre Sprache L. Gilt $L = \emptyset$?
- Endlichkeitsproblem: Gegeben eine reguläre Sprache L. Ist L endlich?

Entscheidbarkeit

Probleme (Fortsetzung)

- Schnittproblem: Gegeben zwei reguläre Sprachen L_1, L_2. Gilt $L_1 \cap L_2 = \emptyset$?
- Inklusionsproblem: Gegeben zwei reguläre Sprachen L_1, L_2. Gilt $L_1 \subseteq L_2$?
- Äquivalenzproblem: Gegeben zwei reguläre Sprachen L_1, L_2. Gilt $L_1 = L_2$?

Entscheidbarkeit

Wortproblem ($w \in L$?)

Gegeben sind eine reguläre Sprache L und ein Wort $w \in \Sigma^*$.

Lösung: Bestimme einen DFA M für L und verfolge die Zustandsübergänge von M, wie durch w vorgegeben.
Endzustand wird erreicht $\rightsquigarrow w \in L$
Nicht-Endzustand wird erreicht $\rightsquigarrow w \notin L$

Entscheidbarkeit

Leerheitsproblem ($L = \emptyset$?)

Gegeben ist eine reguläre Sprache L.

Lösung: Bestimme einen NFA M für L.

$L = \emptyset$

$\iff$ es gibt keinen Pfad von einem Start- zu einem Endzustand.

Entscheidbarkeit

Endlichkeitsproblem (Ist L endlich?)

Gegeben ist eine reguläre Sprache L.

Lösung: Bestimme einen NFA M für L.
L ist unendlich
$\iff$ in M gibt es unendlich viele Pfade von einem Start- zu einem Endzustand
$\iff$ es gibt einen erreichbaren Zyklus in M, von dem aus wiederum ein Endzustand erreichbar ist.

Entscheidbarkeit

Schnittproblem ($L_1 \cap L_2 = \emptyset?$)

Gegeben sind reguläre Sprachen L_1, L_2.

Lösung: Bestimme NFAs M_1, M_2 für L_1, L_2 und bilde das Kreuzprodukt von M_1, M_2. Wende dann den Leerheitstest auf das Kreuzprodukt an.
(Siehe auch den Abschnitt über Abschlusseigenschaften
▸ Schnitt regulärer Sprachen
.)

Entscheidbarkeit

Inklusionsproblem ($L_1 \subseteq L_2$?)

Gegeben sind reguläre Sprachen L_1, L_2.

Lösung: Es gilt $L_1 \subseteq L_2$ genau dann, wenn $L_1 \cap \overline{L_2} = \emptyset$. Da Schnitt und Komplement konstruktiv bestimmbar sind und ein Leerheitstest existiert, kann damit das Inklusionsproblem gelöst werden.

Anmerkung: für dieses Problem gibt es auch effizientere Methoden, bei denen die Komplementierung von L_2 – für die die Konstruktion eines deterministischen Automaten erforderlich ist – vermieden wird.

Entscheidbarkeit

Äquivalenzproblem ($L_1 = L_2$?)

Gegeben sind reguläre Sprachen L_1, L_2.

Lösung: Es gilt $L_1 = L_2$ genau dann, wenn $L_1 \subseteq L_2$ und $L_1 \supseteq L_2$. Das Inklusionsproblem ist – wie wir vorher gesehen haben – lösbar.

Eine andere Methode: Bestimme jeweils zu L_1 und L_2 die minimalen DFAs M_1 und M_2. Da der minimale DFA eindeutig ist, muss jetzt nur noch gezeigt werden, dass M_1 und M_2 strukturell gleich sind, d.h., es gibt eine Umbenennung der Zustände, die M_1 in M_2 überführt. Man sagt auch: M_1 und M_2 sind isomorph.

Entscheidbarkeit

Effizienzgesichtspunkte:

Je nachdem, in welcher Darstellung eine Sprache L gegeben ist, kann die Komplexität der oben beschriebenen Verfahren sehr unterschiedlich ausfallen.

Beispiel Äquivalenzproblem:

- L_1, L_2 gegeben als DFAs $\rightsquigarrow$ Komplexität $O(n^2)$
  (d.h., quadratisch viele Schritte in der Größe der Eingabe)
- L_1, L_2 gegeben als Grammatiken, reguläre Ausdrücke oder NFAs $\rightsquigarrow$ Komplexität NP-hart
  Das bedeutet unter anderem: es ist nicht bekannt, ob dieses Problem in polynomieller Zeit lösbar ist. (Mehr zur Komplexitätsklasse NP und verwandten Fragestellungen in der Vorlesung "Berechenbarkeit und Komplexität".)

Anwendung: Verifikation

Mit Hilfe von Sprachen bzw. den dazugehörigen endlichen Automaten kann man oft alle Abläufe eines Systems beschreiben (zumindest falls das System nur endlich viele Zustände hat).

Sei also L_{Sys} die Menge aller Systemabläufe und L_{Spec} die Spezifikation, d.h., die Menge aller korrekten Systemabläufe. Wir wollen zeigen, dass

$$L_{Sys} \subseteq L_{Spec}$$

Das kann man mit den eingeführten Verfahren machen, wenn beide Sprachen durch Automaten gegeben sind. Diesen Vorgang nennt man auch (System-)Verifikation.

Anwendung: Verifikation

Ein System kann dabei ein Programm, ein Prozess oder ein verteiltes System (bestehend aus mehreren Prozessen) sein.

Beispiele für Verifikation:

- Zeige, dass alle Pfade eines Adventures mindestens einen Schatz enthalten.
- Zeige, dass in einem Programm niemals eine Division durch 0 auftritt.
- Zeige, dass in einem System von nebenläufig arbeitenden Prozessen der wechselseitige Ausschluss nicht verletzt wird.

Anwendung: Verifikation

Ein abschließendes ausführliches Beispiel:

- Wir betrachten zwei Prozesse P_1, P_2, die auf eine gemeinsame Ressource (Drucker, Datei, ...) zugreifen wollen.
- Jeder Prozess hat einen sogenannten kritischen Bereich, in dem auf die Ressource zugegriffen wird. Es darf sich jeweils nur ein Prozess im kritischen Bereich befinden.
- Es stehen gemeinsame Variable zur Verfügung, über die sich die Prozesse synchronisieren können. Diese Variablen sind jedoch keine Semaphoren, d.h., eine atomare Operation, bei der gleichzeitig gelesen und geschrieben wird, ist nicht möglich.

Wir möchten zeigen, dass der wechselseitige Ausschluss gewährleistet ist und dass gewisse Fairnessbedingungen (jeder Prozess kommt irgendwann an die Reihe) eingehalten werden.

Anwendung: Verifikation

Was hat das mit formalen Sprachen zu tun?

- Die Menge aller Abläufe der Prozesse wird durch einen endlichen Automaten beschrieben. Insbesondere gibt es Automaten für jeden Prozess und einen Automaten für die Abläufe des Gesamtsystems, der durch ein Kreuzprodukt erzeugt wird.
- Wir möchten zeigen, dass ein System *Sys* eine Spezifikation *Spec* erfüllt. Sei L_{Sys} die Menge aller möglichen Abläufe des Systems und L_{Spec} die Menge aller Abläufe, die *Spec* erfüllen.

  Dann ist zu zeigen: $L_{Sys} \subseteq L_{Spec}$.

  Und wenn beide Sprachen regulär sind, dann gibt es dafür ein effektives Verfahren!

Anwendung: Verifikation

Versuch 1: Beide Prozesse P_1, P_2 verwenden eine gemeinsame Boolesche Variable `f`, die mit `false` initialisiert wird.

Programmcode für P_1, P_2

```
  while true do
1:  if (f = false?) then do
      begin
2:      f := true;
3:      [Betrete krit. Bereich];
4:      [Verlasse krit. Bereich];
5:      f := false
      end
    endif
  enddo
```

Anwendung: Verifikation

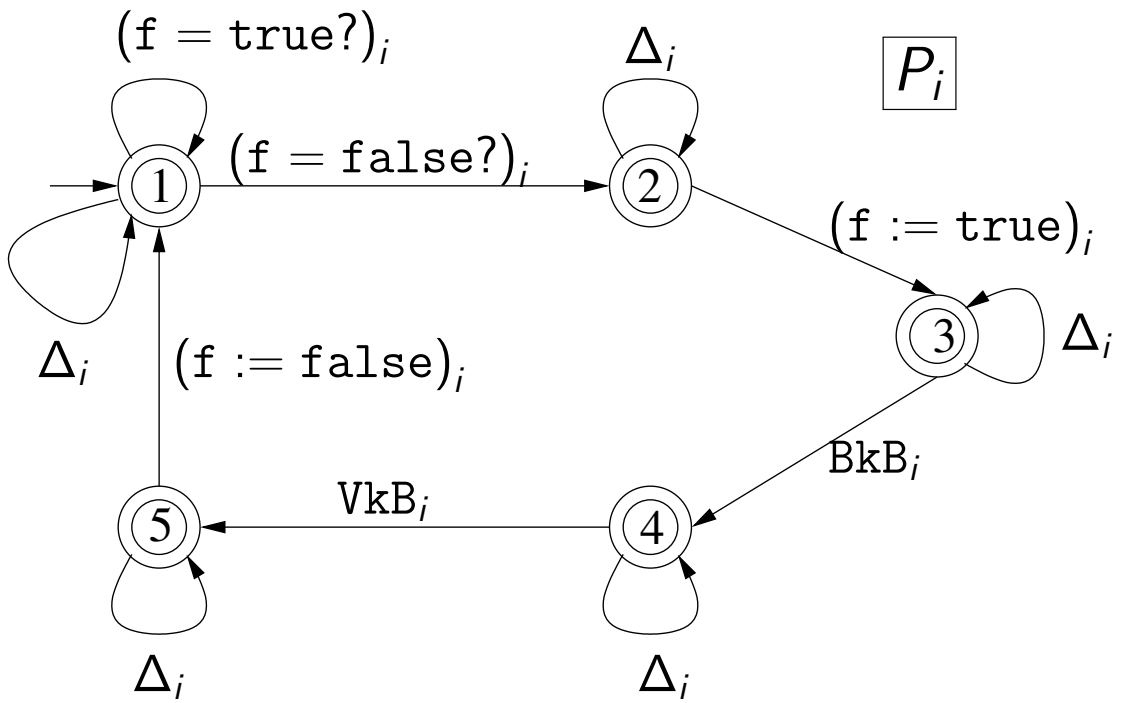
Wir verwenden folgendes Alphabet, bestehend aus den Programm-Befehlen und den Abfragen der Booleschen Variablen:

$$
\begin{aligned}
\Sigma \quad = \quad & \{(\mathtt{f} := \mathtt{true})_i, (\mathtt{f} := \mathtt{false})_i, (\mathtt{f} = \mathtt{true?})_i, \\
& (\mathtt{f} = \mathtt{false?})_i \mid i \in \{1,2\}\} \\
& \qquad \text{(Synchronisation von Prozess } i \text{ mit Variable } \mathtt{f}\text{)} \\
\cup \quad & \{\mathtt{BkB}_i, \mathtt{VkB}_i \mid i \in \{1,2\}\} \\
& \qquad \text{(Prozess } i \text{ betritt/verlässt kritischen Bereich).}
\end{aligned}
$$

Der Index $i \in \{1,2\}$ gibt an, ob die jeweilige Aktion vom ersten oder vom zweiten Prozess ausgeführt wird.

Anwendung: Verifikation

Beschreibung der Abläufe eines Prozesses i als endlicher Automat:



mit $\Delta_i =$
$\{(\texttt{f} := \texttt{true})_j, (\texttt{f} := \texttt{false})_j, (\texttt{f} = \texttt{true?})_j, (\texttt{f} = \texttt{false?})_j, \texttt{BkB}_j, \texttt{VkB}_j\}$
wobei $j = 2$, falls $i = 1$, und $j = 1$, falls $i = 2$.

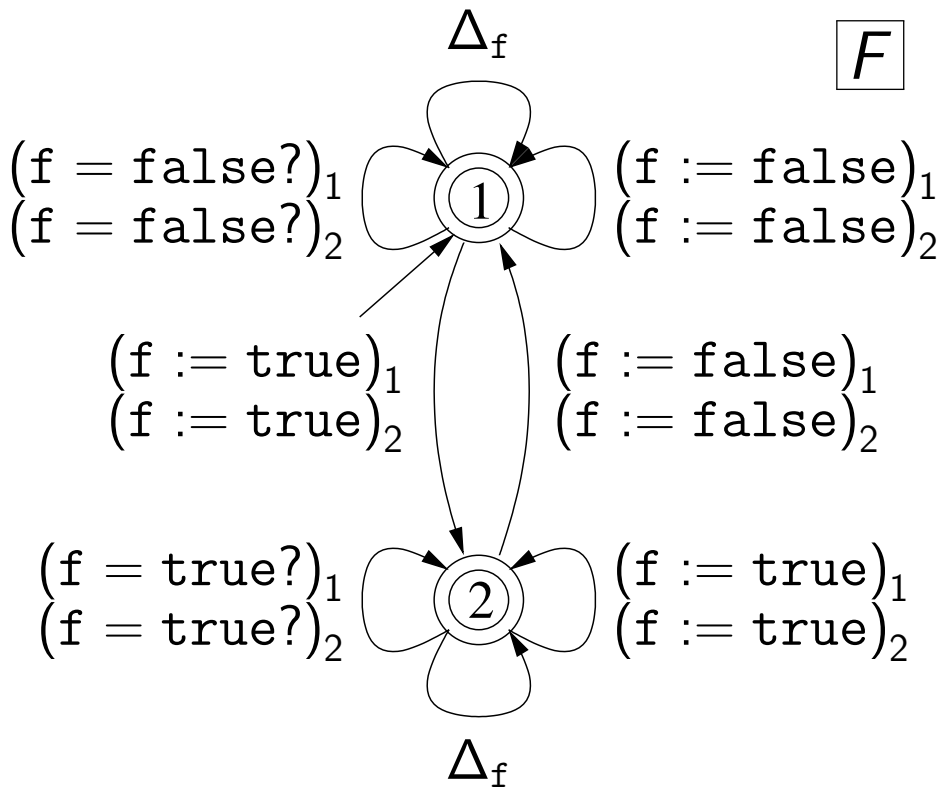
Anwendung: Verifikation

Bemerkungen:

- Bedeutung der Zustände 1, 2, 3, 4, 5: diese entsprechen den mit Labels markierten Programmzeilen
- Bedeutung der Schleifen mit Alphabetsymbolen aus Δ_i: da wir später das Kreuzprodukt bilden werden, um mehrere Automaten zu synchronisieren, dürfen Übergänge anderer Automaten, die den Prozess i nicht betreffen, nicht ausgeschlossen werden. Sie werden einfach "mitgehört" und haben keinen Einfluss auf die Zustandsübergänge.

Anwendung: Verifikation

Beschreibung der Booleschen Variable f durch einen Automaten:

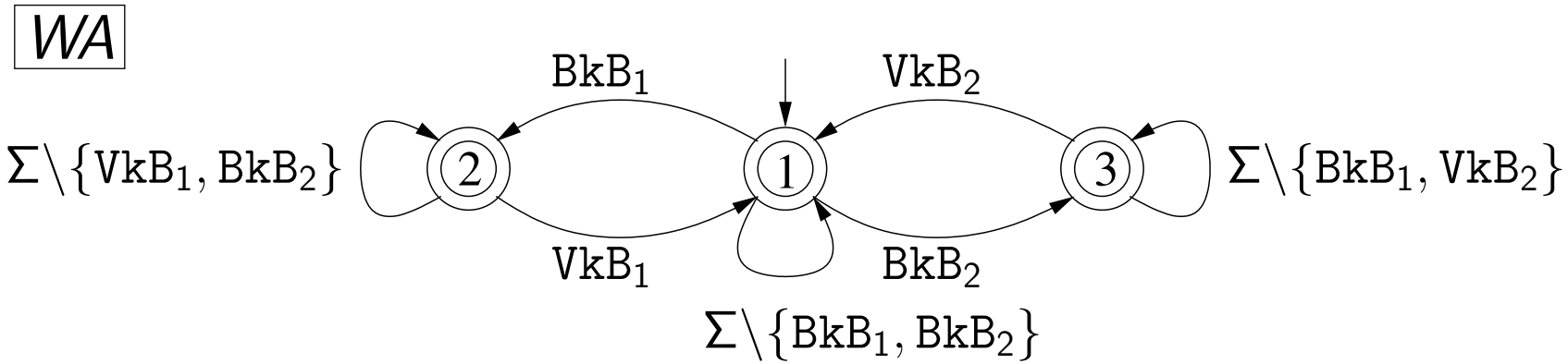


mit $\Delta_{\mathtt{f}} = \{\mathtt{BkB}_1, \mathtt{VkB}_1, \mathtt{BkB}_2, \mathtt{VkB}_2\}$.

Anwendung: Verifikation

Die Sprache aller Abläufe des Gesamtsystems ist $T(P_1) \cap T(P_2) \cap T(F)$.

Der Automat WA, der alle Abläufe beschreibt, die den wechselseitigen Ausschluss erfüllen (beide Prozesse sind nicht gleichzeitig im kritischen Bereich) sieht folgendermaßen aus:



Damit ist zu zeigen $T(P_1) \cap T(P_2) \cap T(F) \subseteq T(WA)$.

Anwendung: Verifikation

Strategie : Kreuzprodukt der Automaten P_1, P_2, F bilden; Automat WA komplementieren und dann wiederum das Kreuzprodukt bilden; die Sprachen sind ineinander enthalten, genau dann, wenn der entstehende Automat die leere Sprache akzeptiert.

Anwendung: Verifikation

Die entstehende Sprache ist nicht leer! Es gibt also Abläufe, die die Bedingung des wechselseitigen Ausschluss verletzen.

Einer davon ist:

$$(\texttt{f} = \texttt{false?})_2\,(\texttt{f} = \texttt{false?})_1\,(\texttt{f} := \texttt{true})_2\,\texttt{BkB}_2\,(\texttt{f} := \texttt{true})_1\,\texttt{BkB}_1.$$

Grund für die Verletzung des wechselseitigen Ausschlusses: Es gibt keine atomare Schreib- und Leseoperation. Daher können beide Prozesse nacheinander die Variable auslesen, anschließend setzen beide die Variable und betreten den kritischen Bereich.

Der Algorithmus ist also falsch!

Anwendung: Verifikation

Versuch 2: Wir betrachten nun das Verfahren zum wechselseitigen Ausschluss von Lamport.

Dabei betrachten wir: zwei Prozesse P_1, P_2 mit unterschiedlichem Programmcode und zwei Boolesche Variable `f1`, `f2` (initialisiert mit `false`).

Anwendung: Verifikation

Prozess P_1

```
  while true do
1:  f1 := true;  (#)
2:  while (f2 = true?) do
      skip
    od;
3:  [Betrete krit. Bereich];
4:  [Verlasse krit. Bereich];
5:  f1 := false
  od;
```

`skip`: Null-Operation (hat keine Auswirkungen)

Anwendung: Verifikation

Prozess P_2

```
  while true do
1:  f2 := true;  (#)
2:  if (f1 = true?) then do
      begin
3:      f2 := false;
4:      while (f1 = true?) do skip od;
        goto 1
      end;
5:  [Betrete krit. Bereich];
6:  [Verlasse krit. Bereich];
7:  f2 := false
  od;
```

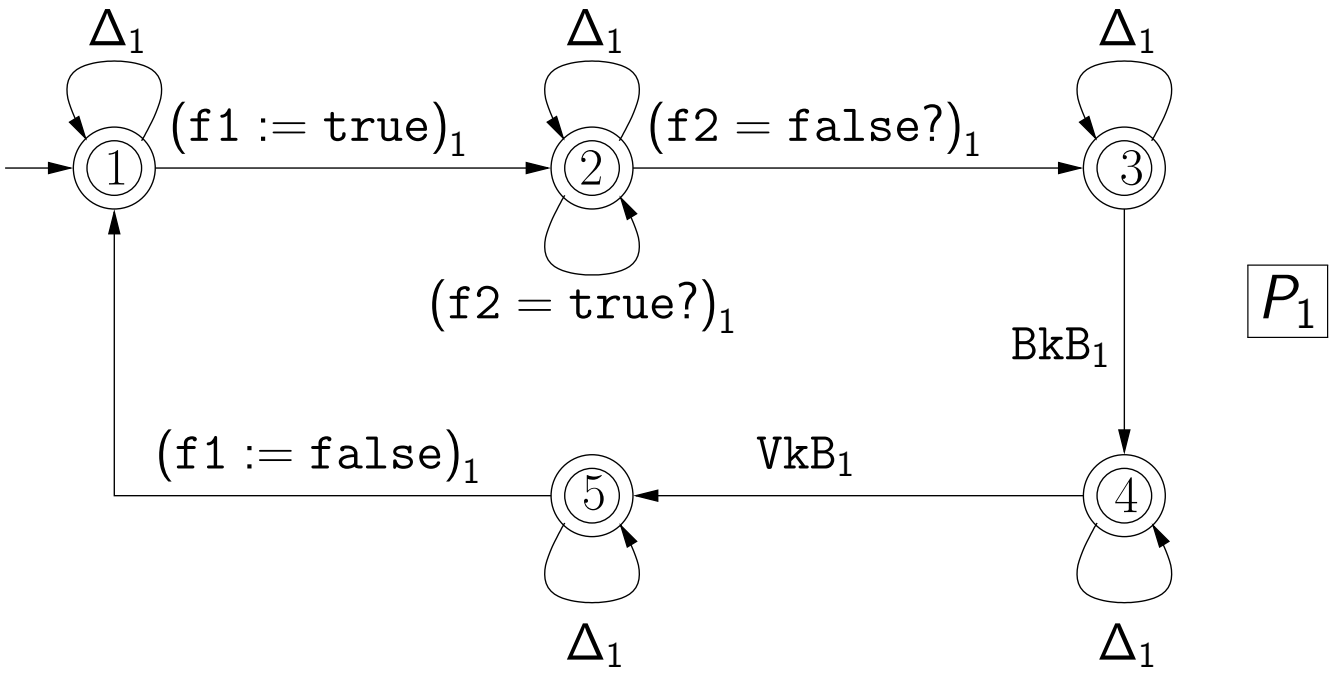
Anwendung: Verifikation

In diesem Fall betrachten wir folgendes Alphabet Σ:

$$\begin{aligned}\Sigma \quad = \quad & \{(\texttt{f1} := \texttt{true})_1, (\texttt{f1} := \texttt{false})_1, \\ & (\texttt{f1} = \texttt{true?})_2, (\texttt{f1} = \texttt{false?})_2, \\ & (\texttt{f2} := \texttt{true})_2, (\texttt{f2} := \texttt{false})_2, \\ & (\texttt{f2} = \texttt{true?})_1, (\texttt{f2} = \texttt{false?})_1, \\ & \texttt{BkB}_1, \texttt{VkB}_1, \texttt{BkB}_2, \texttt{VkB}_2\}.\end{aligned}$$

Anwendung: Verifikation

Automat für den Prozess P_1:

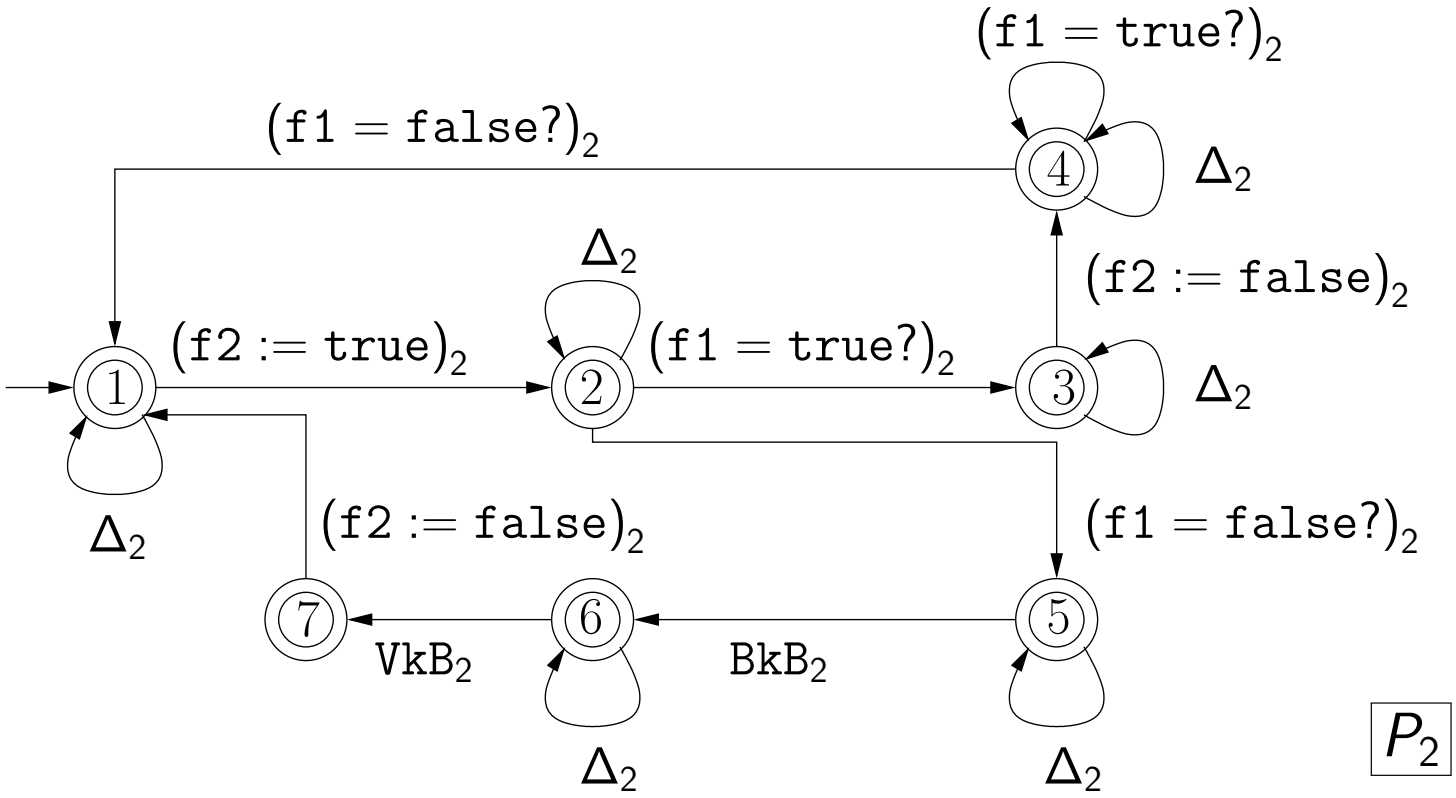


Dabei gilt für die "mitgehörten" Alphabetsymbole:

$$\begin{aligned}\Delta_1 \quad = \quad & \{(\texttt{f2} := \texttt{true})_2, (\texttt{f2} := \texttt{false})_2, (\texttt{f1} = \texttt{true?})_2, \\ & (\texttt{f1} = \texttt{false?})_2, \texttt{BkB}_2, \texttt{VkB}_2\}\end{aligned}$$

Anwendung: Verifikation

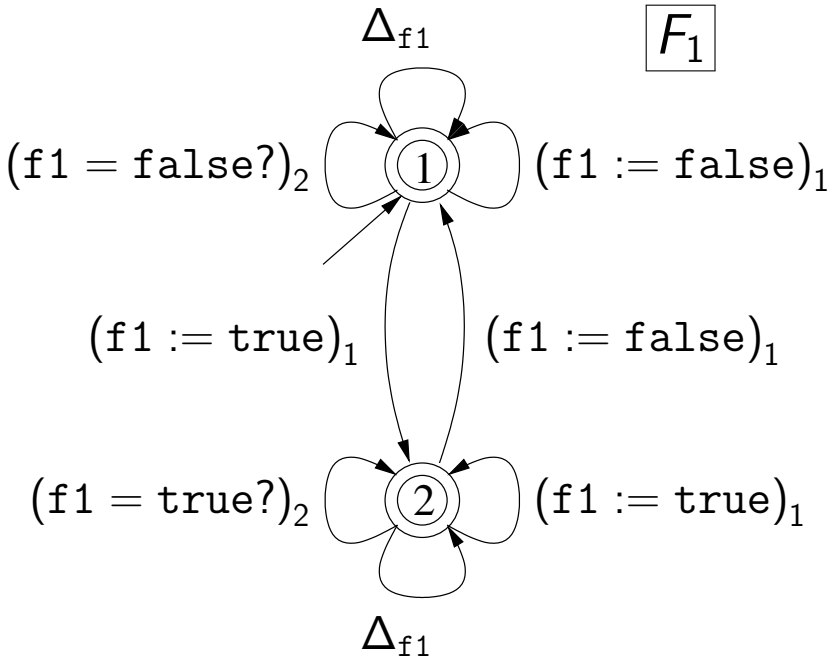
Automat für den Prozess P_2:

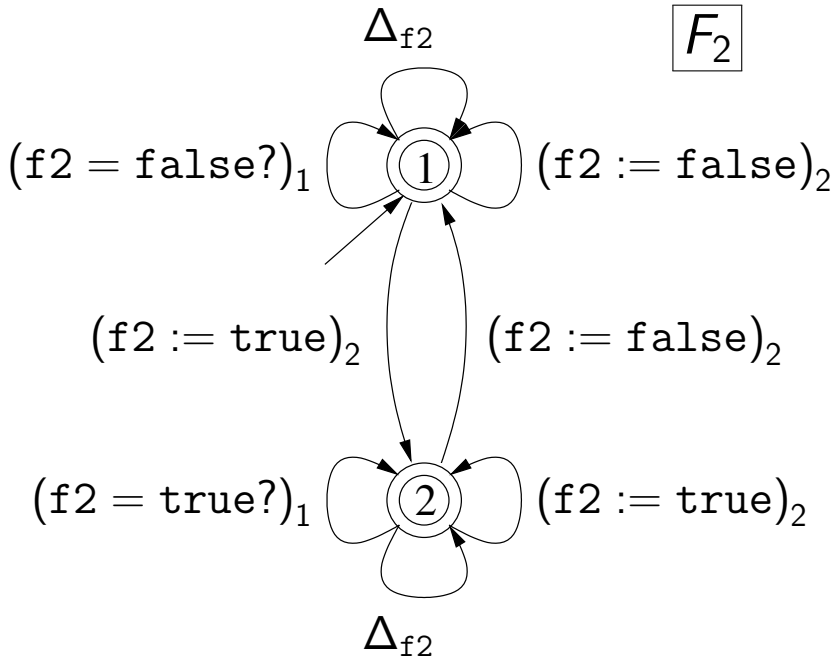


$$\Delta_2 = \{(\texttt{f1} := \texttt{true})_1, (\texttt{f1} := \texttt{false})_2, (\texttt{f2} = \texttt{true?})_1, (\texttt{f2} = \texttt{false?})_1, \texttt{BkB}_1, \texttt{VkB}_1\}$$

Anwendung: Verifikation

Automaten für die beiden Variablen:





$$\begin{aligned}\Delta_{\texttt{f1}} \quad = \quad & \{(\texttt{f2} := \texttt{true})_2, (\texttt{f2} := \texttt{false})_2, (\texttt{f2} = \texttt{true?})_1, \\ & (\texttt{f2} = \texttt{false?})_1, \texttt{BkB}_1, \texttt{BkB}_2, \texttt{VkB}_1, \texttt{VkB}_2\}\end{aligned}$$

Analog für $\Delta_{\texttt{f2}}$.

Anwendung: Verifikation

In diesem Fall ist der wechselseitige Ausschluss erfüllt, d.h., es gilt $T(P_1) \cap T(P_2) \cap T(F_1) \cap T(F_2) \subseteq T(WA)$.

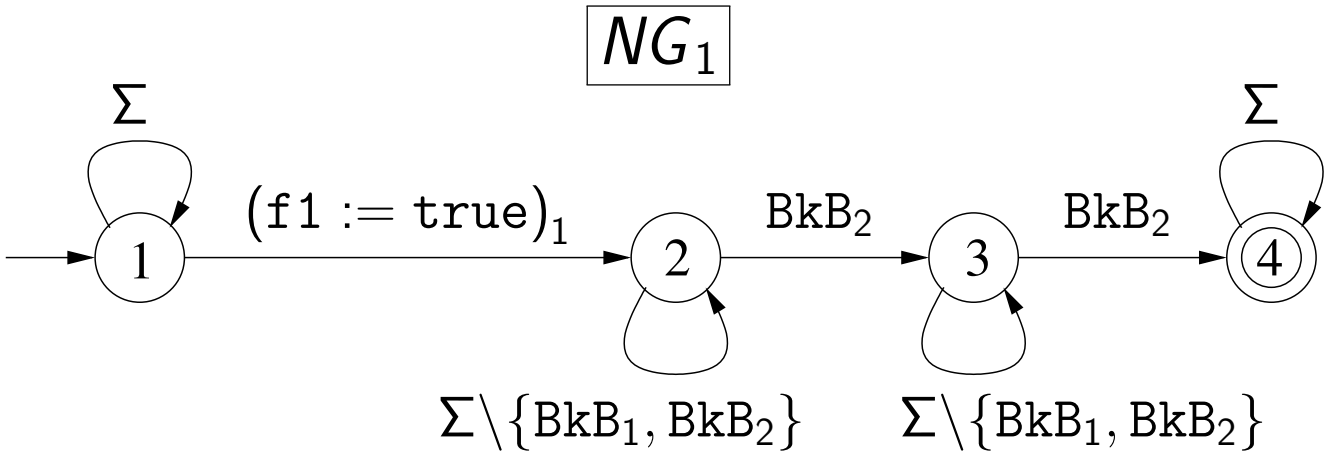
Anwendung: Verifikation

Neben dem wechselseitigen Ausschluss soll noch folgende Fairness-Bedingung für jeden Prozess i überprüft werden:

(G_i) "Sobald Prozess i seine Bereitschaft bekundet hat, den kritischen Bereich zu betreten, indem er die Anweisung (#) ausführt, kann der andere Prozess j nicht zweimal hintereinander den kritischen Bereich betreten, ohne dass Prozess i zwischendurch den kritischen Bereich betritt."

Anwendung: Verifikation

Automat NG_1, der genau die Abläufe erkennt, die (G_1) nicht erfüllen:



Wir wollen zeigen, dass
$T(P_1) \cap T(P_2) \cap T(F_1) \cap T(F_2) \cap T(NG_1) = \emptyset$ gilt.

Anwendung: Verifikation

Fairness ist erfüllt für Prozess 1

Anwendung: Verifikation

Fairness ist *nicht* erfüllt für Prozess 2:

Der erste Ablauf, der die Fairness verletzt, entspricht:

$$(\texttt{f2} := \texttt{true})_2 \quad (\texttt{f1} := \texttt{true})_1 \quad (\texttt{f1} = \texttt{true?})_2 \quad (\texttt{f2} := \texttt{false})_2$$
$$(\texttt{f2} = \texttt{false?})_1 \quad \texttt{BkB}_1 \quad \texttt{VkB}_1 \quad (\texttt{f1} := \texttt{false})_1 \quad (\texttt{f1} := \texttt{true})_1$$
$$(\texttt{f2} = \texttt{false?})_1 \quad \texttt{BkB}_1$$

Anwendung: Verifikation

Zusammenfassung:

- Wir haben mit Hilfe von endlichen Automaten zwei Protokolle modelliert, die wechselseitigen Ausschluss realisieren sollen.
- Mit Hilfe der Lösungsverfahren für das Inklusions- bzw. Schnittproblem haben wir überprüft, ob diese Protokolle tatsächlich wechselseitigen Ausschluss und Fairness realisieren.

Das bedeutet: die vorgestellten Verfahren können zur Programmverifikation eingesetzt werden.

Bemerkung: Bei realen Programmen hat man allerdings noch damit zu kämpfen, dass der Zustandsraum eines Programms oft unendlich ist. Damit wird vieles unentscheidbar und muss durch approximative Verfahren gelöst werden.

Ausblick

Im Laufe der verbleibenden Vorlesungseinheiten werden wir uns den kontextfreien Sprachen zuwenden:

- Wir werden ein effizienteres Verfahren zur Entscheidung des Wortproblems kennenlernen.
- Analog zum Pumping-Lemma für reguläre Sprachen betrachten wir das Pumping-Lemma für kontextfreie Sprachen, mit dem gezeigt werden kann, dass eine Sprache nicht kontextfrei ist.
- Wir untersuchen, unter welchen Operatoren kontextfreie Sprachen abgeschlossen sind.
- Analog zu DFAs und NFAs werden wir Sprachakzeptoren für kontextfreie Sprachen definieren, die so genannten Kellerautomaten.

Fragestellungen zu dieser Vorlesungseinheit

Zum Ende einer jeden Vorlesungseinheit betrachten wir im Regelfall drei Fragestellungen, die mithilfe der in dieser Einheit besprochenen Inhalte beantwortet werden sollen. In der darauffolgenden Einheit können zu Beginn mögliche Antworten gesammelt werden.

Fragen zur siebten Vorlesungseinheit

- Wie kann man einen DFA minimieren?
- Wie stellt man fest ob die Sprache, die ein NFA akzeptiert, leer ist?
- Wie stellt man fest ob zwei NFAs die gleiche Sprache akzeptieren?

Kontextfreie Sprachen

Wir behandeln nun die kontextfreien oder Typ-2-Sprachen.

Wiederholung: Produktionen kontextfreier Grammatiken

Bei kontextfreien Grammatiken haben alle Produktionen die Form $A \to w$, wobei $A \in V$ (d.h., A ist eine Variable) und $w \in (V \cup \Sigma)^*$.

Betrachtete Beispielgrammatiken:

- Grammatik, die korrekt geklammerte arithmetische Ausdrücke erzeugt
- Grammatik, die Sätze der natürlichen Sprache erzeugt

Ein weiteres Beispiel: die Sprache $L = \{a^k b^k \mid k \geq 0\}$ ist kontextfrei.
Produktionen: $S \to \varepsilon \mid T$, $T \to ab \mid aTb$

Kontextfreie Sprachen

Anwendungen kontextfreier Sprachen

Hauptanwendung: Beschreibung der Syntax von Programmiersprachen

Viele der hier besprochenen Techniken sind daher interessant für den Einsatz im Compilerbau.

Bemerkung: Bisher ist es noch niemandem gelungen eine vollständige Grammatik aller korrekten natürlichsprachigen Sätze zu bilden. Frage: Was ist überhaupt ein korrekter Satz?

Kontextfreie Sprachen

Es ist unter Linguisten umstritten, ob es eine kontextfreie Grammatik geben kann, die eine natürliche Sprache erzeugt. Ein Gegenargument beispielsweise für schweizer Deutsch ist, dass es Verben gibt, die zwei Nominalphrasen verschiedenen Typs (in diesem Fall: Akkusativobjekte und Dativobjekte) benötigen und eine Satzordnung möglich ist, in der beispielsweise alle Akkusativobjekte gruppiert vor alle Dativobjekte und diese wiederum vor alle zugehörigen Verben gestellt werden.

Kontextfreie Sprachen

Inhalt des Abschnitts "Kontextfreie Sprachen"

- Normalformen – wichtig für die Anwendung bestimmter Verfahren/Techniken ist es, eine Grammatik in eine bestimmte Normalform zu bringen
- Wortproblem – ein Algorithmus, um das Wortproblem zu lösen (CYK-Algorithmus)
- Pumping-Lemma für kontextfreie Sprachen
- Abschlusseigenschaften – die kontextfreien Sprachen verhalten sich hier nicht ganz so gutartig wie die regulären Sprachen
- Kellerautomaten – das Automatenmodell zu kontextfreien Sprachen

Normalformen

Wir beschäftigen uns zunächst noch einmal mit der "ε-Sonderregelung":

Die Definition für kontextfreie Grammatiken (mit ε-Sonderregelung) fordert, dass S auf keiner rechten Seite auftauchen darf, wenn $S \to \varepsilon$ als Produktion vorkommt. Außerdem dürfen keine weiteren Produktionen der Form $A \to \varepsilon$ auftauchen.

Was passiert, wenn man diese Bedingungen für kontextfreie Grammatiken aufhebt und beliebige Regeln der Form $A \to \varepsilon$ erlaubt? Kann es dann passieren, dass man eine nicht-kontextfreie Sprache erzeugt?

Antwort: nein

Normalformen

ε-freie Grammatiken (Satz)

Gegeben sei eine Grammatik $G = (V, \Sigma, P, S)$ mit Produktionen der Form $A \to w$, $w \in (V \cup \Sigma)^*$ und $\varepsilon \notin L(G)$.
Dann gibt es eine Grammatik $G' = (V, \Sigma, P, S)$ mit Produktionen der Form $A \to w$, $w \in (V \cup \Sigma)^+$ und $L(G) = L(G')$.

Das bedeutet, dass Grammatiken, bei denen auf der linken Seite einer Produktionsregel stets genau eine Variable steht, auch wenn sie nicht kontextsensitiv sind (also ε-Ableitungen, die der ε-Sonderregel nicht genügen, enthalten), stets kontextfreie Sprachen erzeugen.

Normalformen

Verfahren zur Entfernung von ε-Produktionen:

1. Bestimme die Variablenmenge $V_1 \subseteq V$ mit $V_1 = \{A \in V \mid A \Rightarrow^* \varepsilon\}$, d.h., die Menge aller Variablen, aus denen sich das leere Wort ableiten lässt.
2. Füge für jede Produktion der Form $B \to xAy$ mit $A \in V_1$, $x, y \in (V \cup \Sigma)^*$ eine Produktion $B \to xy$ zur Produktionenmenge hinzu. (Diese Produktion "simuliert" das Löschen von A.)
   Wiederhole diesen Schritt solange, bis keine neuen Regeln mehr entstehen. (Achtung: für die rechte Seite einer Produktion gibt es evtl. mehrere Möglichkeiten, sie in xAy aufzuspalten.)
3. Entferne alle Produktionen der Form $A \to \varepsilon$.

Normalformen

Beispiel: ε-Produktionen entfernen
Sei $G = (V, \Sigma, P, S)$, wobei $V = \{S, X, Y, Z\}$, $\Sigma = \{a, b\}$ und P enthält folgende Produktionen:

$$S \to XZ$$
$$X \to aYb \mid \varepsilon$$
$$Y \to bXa \mid bb$$
$$Z \to \varepsilon \mid aSa$$

Bemerkung: Für diese Grammatik G gilt $\varepsilon \in L(G)$. Durch die Umwandlung entsteht eine Grammatik G' mit $L(G') = L(G) \backslash \{\varepsilon\}$.

Normalformen

Bemerkung:

Weil wir jede Grammatik, die "fast" kontextfrei ist, aber das leere Wort als rechte Seite enthält, in eine kontextfreie Grammatik umwandeln können, werden wir im Folgenden, um zu zeigen, dass eine Sprache kontextfrei ist, Grammatiken verwenden, bei denen beliebige Wörter als rechte Seiten zugelassen sind, auch das leere Wort.

Manchmal ist es in Konstruktionen und Beweisen trotzdem praktisch davon auszugehen, dass ε nicht als rechte Seite vorkommt (außer als $S \to \varepsilon$, siehe ε-Sonderregel). Daher gilt weiterhin: Ist eine *Grammatik* kontextfrei, bedeutet das, dass alle ε-Ableitungen der ε-Sonderregel genügen.

Normalformen

Wir betrachten nun eine weitere nützliche Normalform.

Chomsky-Normalform (Definition)

Eine kontextfreie Grammatik G mit $\varepsilon \notin L(G)$ ist in Chomsky-Normalform (kurz: CNF), falls alle Produktionen eine der folgenden zwei Formen haben:

$$A \to BC \qquad A \to a$$

Dabei sind $A, B, C \in V$ Variablen und $a \in \Sigma$ ein Alphabetsymbol.

Normalformen

Umwandlung in Chomsky-Normalform (Satz)

Zu jeder kontextfreien Grammatik G mit $\varepsilon \notin L(G)$ gibt es eine Grammatik G' in Chomsky-Normalform mit $L(G) = L(G')$.

Die Chomsky-Normalform ist besonders nützlich, weil Ableitungen in solchen Grammatiken die Form eines Binärbaums annehmen. Wir werden dies verwenden, um das Wortproblem für kontextfreie Sprachen effizient zu beantworten und um zu beweisen, dass eine Sprache nicht kontextfrei ist.

Normalformen

Verfahren zur Umwandlung in Chomsky-Normalform:

0. (Falls die Grammatik nicht kontextfrei, ist aber jede Produktionsregel auf der linken Seite nur eine Variable enthält: ε-Produktionen entfernen ▸ ε-Produktionen entfernen)
1. Kettenproduktionen entfernen ($A \to B$)
2. Alphabetsymbole aus den rechten Seiten entfernen
3. Lange rechte Seiten aufteilen

Normalformen

Verfahren zur Umwandlung in Chomsky-Normalform:

1. Entferne alle Kettenproduktionen der Form $A \to B$. Hierfür unterscheidet man zwei Fälle:

   1. Fall: Eine Kettenproduktion liegt auf einem Zyklus $A_1 \to A_2 \to \cdots \to A_k \to A_1$ von Produktionen. In diesem Fall werden alle Variablen $A_1, \ldots, A_k$ durch eine einzige Variable A ersetzt und die Kettenproduktionen entfernt. Für jede Produktion $A_i \to w$, $1 \leq i \leq k$, $w \notin \{A_1, \ldots A_k\}$ fügen wir eine Produktion $A \to w$ hinzu und jedes Vorkommen eines A_i, $1 \leq i \leq k$ auf einer rechten Seite wird durch A ersetzt.

Normalformen

2. Fall: Es existiert kein Zyklus. In diesem Fall kann man die Variablen durchnummerieren: $A_1, \ldots, A_k$, so dass $A_i \to A_j$ nur gilt, falls $i < j$ (topologische Sortierung). Man geht nun von den höheren zu den niedrigeren Indizes ($i = k-1, \ldots, 1$) und ersetzt $A_i \to A_j$ durch

$$A_i \to x_1 \mid \cdots \mid x_n,$$

falls die Regeln mit A_j auf der linken Seite folgende Form haben:

$$A_j \to x_1 \mid \cdots \mid x_n$$

(Einführen von "Shortcuts")

Normalformen

2. Falls eine Regel $A \to w$ Terminalzeichen in w enthält und $|w| > 1$ gilt, so wird jedes Terminalzeichen a in w durch eine neue Variable U_a ersetzt. Außerdem werden Produktionen $U_a \to a$ hinzugefügt. Dadurch befinden sich nur noch Variablen auf der rechten Seite.
3. Im letzten Schritt werden Produktionen der Form $A \to B_1 \dots B_k$ eliminiert: führe neue Variable $C_1, \dots, C_{k-2}$ ein, entferne die ursprüngliche Regel und ersetze sie durch:

$$\begin{aligned} A &\to B_1 C_1 \\ C_1 &\to B_2 C_2 \\ &\vdots \\ C_{k-2} &\to B_{k-1} B_k \end{aligned}$$

Normalformen

Beispiel: Wir wandeln folgende Grammatik G in Chomsky-Normalform um. Dazu muss sie zunächst ε-frei gemacht werden.

$$G = (\{S, A\}, \{a, b, c\}, P, S)$$

mit folgender Produktionenmenge P:

$$\begin{aligned} S &\to aAb \\ A &\to S \mid aaSc \mid \varepsilon \end{aligned}$$

Normalformen

Entfernung von ε-Produktionen:

$$G = (\{S, A\}, \{a, b, c\}, P, S)$$

mit folgender Produktionenmenge P:

$$\begin{aligned} S &\rightarrow aAb \mid ab \\ A &\rightarrow S \mid aaSc \end{aligned}$$

Normalformen

Entfernung von Kettenproduktionen:

$$G = (\{S, A\}, \{a, b, c\}, P, S)$$

mit folgender Produktionenmenge P:

$$\begin{aligned} S &\rightarrow aAb \mid ab \\ A &\rightarrow aAb \mid ab \mid aaSc \end{aligned}$$

Normalformen

Entfernung von Terminalsymbolen aus den rechten Seiten:

$$G = (\{S, A, U_a, U_b, U_c\}, \{a, b, c\}, P, S)$$

mit folgender Produktionenmenge P:

$$\begin{aligned} S &\to U_a A U_b \mid U_a U_b \\ A &\to U_a A U_b \mid U_a U_b \mid U_a U_a S U_c \\ U_a &\to a \\ U_b &\to b \\ U_c &\to c \end{aligned}$$

Normalformen

Zu lange rechte Seiten aufspalten:

$$G = (\{S, A, U_a, U_b, U_c, C_1, C_2, C_3\}, \{a, b, c\}, P, S)$$

mit folgender Produktionenmenge P:

$$\begin{aligned}
S &\to U_a C_1 \mid U_a U_b \\
A &\to U_a C_1 \mid U_a U_b \mid U_a C_2 \\
U_a &\to a \\
U_b &\to b \\
U_c &\to c \\
C_1 &\to A U_b \\
C_2 &\to U_a C_3 \\
C_3 &\to S U_c
\end{aligned}$$

Think-Pair-Share: Chomsky-Normalform

Wandeln Sie Schritt für Schritt die folgende Grammatik in die Chomsky-Normalform um:

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

mit folgender Produktionenmenge P:

$$\begin{aligned} S &\to AB \mid B \\ A &\to ab \mid aAb \\ B &\to c \mid cB \end{aligned}$$

Erarbeiten Sie zunächst vier Minuten in Einzelarbeit eine Lösung. Anschließend tauschen Sie sich für weitere vier Minuten mit ihrem Sitznachbarn aus. Schlussendlich besprechen wir die Lösung im Plenum.

Lösungsvorschlag zu Think-Pair-Share: Chomsky-Normalform

$$G = (\{S, A, B, U_a, U_b, U_c, C_1\}, \{a, b, c\}, P, S)$$

mit folgender Produktionenmenge P:

$$\begin{aligned}
S &\to AB \mid c \mid U_c B \\
A &\to U_a U_b \mid U_a C_1 \\
B &\to c \mid U_c B \\
U_a &\to a \\
U_b &\to b \\
U_c &\to c \\
C_1 &\to A U_b
\end{aligned}$$

Der CYK-Algorithmus

Wir kennen bereits ein Verfahren, mit dem man das Wortproblem für G lösen kann, wobei G eine Typ-1-, Typ-2- oder Typ-3-Grammatik sein kann. (Im Wesentlichen: Aufzählen aller Wörter bis zu einer bestimmten Länge.)

Da dieses Verfahren jedoch exponentielle Laufzeit (in der Länge des Wortes) haben kann, betrachten wir hier ein anderes Verfahren für kontextfreie Grammatiken: den CYK-Algorithmus (entwickelt von Cocke, Younger, Kasami).

Voraussetzung: die Grammatik ist in Chomsky-Normalform gegeben. (Alle Regeln haben die Form $A \to a$ oder $A \to BC$.)

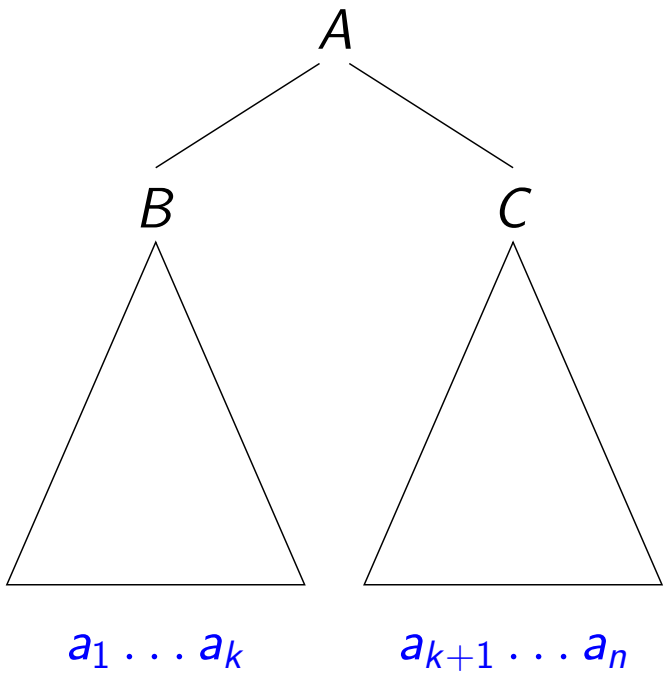
Der CYK-Algorithmus

Idee: Gegeben sei ein Wort $x \in \Sigma^*$. Wir wollen feststellen, aus welchen Variablen es abgeleitet worden sein könnte.

- Möglichkeit 1: $x = a \in \Sigma$, d.h., x besteht aus einem einzigen Alphabetsymbol. Dann kann w nur aus Variablen A abgeleitet worden sein, für die es eine Produktion $A \to a$ gibt.
- Möglichkeit 2: $x = a_1 \dots a_n$ mit $n \geq 2$. In diesem Fall gilt: Zunächst muss eine Produktion $A \to BC$ angewandt werden, dann muss ein Teil $a_1 \dots a_k$ des Wortes aus B und der andere Teil $a_{k+1} \dots a_n$ aus C abgeleitet werden. $(1 \leq k < n)$

Der CYK-Algorithmus

Möglichkeit 2 lässt sich schematisch folgendermaßen darstellen:



Der CYK-Algorithmus

Es ist jedoch nicht klar, wo das Wort x geteilt werden muss, d.h., wie groß der Index k ist!

Daher: Probiere alle möglichen k's durch. Das heißt:

Gegeben ein Wort $x = a_1 \dots a_n$. Überprüfe für alle k mit $1 \leq k < n$:

- Bestimme alle Variablen V_1, aus denen sich $a_1 \dots a_k$ ableiten lässt.
- Bestimme alle Variablen V_2, aus denen sich $a_{k+1} \dots a_n$ ableiten lässt.
- Stelle fest, ob es Variablen A, B, C gibt mit $(A \to BC) \in P$, $B \in V_1$ und $C \in V_2$. In diesem Fall gilt, dass sich x aus A ableiten lässt.

Der CYK-Algorithmus

Um Mehraufwand zu vermeiden: verwende Methoden der dynamischen Programmierung, das heißt:

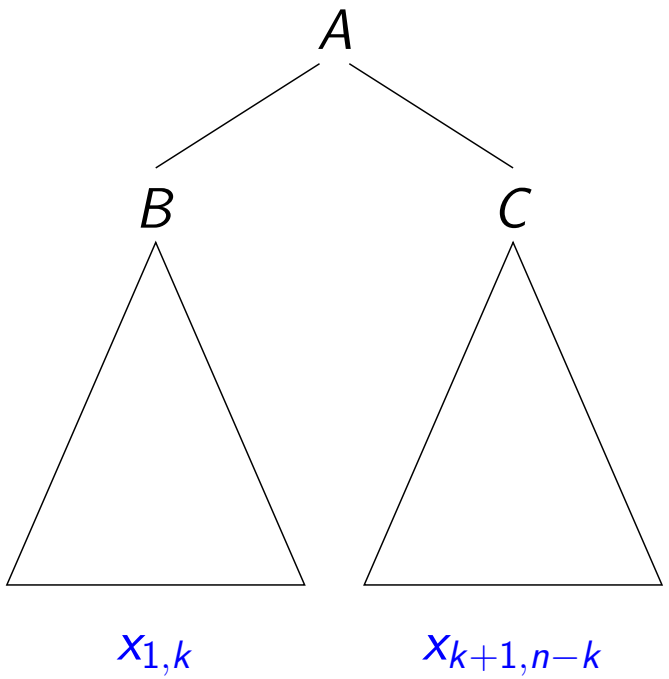
- berechne zuerst alle Variablen, aus denen sich Teilwörter der Länge 1 ableiten lassen,
- berechne dann alle Variablen, aus denen sich Teilwörter der Länge 2 ableiten lassen,
- ...
- zuletzt berechne alle Variablen, aus denen sich x ableiten lässt. Falls sich das Axiom S unter diesen Variablen befindet, so liegt x in der von der Grammatik erzeugten Sprache.

Der CYK-Algorithmus

Notation: Wir bezeichnen mit $x_{i,j}$ das Teilwort von x, das an der Stelle i beginnt und die Länge j hat.

$x = a_1 \dots a_n \quad \rightsquigarrow \quad x_{i,j} = a_i \dots a_{i+j-1}$.

Damit sieht das vorherige Bild folgendermaßen aus:



Der CYK-Algorithmus

Wir bezeichnen mit $T_{i,j}$ die Menge aller Variablen, aus denen sich $x_{i,j}$ herleiten lässt.

$T_{i,j}$ lässt sich folgendermaßen bestimmen:

- Falls $j = 1$, dann

$$T_{i,j} = \{A \mid (A \to x_{i,j}) \in P\}$$

- Falls $j > 1$, dann

$$T_{i,j} = \{A \mid (A \to BC) \in P \text{ und es gibt } k < j \text{ mit } B \in T_{i,k} \text{ und } C \in T_{i+k,j-k}\}$$

Der CYK-Algorithmus

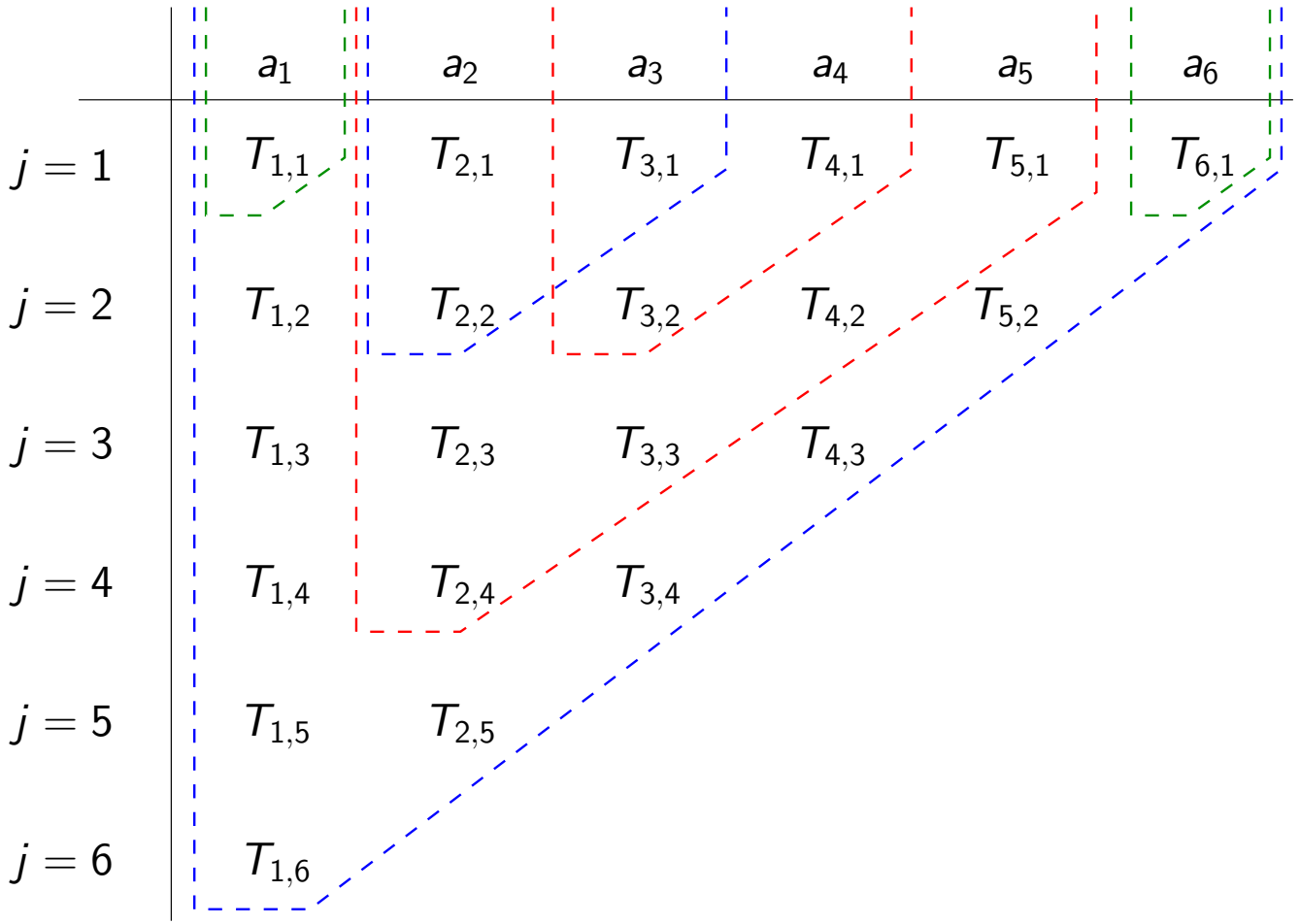
Praktische Ausführung des CYK-Algorithmus:

Wir tragen die Variablenmengen $T_{i,j}$ (von oben nach unten) in folgende Tabelle ein:

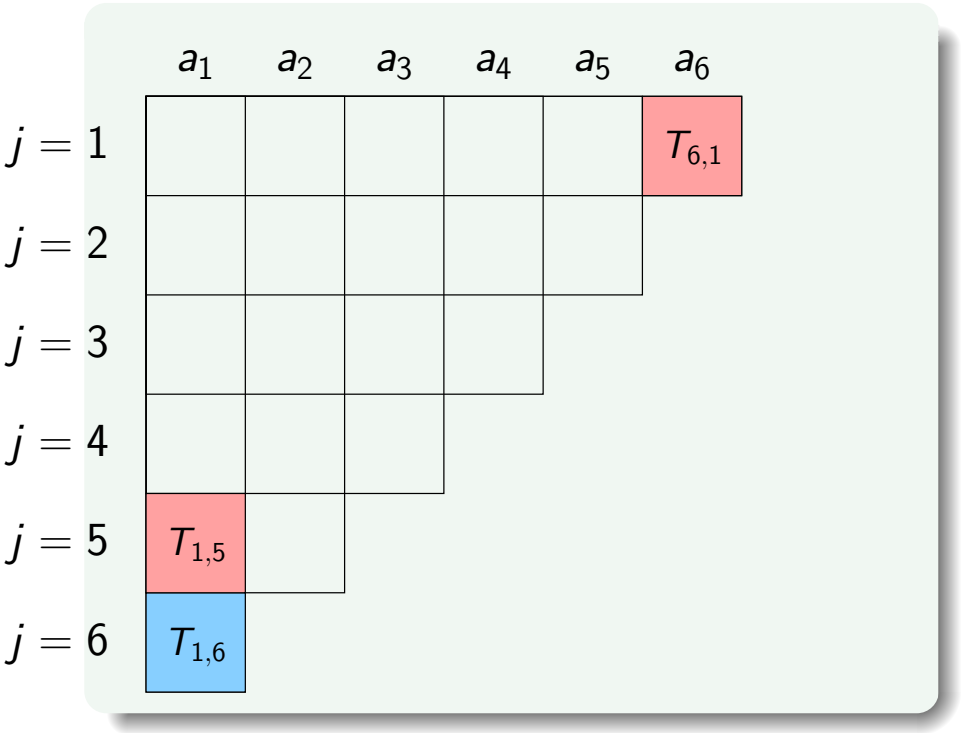
	a_1	a_2	$\dots$		a_{n-1}	a_n
$j = 1$	$T_{1,1}$	$T_{2,1}$	$\dots$	$\dots$	$T_{n-1,1}$	$T_{n,1}$
$j = 2$	$T_{1,2}$	$T_{2,2}$	$\dots$	$\dots$	$T_{n-1,2}$	
	$\dots$	$\dots$	$\dots$	$\dots$		
$\dots$	$\dots$	$\dots$	$\dots$			
$j = n-1$	$T_{1,n-1}$	$T_{2,n-1}$				
$j = n$	$T_{1,n}$					

Der CYK-Algorithmus

Folgendermaßen lässt sich veranschaulichen, welche Variablenmenge welches Teilwort ableitet:



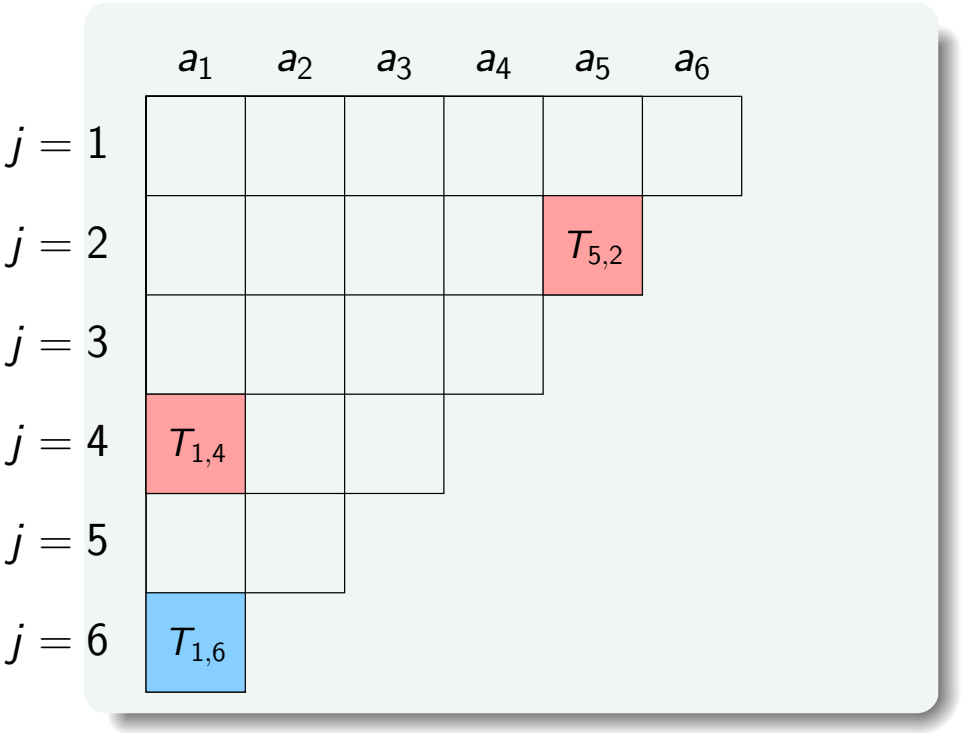
Der CYK-Algorithmus



$x = a_1 a_2 a_3 a_4 a_5 \mid a_6$

$(A \to BC) \in P$,
$B \in T_{1,5}$, $C \in T_{6,1} \Rightarrow A \in T_{1,6}$

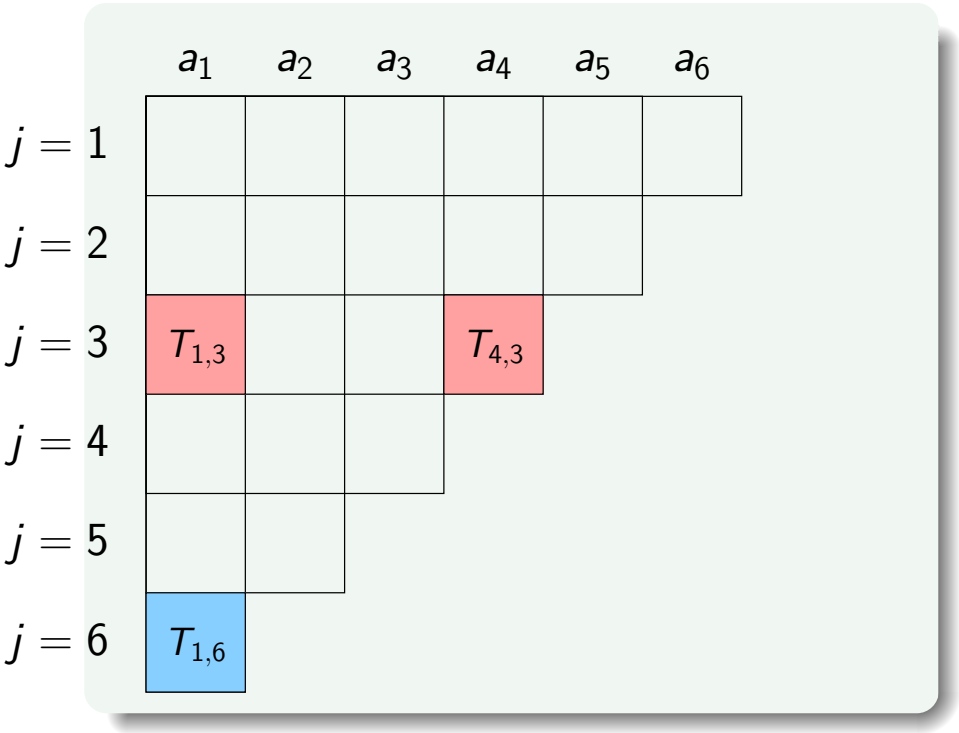
Der CYK-Algorithmus



$$x = a_1 a_2 a_3 a_4 \mid a_5 a_6$$

$$(A \to BC) \in P,$$
$$B \in T_{1,4},\ C \in T_{5,2} \Rightarrow A \in T_{1,6}$$

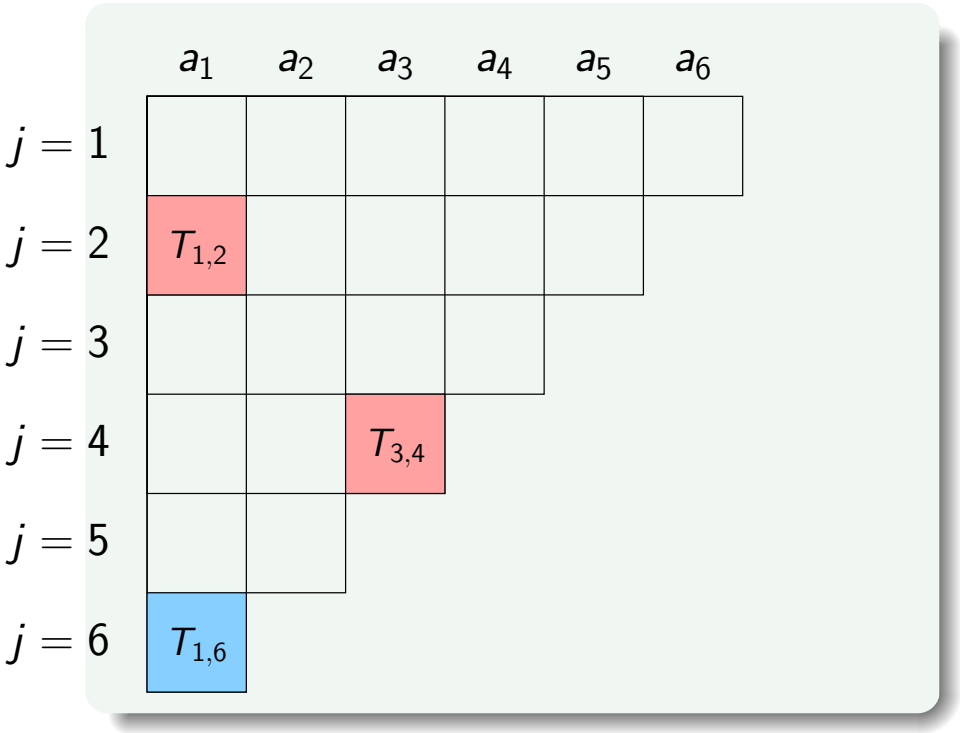
Der CYK-Algorithmus



$x = a_1 a_2 a_3 \mid a_4 a_5 a_6$

$(A \to BC) \in P$,
$B \in T_{1,3}$, $C \in T_{4,3} \Rightarrow A \in T_{1,6}$

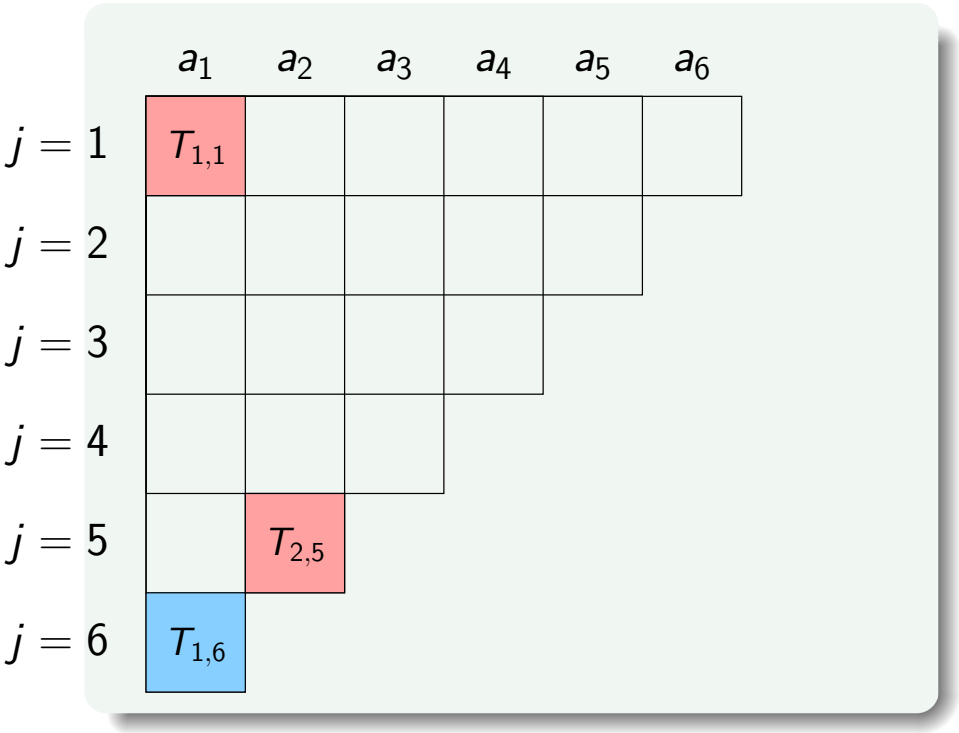
Der CYK-Algorithmus



$x = a_1 a_2 \mid a_3 a_4 a_5 a_6$

$(A \to BC) \in P$,
$B \in T_{1,2}$, $C \in T_{3,4} \Rightarrow A \in T_{1,6}$

Der CYK-Algorithmus



$x = a_1 \mid a_2 a_3 a_4 a_5 a_6$

$(A \to BC) \in P$,
$B \in T_{1,1}$, $C \in T_{2,5} \Rightarrow A \in T_{1,6}$

Der CYK-Algorithmus

Beispiel: Betrachte eine Grammatik mit folgenden Produktionen:

$$
\begin{aligned}
S &\to AD \mid FG \\
D &\to SE \mid BC \\
E &\to BC \\
F &\to AF \mid a \\
G &\to BG \mid CG \mid b \\
A &\to a \\
B &\to b \\
C &\to c
\end{aligned}
$$

Frage: Sei $x = aabcbc$. Gilt $x \in L$?

Der CYK-Algorithmus

	a	a	b	c	b	c
$j=1$	A, F	A, F	B, G	C	B, G	C
$j=2$	F	S	D, E	G	D, E	
$j=3$	S	S	G			
$j=4$		S				
$j=5$	S	D				
$j=6$	S					

Think-Pair-Share: Der CYK-Algorithmus

Wir betrachten eine Grammatik

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

für die Sprache $L = \{a^k b^k c^j \mid k \geq 0, j \geq 1\}$ mit folgenden Produktionen:
mit folgender Produktionenmenge P:

$$\begin{aligned} S &\to AB \mid B \\ A &\to ab \mid aAb \\ B &\to c \mid cB \end{aligned}$$

Untersuchen Sie mithilfe des CYK-Algorithmus, ob $x = abcc \in L$.
Erarbeiten Sie zunächst vier Minuten in Einzelarbeit eine Lösung. Anschließend tauschen Sie sich für weitere vier Minuten mit ihrem Sitznachbarn aus. Schlussendlich besprechen wir die Lösung im Plenum.

Lösungsvorschlag zu Think-Pair-Share: Der CYK-Algorithmus

Wir formen die Grammatik in Chomsky-Normalform um (vgl. vorherige Think-Pair-Share-Aufgabe) und berechnen:

	a	b	c	c
$j=1$	U_a	U_b	U_c, S, B	U_c, S, B
$j=2$	A	$-$	S, B	
$j=3$	S	$-$		
$j=4$	S			

Der CYK-Algorithmus

Komplexität des CYK-Algorithmus

Sei $n = |x|$ die Länge des Wortes, das untersucht wird. Die Größe der Grammatik wird als konstant angesehen. Dann gilt:

- $O(n^2)$ Tabellenfelder müssen ausgefüllt werden.
- Für das Ausfüllen jedes Tabellenfeldes müssen bis zu $O(n)$ andere Felder betrachtet werden.

  (Für $T_{1,n}$ müssen beispielsweise die Felder $T_{1,n-1}$, $T_{n,1}$ und $T_{1,n-2}$, $T_{n-1,2}$ und ... und $T_{1,1}$, $T_{2,n-1}$ betrachtet werden. Insgesamt $n-1$ Paare von Feldern.)

Daher ergibt sich insgesamt als Zeitkomplexität: $O(n^3)$.

Die Zeitkomplexität ist polynomiell, aber für das Parsen großer Programme nicht mehr geeignet. Dafür gibt es spezielle Methoden für bestimmte kontextfreie Grammatiken (Stichwort: $LR(k)$).

Fragestellungen zu dieser Vorlesungseinheit

Zum Ende einer jeden Vorlesungseinheit betrachten wir im Regelfall drei Fragestellungen, die mithilfe der in dieser Einheit besprochenen Inhalte beantwortet werden sollen. In der darauffolgenden Einheit können zu Beginn mögliche Antworten gesammelt werden.

Fragen zur achten Vorlesungseinheit

- Wie überführt man eine Grammatik, bei denen die linke Seite einer jeden Überführungsregel eine einzelne Variable ist, in Chomsky-Normalform?
- Wie findet man für eine kontextfreie Grammatik G in CNF effizient heraus, ob ein gegebenes Wort w in $L(G)$ liegt?
- Angenommen der CYK-Algorithmus wurde auf das Wort $w = abcabc$ und die Grammatik G angewandt, welche Bedeutung hat dann der Eintrag in der zweiten Zeile, dritte Spalte in der Tabelle?

Pumping-Lemma

Weitgehend analog zu regulären Sprachen kann man nun ein Pumping-Lemma für kontextfreie Sprachen zeigen.

Die für reguläre Sprachen und endliche Automaten geltende Aussage

> Jedes ausreichend lange Wort durchläuft einen Zustand des Automaten zweimal.

wird dabei ersetzt durch

> Auf einem Pfad des Syntaxbaums, der die Ableitung eines ausreichend langen Wortes durch eine kontextfreie Grammatik darstellt, kommt eine Variable mindestens zweimal vor.

Pumping-Lemma

Was bedeutet hier "ausreichend langes Wort"?

Die Beantwortung dieser Frage hängt davon ab, in welcher Form sich die Grammatik befindet. Wir nehmen an, sie befindet sich in Chomsky-Normalform.

Dann gilt: Syntaxbäume sind (bis auf die unterste Schicht der Blätter) immer Binärbäume (aufgrund der Produktionen der Form $A \to BC$). Und für Binärbäume gilt:

Länge von Pfaden in Binärbäumen (Lemma)

Sei B ein Binärbaum (d.h., jeder Knoten in B hat entweder null oder zwei Kinder) mit mindestens 2^k Blättern.
Dann hat B einen von der Wurzel ausgehenden Pfad, der aus mindestens k Kanten und $k+1$ Knoten besteht.

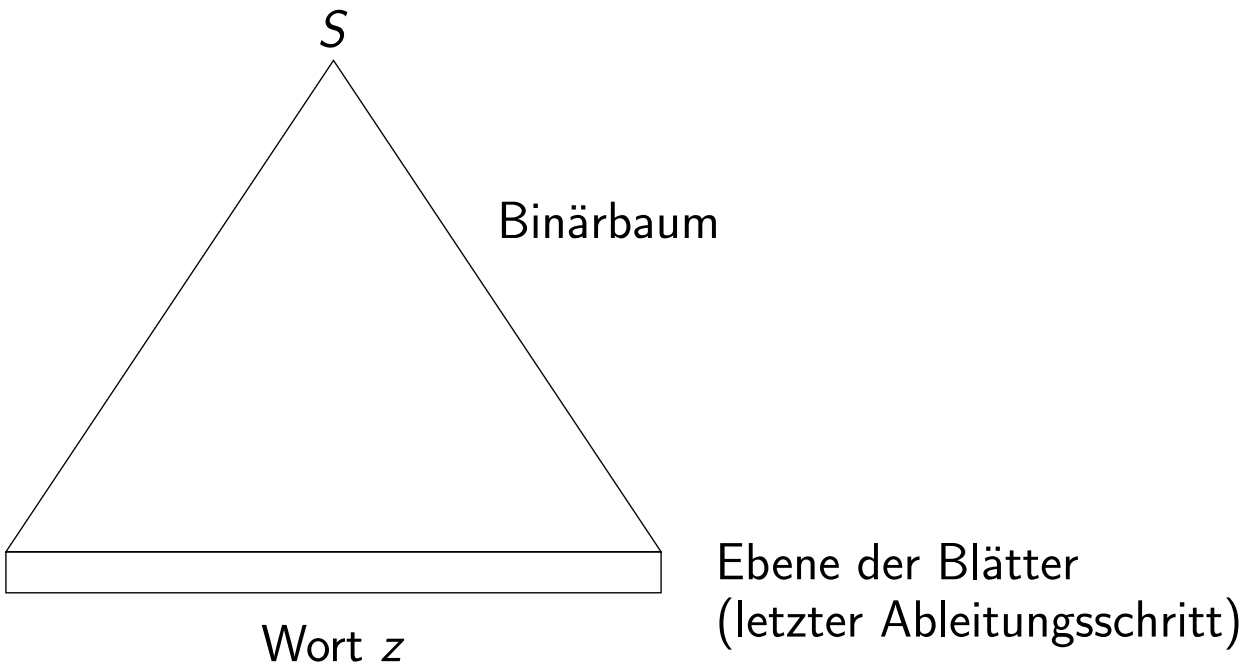
Pumping-Lemma

Das bedeutet:

- Sei k die Anzahl von Variablen in G ($k = |V|$). Für ein Wort $z \in L$ mit $|z| \geq 2^k$ hat dann der zugehörige Syntaxbaum mindestens 2^k Blätter.
- Das bedeutet auch, dass der obere Teil des Syntaxbaums (bei dem die Blätter abgeschnitten sind) mindestens einen Pfad mit $k + 1$ Knoten hat.
- Auf diesem Pfad, der nur innere Knoten enthält, muss eine Variable – nennen wir sie A – mindestens zweimal vorkommen.

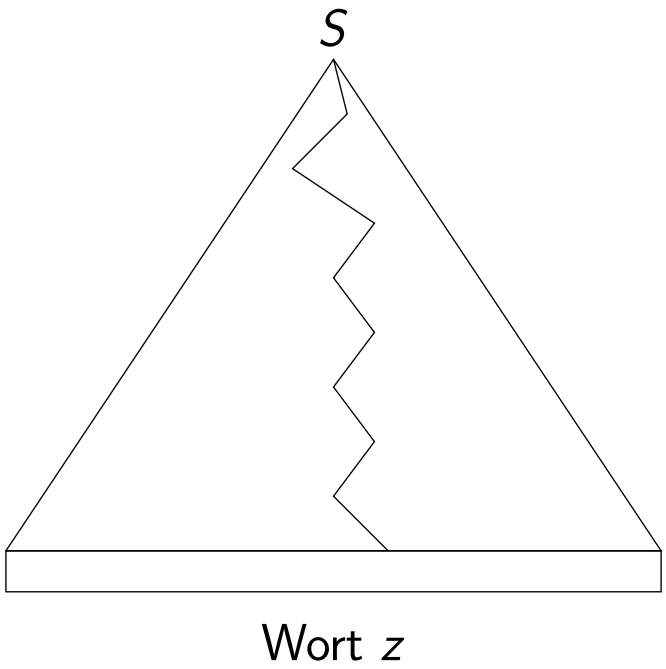
Pumping-Lemma

Syntaxbaum für ein Wort z mit $|z| \geq n = 2^k$
n ist hier die "Konstante des Pumping-Lemmas"



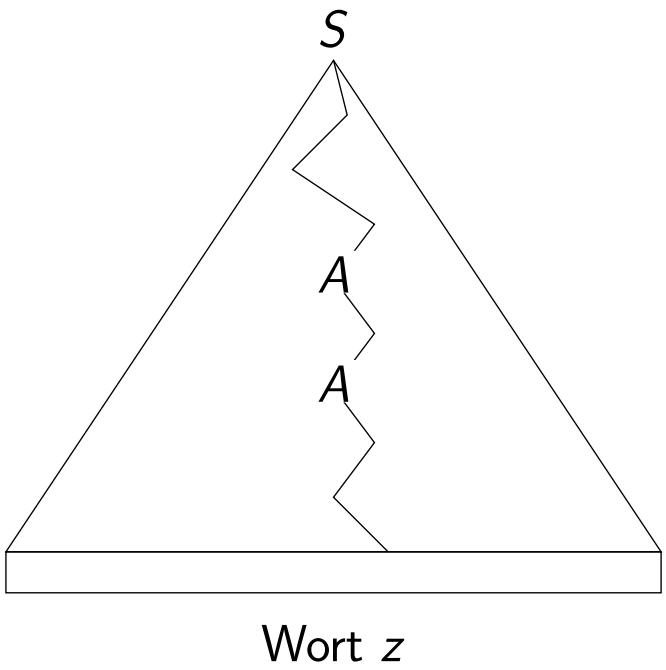
Pumping-Lemma

Es gibt einen Pfad mit mindestens $k + 1$ inneren Knoten.



Pumping-Lemma

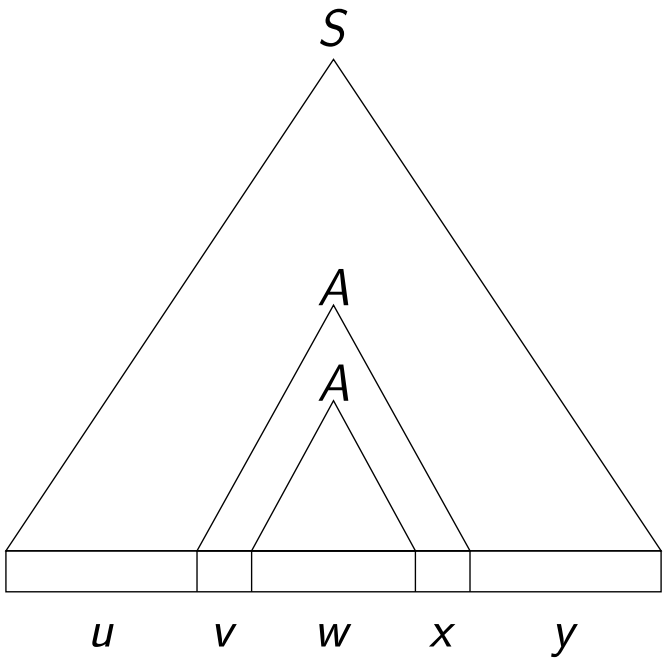
Auf diesem Pfad gibt es eine Variable, die zweimal auftaucht, beispielsweise A.



Pumping-Lemma

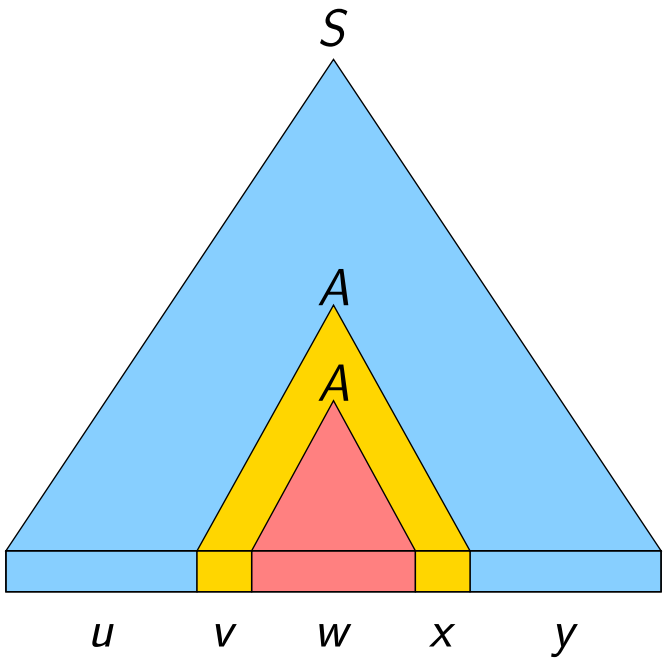
Das Wort z wird nun in fünf Teilwörter u, v, w, x, y aufgespalten:

- w wird aus dem unteren A abgeleitet: $A \Rightarrow^* w$
- vwx wird aus dem oberen A abgeleitet: $A \Rightarrow^* vwx$



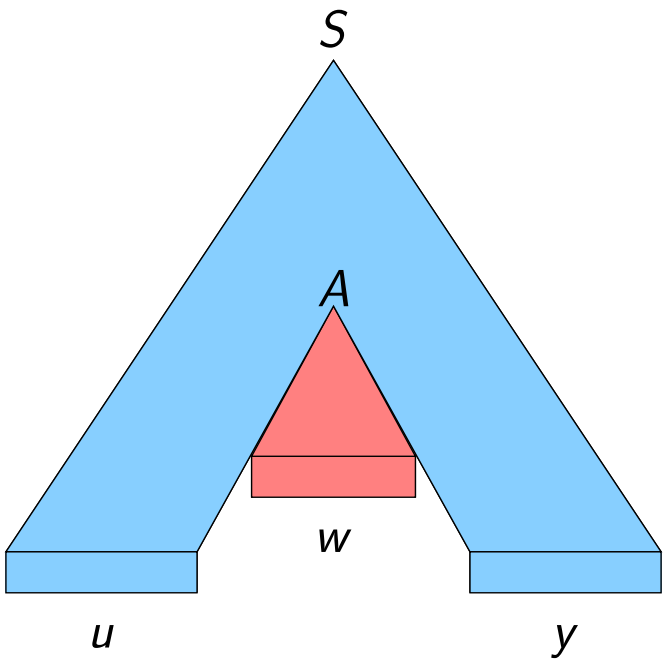
Pumping-Lemma

Damit erhält man drei ineinander enthaltene Teil-Syntaxbäume, die man neu zusammenstecken kann.



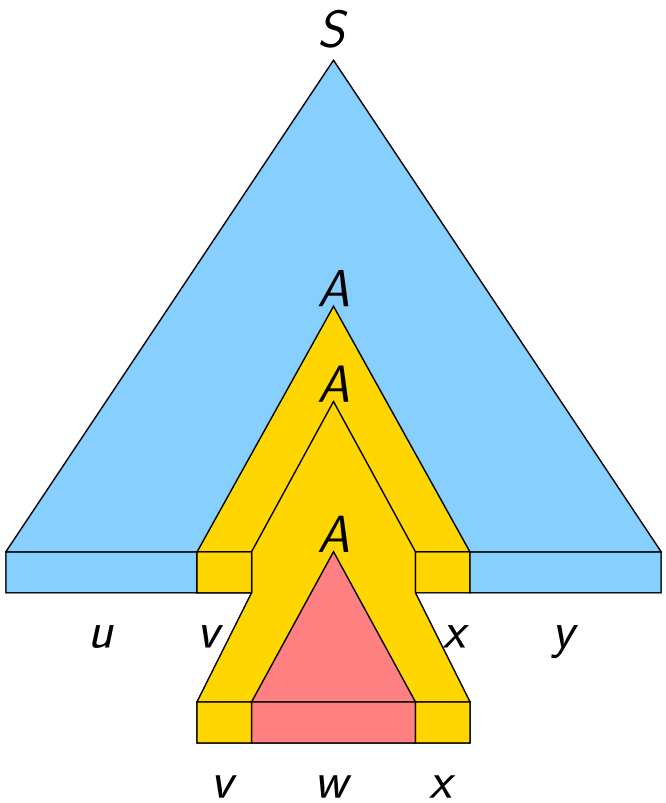
Pumping-Lemma

Durch Weglassen des mittleren Teilbaums erhält man einen Syntaxbaum für uwy. Damit gilt: $uwy \in L$.



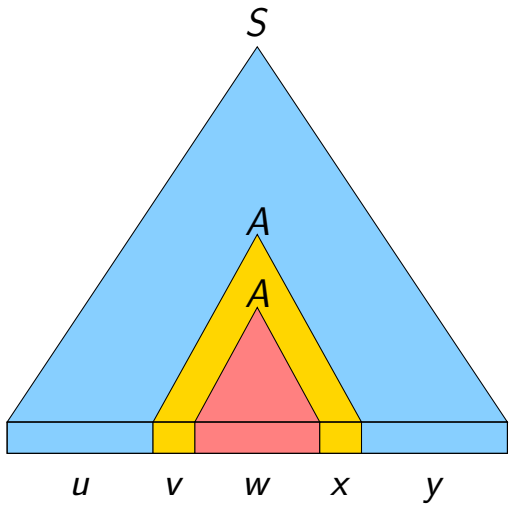
Pumping-Lemma

Durch Verdoppeln des mittleren Teilbaums erhält man einen Syntaxbaum für uv^2wx^2y. Damit gilt: $uv^2wx^2y \in L$.



Pumping-Lemma

Außerdem kann man für v, w, x folgende Eigenschaften verlangen:
$|vwx| \leq n = 2^k$:
Wir können annehmen, dass wir das am weitesten unten liegende Doppelvorkommen gewählt haben, d.h., das Doppelvorkommen mit der größten Tiefe. Das kann dadurch erreicht werden, dass einer der Pfade maximaler Länge von unten nach oben verfolgt wird. Demnach ist der Abstand des oberen A zur Blattebene höchstens k und der darunter hängende Binärbaum hat höchstens 2^k Blätter.

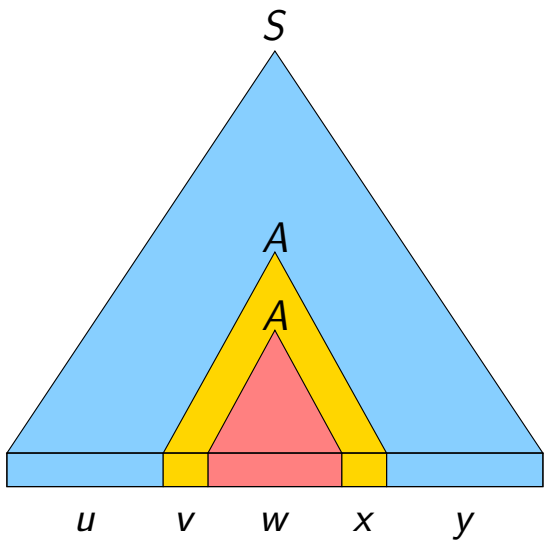


Pumping-Lemma

$|vx| \geq 1$:

Seien B, C die beiden Kinder des oberen A. Dann geht das untere A entweder aus B oder C hervor. Die jeweils andere Variable muss – da die Grammatik in Chomsky-Normalform ist – ein nicht-leeres Wort ableiten.

Und dieses Wort ist ein Teilwort von v bzw. von x.



Das Pumping-Lemma

Pumping-Lemma, *uvwxy*-Theorem (Satz)

Sei L eine kontextfreie Sprache. Dann gibt es eine Zahl n, so dass sich alle Wörter $z \in L$ mit $|z| \geq n$ zerlegen lassen in $z = uvwxy$, so das folgende Eigenschaften erfüllt sind:

1. $|vx| \geq 1$,
2. $|vwx| \leq n$ *und*
3. für alle $i = 0, 1, 2, \ldots$ gilt: $uv^i wx^i y \in L$.

Dabei geht $n = 2^k$ aus der Anzahl k der Variablen einer kontextfreien Grammatik für L hervor.

Das Pumping-Lemma

Wie bereits beim Pumping-Lemma für reguläre Sprachen kann das Pumping-Lemma für kontextfreie Sprachen dazu genutzt werden, zu zeigen, dass eine Sprache *nicht* kontextfrei ist, indem wir die Aussage negieren. Also, wenn

- *Für alle* Zahlen $n \in \mathbb{N}_0$
- ein Wort $z \in L$ *existiert*, so dass
- *für alle* Zerlegungen $z = uvwxy$ mit $|vx| \geq 1$, $|vwx| \leq n$
- ein $i \in \mathbb{N}_0$ *exitiert*, so dass $uv^i wx^i y \notin L$

dann ist L *nicht* kontextfrei.
Anmerkung: Wie auch beim Pumping-Lemma für reguläre Sprachen gilt die Implikation nur in die angegebene Richtung, wenn die Pumping-Eigenschaft für eine Sprache L erfüllt ist, muss L nicht zwingend kontextfrei sein.

Pumping-Lemma

Anwendung des Pumping-Lemmas: wir zeigen, dass die Sprache $L = \{a^m b^m c^m \mid m \geq 1\}$ nicht kontextfrei ist.

1. Wir nehmen eine *beliebige* Zahl n an.
2. Wir wählen ein Wort $z \in L$ mit $|z| \geq n$. In diesem Fall eignet sich $z = a^n b^n c^n$.
3. Wir betrachten nun *alle* möglichen Zerlegungen $z = uvwxy$ mit den Einschränkungen $|vx| \geq 1$ und $|vwx| \leq n$.

   Wegen $|vwx| \leq n$ gilt, dass vx nicht aus a's, b's *und* c's bestehen kann, denn es kann sich nicht über den gesamten b-Block erstrecken.

Pumping-Lemma

4. Wir wählen für alle diese möglichen Zerlegungen $i = 2$ und betrachten uv^2wx^2y. Wegen der obigen Überlegungen sind nun ein oder zwei Alphabetsymbole gepumpt worden, mindestens eines jedoch nicht.

   Damit ist klar, dass uv^2wx^2y nicht in L liegen kann, denn jedes Wort in L hat gleich viele a's, b'c und c's.

Pumping-Lemma

Als weiteres Beispiel betrachten wir das Adventure, Level 3, und zeigen, dass die Menge aller zulässigen Pfade eines Adventures nicht notwendigerweise kontextfrei sein muss.

Wiederholung der Regeln:

Die Schatz-Regel

Man muss mindestens zwei Schätze finden.

Pumping-Lemma

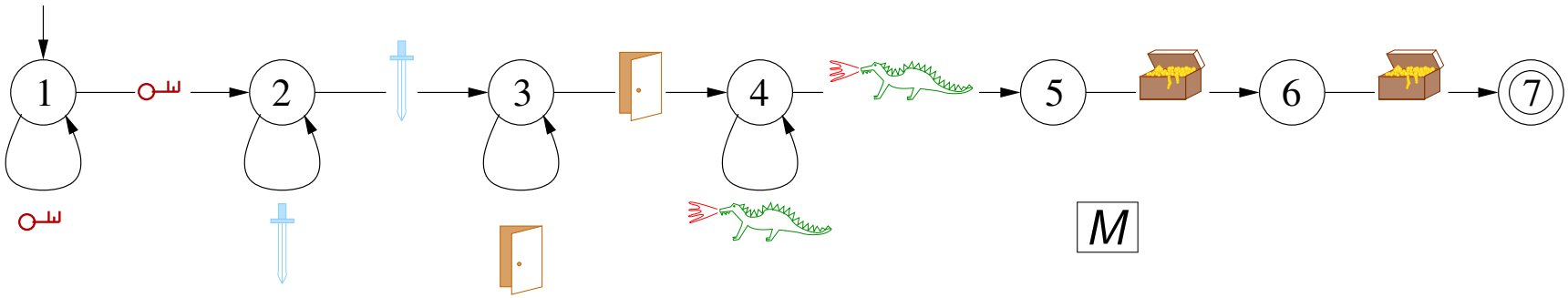
Neue Drachen-Regel

Auch Schwerter werden durch das Drachenblut unbenutzbar, sobald man einen Drachen damit getötet hat. Außerdem werden Drachen sofort wieder "ersetzt".
Es gibt jedoch immer noch die Option, ein Schwert nicht zu benutzen und nach der Begegnung mit dem Drachen in den Fluss zu springen.

Neue Tür-Regel

Die Schlüssel sind magisch und verschwinden sofort, nachdem eine Tür mit ihnen geöffnet wurde. Sobald man eine Tür durchschritten hat, schließt sie sich sofort wieder.

Pumping-Lemma



Wir betrachten folgende Sprache A_M:

$$\begin{aligned} A_M &= \{w \mid w \text{ entspricht einem Pfad durch das oben} \\ &\qquad \text{angegebene Adventure, d.h., } w \in T(M), \text{ und} \\ &\qquad \text{erfüllt alle Regeln für Level 3}\} \\ &= \{\, L^k W^\ell T^m D^n A^2 \mid k \geq m \geq 1, \ell \geq n \geq 1 \,\} \end{aligned}$$

L=Schlüssel W=Schwert T=Tür D=Drache A=Schatz

Pumping-Lemma

Wir zeigen nun, dass A_M nicht kontextfrei ist.

1. Gegeben sei eine beliebige Zahl n.
2. Wir wählen als Wort $z = L^n W^n T^n D^n A^2 \in A_M$.
3. Sei nun $z = uvwxy$ eine beliebige Zerlegung von x mit $|vx| \geq 1$ und $|vwx| \leq n$.

   Dann kann vx nicht gleichzeitig Schlüssel und Türen und nicht gleichzeitig Schwerter und Drachen enthalten.

Pumping-Lemma

4. Wir machen nun folgende Fallunterscheidung:
   - vx enthält zumindest einen Schatz: dann enthält uv^0wx^0y höchstens noch einen Schatz und kann nicht in A_M liegen, da die Schatz-Regel verletzt ist.
   - vx enthält zumindest einen Schlüssel: dann enthält uv^0wx^0y weniger als n Schlüssel, aber immer noch n Türen und kann nicht in A_M liegen.
   - vx enthält zumindest ein Schwert: dann enthält uv^0wx^0y weniger als n Schwerter, aber immer noch n Drachen und kann nicht in A_M liegen.
   - vx enthält zumindest eine Tür: dann enthält uv^2wx^2y mehr als n Türen, aber immer nur noch n Schlüssel und kann nicht in A_M liegen.
   - vx enthält zumindest einen Drachen: dann enthält uv^2wx^2y mehr als n Drachen, aber immer nur noch n Schwerter und kann nicht in A_M liegen.

Pumping-Lemma

Da wir damit jeden Fall behandelt haben, folgt daraus, dass A_M nicht kontextfrei ist.

Nebenbemerkung: Folgende Sprachen mit vertauschten Blöcken sind allerdings kontextfrei.

- $\{ L^k T^m W^\ell D^n A^2 \mid k \geq m \geq 1, \ell \geq n \geq 1 \}$
- $\{ L^k W^\ell D^n T^m A^2 \mid k \geq m \geq 1, \ell \geq n \geq 1 \}$

Übungsaufgabe: Kontextfreie Grammatiken für diese Sprachen angeben.

Think-Pair-Share: Pumping-Lemma

Wir betrachten die folgende Sprache:

$$L = \{a^n b^m c^l \mid n \geq m \geq l\}$$

Zeigen Sie mithilfe des Pumping-Lemmas für kontextfreie Sprachen, dass L nicht kontextfrei ist. Wählen Sie für ein gegebenes $n \in \mathbb{N}_0$ ein passendes Wort z, das mindestens die Länge n hat, und nehmen Sie eine passende Fallunterscheidung für alle möglichen Zerlegungen $z = uvwxy$ vor, so dass $|vx| \geq 1$ und $|vwx| \leq n$.
Erarbeiten Sie zunächst fünf Minuten in Einzelarbeit eine Lösung. Anschließend tauschen Sie sich für weitere fünf Minuten mit ihrem Sitznachbarn aus. Schlussendlich besprechen wir die Lösung im Plenum.
Hinweis: Bei günstiger Wahl von z kann es sinnvoll sein, eine Fallunterscheidung danach durchzuführen, ob vx ein a enthält oder nicht.

Lösungsvorschlag zu Think-Pair-Share: Pumping-Lemma

Es sei ein $n \in \mathbb{N}_0$ beliebig gegeben. Wir wählen $z = a^n b^n c^n$ und zerlegen $z = uvwxy$ mit $|vx| \geq 1$, $|vwx| \leq n$. Wir führen eine Fallunterscheidung danach durch, ob v ein a enthält oder nicht.
Falls vx ein a enthält, enthält vwx wegen $|vwx| \leq n$ kein c, also enthält uwy maximal $n-1$ as und n cs, also ist $uwy \notin L$.
Falls vx kein a enthält, enthält v oder x mindestens ein b oder ein c, da $|vx| \geq 1$. Also enthält uv^2wx^2y weiterhin n as aber mindestens $n+1$ bs oder mindestens $n+1$ cs. In beiden Fällen ist $uv^2wx^2y \notin L$.

Pumping-Lemma

Man kann auch für folgende Sprachen zeigen, dass sie nicht kontextfrei sind:

$$\begin{aligned} L_1 &= \{0^p \mid p \text{ ist Primzahl}\} \\ L_2 &= \{0^n \mid n \text{ ist Quadratzahl}\} \\ L_3 &= \{0^{2^n} \mid n \geq 0\} \end{aligned}$$

Bemerkung: Man kann zeigen, dass eine kontextfreie Sprache über einem einelementigen Alphabet immer regulär ist. Daher reicht es, nachzuweisen, dass die obigen Sprachen nicht regulär sind.

Abschlusseigenschaften

Abgeschlossenheit

Die kontextfreien Sprachen sind abgeschlossen unter:

- Vereinigung (L_1, L_2 kontextfrei $\Rightarrow$ $L_1 \cup L_2$ kontextfrei)
- Produkt/Konkatenation (L_1, L_2 kontextfrei $\Rightarrow$ $L_1 L_2$ kontextfrei)
- Stern-Operation (L kontextfrei $\Rightarrow$ L^* kontextfrei)

Die kontextfreien Sprachen sind nicht abgeschlossen unter:

- Schnitt
- Komplement

Abschlusseigenschaften

Abschluss unter Vereinigung

Wenn L_1 und L_2 kontextfreie Sprachen sind, dann ist auch $L_1 \cup L_2$ kontextfrei.

Begründung: Gegeben zwei kontextfreie Grammatiken

$$G_1 = (V_1, \Sigma, P_1, S_1), \quad G_2 = (V_2, \Sigma, P_2, S_2)$$

(mit $V_1 \cap V_2 = \emptyset$) für L_1, L_2, so ist mit der neuen Variable $S \notin V_1 \cup V_2$:

$$G = (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}, S)$$

eine kontextfreie Grammatik für $L_1 \cup L_2$. (Ggf. müssen ε-Ableitungen noch von S_1 / S_2 nach S vorgezogen werden.)

Abschlusseigenschaften

Abschluss unter Produkt/Konkatenation

Wenn L_1 und L_2 kontextfreie Sprachen sind, dann ist auch $L_1 L_2$ kontextfrei.

Begründung: Gegeben zwei kontextfreie Grammatiken

$$G_1 = (V_1, \Sigma, P_1, S_1), \quad G_2 = (V_2, \Sigma, P_2, S_2)$$

(mit $V_1 \cap V_2 = \emptyset$) für L_1, L_2, so ist

$$G = (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S)$$

eine kontextfreie Grammatik für $L_1 L_2$. (Ggf. müssen ε-Ableitungen noch von S_1 / S_2 nach S vorgezogen werden.)

Abschlusseigenschaften

Abschluss unter der Stern-Operation

Wenn L eine kontextfreie Sprache ist, dann ist auch L^* kontextfrei.

Begründung: Gegeben sei eine kontextfreie Grammatiken

$$G_1 = (V_1, \Sigma, P_1, S_1)$$

für L. Dann ist

$$G = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to \varepsilon, S \to S_1 S\}, S)$$

eine kontextfreie Grammatik für L^*. (Ggf. muss eine ε-Ableitungen von S_1 noch entfernt werden.)

Abschlusseigenschaften

Kein Abschluss unter Schnitt

Wenn L_1 und L_2 kontextfreie Sprachen sind, dann ist $L_1 \cap L_2$ nicht notwendigerweise kontextfrei.

Gegenbeispiel: Die Sprachen

$$\begin{aligned} L_1 &= \{a^j b^k c^k \mid j \geq 0, k \geq 0\} \\ L_2 &= \{a^k b^k c^j \mid j \geq 0, k \geq 0\} \end{aligned}$$

sind beide kontextfrei. Für ihren Schnitt gilt jedoch

$$L_1 \cap L_2 = \{a^k b^k c^k \mid k \geq 0\}$$

und diese Sprache ist – wie mit dem Pumping-Lemma gezeigt wurde – nicht kontextfrei.

Abschlusseigenschaften

Kein Abschluss unter Komplement

Wenn L eine kontextfreie Sprache ist, dann ist $\overline{L} = \Sigma^* \backslash L$ nicht notwendigerweise kontextfrei.

Begründung: Nehmen wir an, die kontextfreien Sprachen wären unter Komplement abgeschlossen. Wegen $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ wären sie dann auch unter Schnitt abgeschlossen, was aber nicht der Fall ist. D.h., wir erhalten einen Widerspruch.

Kontextfreie Sprachen und XML

Wir betrachten eine wichtige Anwendung kontextfreier Sprachen: Document Type Definitions (DTDs), mit Hilfe derer die Struktur von XML-Dokumenten beschrieben werden kann.

XML (eXtensible Markup Language)

XML ist eine generische Markup-Sprache, die als Standard für die Erstellung von maschinen- und menschen-lesbaren Dokumenten verwendet wird. XML definiert dabei die Regeln für den Aufbau solcher Dokumente.

Für einen bestimmten Typ von Dokumenten, d.h., für eine spezifische Markup-Sprache muss dabei zunächst festgelegt werden, welcher Aufbau und welche Datenstrukturierungen zulässig sind. Dies geschieht mit Hilfe sogenannter DTDs (Document Type Definitions).

Kontextfreie Sprachen und XML

Beispiel (aus Hopcroft, Motwani, Ullman): Anwendungsspezifische Sprache für eine PC-Datenbank. Folgendes Dokument ist in dieser Sprache beschrieben:

```
<PCS>
  <PC>
    <MODEL>Notebook 5000</MODEL>
    <PRICE>EUR 410</PRICE>
    <PROCESSOR>
      <MANF>Intel</MANF>
      <MODEL>Core i3</MODEL>
      <SPEED>1.7 GHz</SPEED>
    </PROCESSOR>
    <RAM>4 GB</RAM>
    <DISK><HARDDISK>
      <MANF>Seagate</MANF>
      <MODEL>SATA</MODEL>
      <SIZE>500 GB</SIZE>
    </HARDDISK></DISK>
    <DISK><DVD>
      <SPEED>16x</SPEED>
    </DVD></DISK>
  </PC>
  <PC>
    ...
  </PC>
</PCS>
```

Kontextfreie Sprachen und XML

Fragen:

- Wie viele PCs können in einer solchen Datei aufgeführt werden? Beliebig viele? Ist auch eine Datei mit überhaupt keinem PC zulässig?
- Welche Einträge braucht man, um einen PC zu beschreiben? Muss der Preis immer angegeben werden?
- Kann ein PC mehrere Prozessoren haben? Oder mehrere Festplatten?

Allgemein: Was ist überhaupt ein zulässiges Dokument?

Kontextfreie Sprachen und XML

Die Frage, welche Dokumente überhaupt zulässig sind, kann durch das Betrachten der DTD (mit Namen `PcSpecs`) beantwortet werden:

```
<!DOCTYPE PcSpecs [
<!ELEMENT PCS (PC*)>
<!ELEMENT PC (MODEL, PRICE, PROCESSOR, RAM, DISK+)>
<!ELEMENT MODEL (#PCDATA)>
<!ELEMENT PRICE (#PCDATA)>
<!ELEMENT PROCESSOR (MANF, MODEL, SPEED)>
<!ELEMENT MANF (#PCDATA)>
<!ELEMENT MODEL (#PCDATA)>
<!ELEMENT SPEED (#PCDATA)>
<!ELEMENT RAM (#PCDATA)>
```

Kontextfreie Sprachen und XML

```
<!ELEMENT DISK (HARDDISK | CD | DVD)>
<!ELEMENT HARDDISK (MANF, MODEL, SIZE)>
<!ELEMENT SIZE (#PCDATA)>
<!ELEMENT CD (SPEED)>
<!ELEMENT DVD (SPEED)>
]>
```

Kontextfreie Sprachen und XML

Bedeutung der DTD-Einträge:

- Die erste Zeile (gekennzeichnet mit `DOCTYPE`) enthält den Namen der DTD (`PcSpecs`).
- Alle anderen Zeilen enthalten Regeln (bzw. Mengen von Regeln) einer kontextfreien Grammatik.
- Alle groß geschriebenen Wörter (`PCS`, `PC`, `MODEL`, etc.) beziehen sich auf Variablen der Grammatik. `#PCDATA` steht für Text, der keine XML-Tags der Form `<..> .. </..>` beinhaltet.

Kontextfreie Sprachen und XML

Regelformat:

- Die Beschreibung einer Regel beginnt mit dem Schlüsselwort `!ELEMENT`.
- Anschließend folgt die linke Seite, bestehend aus einer Variablen A.
- Die rechte Seite ist ein regulärer Ausdruck α. Daher steht eine DTD-Regel nicht für eine kontextfreie Regel, sondern für eine (unendliche) Menge von kontextfreien Regeln, die alle die Form $A \to w$ haben, wobei $w \in L(\alpha)$.

Ein XML-Dokument ist zulässig, wenn es durch diese kontexfreien Regeln erzeugt werden kann.

Kontextfreie Sprachen und XML

Syntax der regulären Ausdrücke: die regulären Ausdrücke auf der rechten Seite einer Regel werden durch folgende Operatoren dargestellt.

- | – Vereinigung, entspricht dem Operator | bei regulären Ausdrücken.
- , – Konkatenation/Produkt
- Drei Varianten des Operators zur Hüllenbildung:
  * – Stern-Operation, null oder mehr Vorkommen
  + – Stern-Operation unter Ausschluss des leeren Wortes: mindestens ein Vorkommen $((r)^+ = (r)^* r)$
  ? – null oder ein Vorkommen $((r)? = (r \mid \varepsilon))$

Kontextfreie Sprachen und XML

Beispiel:

```
<!ELEMENT PC (MODEL, PRICE, PROCESSOR, RAM, DISK+)>
```

steht für alle Regeln der Form $\texttt{PC} \to w$, wobei
$w \in L(\texttt{MODEL PRICE PROCESSOR RAM } (\texttt{DISK})^+)$.

Dazu gehören folgende Regeln:

$$\begin{aligned} \texttt{PC} &\to \texttt{MODEL PRICE PROCESSOR RAM DISK} \\ \texttt{PC} &\to \texttt{MODEL PRICE PROCESSOR RAM DISK DISK} \\ &\ldots \end{aligned}$$

Das bedeutet: eine PC-Beschreibung muss eine oder mehrere Einträge für Disks (Festplatten, CD-Laufwerke, etc.) haben.

Kontextfreie Sprachen und XML

Eine DTD-Grammatik kann schematisch in eine herkömmliche kontextfreie Grammatik übersetzt werden. Beispielsweise kann die Regel

```
<!ELEMENT PC (MODEL, PRICE, PROCESSOR, RAM, DISK+)>
```

übersetzt werden nach:

$$
\begin{aligned}
\texttt{PC} &\rightarrow \texttt{MODEL PRICE PROCESSOR RAM DISKS} \\
\texttt{DISKS} &\rightarrow \texttt{DISK} \\
\texttt{DISKS} &\rightarrow \texttt{DISKS DISK}
\end{aligned}
$$

Kontextfreie Sprachen und XML

Weitere Bemerkungen:

- Zum eindeutigen Parsen werden um jede der ursprünglichen Variablen `A` noch Tags `<A>...</A>` gelegt.
- Das Aufbauen eines Syntaxbaums aus einem XML-Dokument wird automatisch von Funktionen einer XML-Library erledigt. Hierzu muss der Benutzer keinen eigenen Code schreiben.
- In der Praxis benutzte DTDs enthalten noch weitere Einträge, beispielsweise für Attribute, Verweise auf externe Dokumente, etc.

Fragestellungen zu dieser Vorlesungseinheit

Zum Ende einer jeden Vorlesungseinheit betrachten wir im Regelfall drei Fragestellungen, die mithilfe der in dieser Einheit besprochenen Inhalte beantwortet werden sollen. In der darauffolgenden Einheit können zu Beginn mögliche Antworten gesammelt werden.

Fragen zur neunten Vorlesungseinheit

- Welche Rolle spielt die Chomsky-Normalform beim Pumping-Lemma für kontextfreie Sprachen?
- Wie zeigt man mithilfe des Pumping-Lemmas, dass eine Sprache nicht kontextfrei ist?
- Wie kann man einsehen, dass kontextfreie Sprachen nicht unter Schnitt abgeschlossen sind?

Kellerautomaten

Was ist ein geeignetes Automatenmodell für kontextfreie Sprachen?

Analog zu regulären Sprachen suchen wir hier ein Automatenmodell für kontextfreie Sprachen.

Antwort: Kellerautomaten (englisch: push-down automata)

Automaten, die mit einem zusätzlichen Keller (englisch: stack) ausgestattet sind.

Kellerautomaten

Nutzen eines solchen Automatenmodells

Manche Konstruktionen und Verfahren lassen sich besser mit Hilfe des Automatenmodells durchführen (anstatt auf Grammatiken). Dazu gehört:

- das Wortproblem (wir werden herausfinden, dass das Wortproblem unter bestimmten Umständen effizienter als in Zeit $O(n^3)$ gelöst werden kann)
- Abschlusseigenschaften (Abschluss von kontextfreien Sprachen unter Schnitt mit regulären Sprachen lässt sich gut mit Kellerautomaten zeigen)

Kellerautomaten

Wir betrachten die Sprache

$$\begin{aligned} L &= \{a_1 a_2 \dots a_n \$ a_n \dots a_2 a_1 \mid a_i \in \Delta\} \\ &= \{w \$ w^R \mid w \in \Delta^*\} \end{aligned}$$

mit $\Sigma = \Delta \cup \{\$\}$ für ein Alphabet Δ. Dabei steht w^R für die Umkehrung des Wortes w (zum Beispiel: $(abc)^R = cba$).

Ein endlicher Automat kann diese Sprache deshalb nicht erkennen, weil er sich keine beliebig langen Wörter der Form $a_1 a_2 \dots a_n$ "merken" kann. Er müsste sich aber solche Wörter merken, um die Übereinstimmung mit dem Wortteil nach dem $ zu überprüfen.
Übungsaufgabe: Zeigen Sie mit Hilfe von Myhill-Nerode-Äquivalenz, dass L nicht regulär ist.

Kellerautomaten

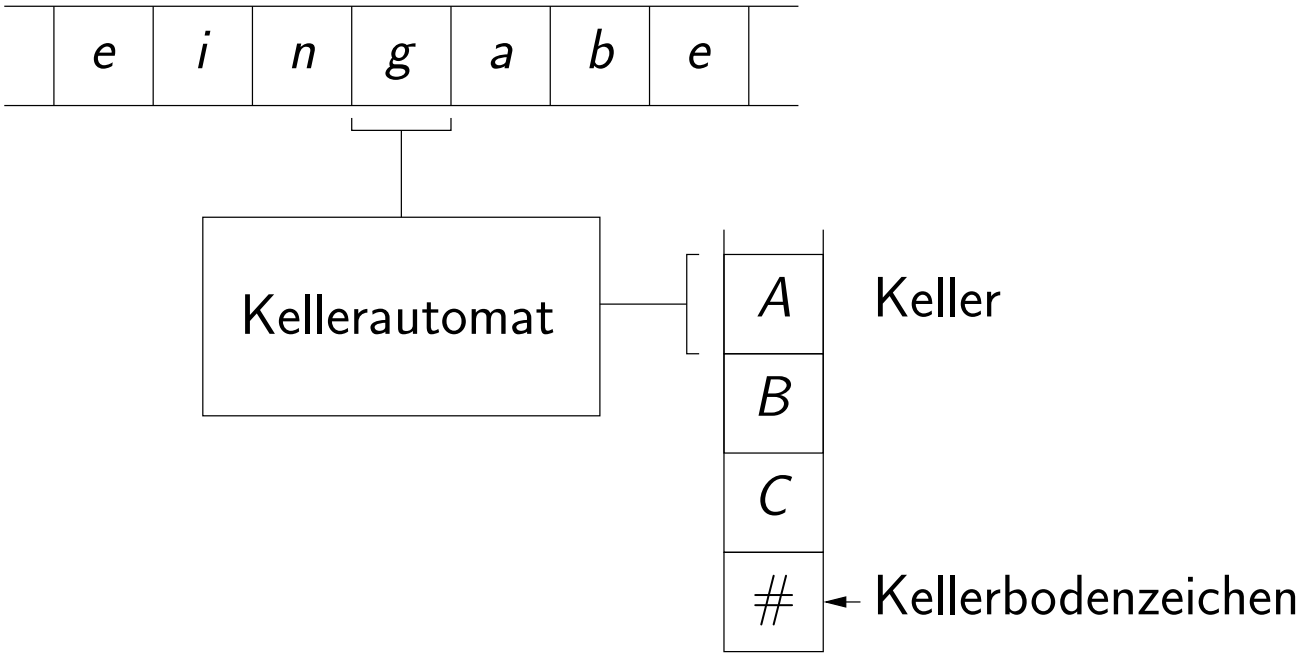
Um ein Automatenmodell für kontextfreie Sprachen zu erhalten,

- führen wir daher einen Keller oder Pushdown-Speicher ein, auf dem sich eine beliebig lange Sequenz von Zeichen befinden darf.
- Beim Einlesen eines neuen Zeichens darf das oberste Zeichen des Kellers gelesen und folgendermaßen verändert werden:
  - entweder bleibt der Keller unverändert *oder*
  - das oberste Zeichen des Kellers wird entfernt *und* evtl. durch eine Sequenz von anderen Zeichen ersetzt.

  An anderen Stellen darf der Keller nicht gelesen oder verändert werden.

Kellerautomaten

Schematische Darstellung eines Kellerautomaten:



Kellerautomaten

Sei $\Delta = \{a, b, c, d\}$ und

$$L = \{a_1 a_2 \dots a_n \$ a_n \dots a_2 a_1 \mid a_i \in \Delta\}.$$

Ein Kellerautomat erkennt diese Sprache folgendermaßen:

- Ein Wort w wird von links nach rechts eingelesen.
- Der Automat hat zwei Zustände:

  Zustand 1: Ersten Teil des Wortes speichern.

  Zustand 2: Zweiten Teil des Wortes überprüfen.

Kellerautomaten

Zustand 1:

- Solange \$ noch nicht erreicht ist: jedes eingelesene Symbol wird als Großbuchstabe auf den Keller gelegt ($a \rightsquigarrow A$, $b \rightsquigarrow B$, ...).
- Wenn \$ eingelesen wird: Keller bleibt unverändert und Automat wechselt in Zustand 2.

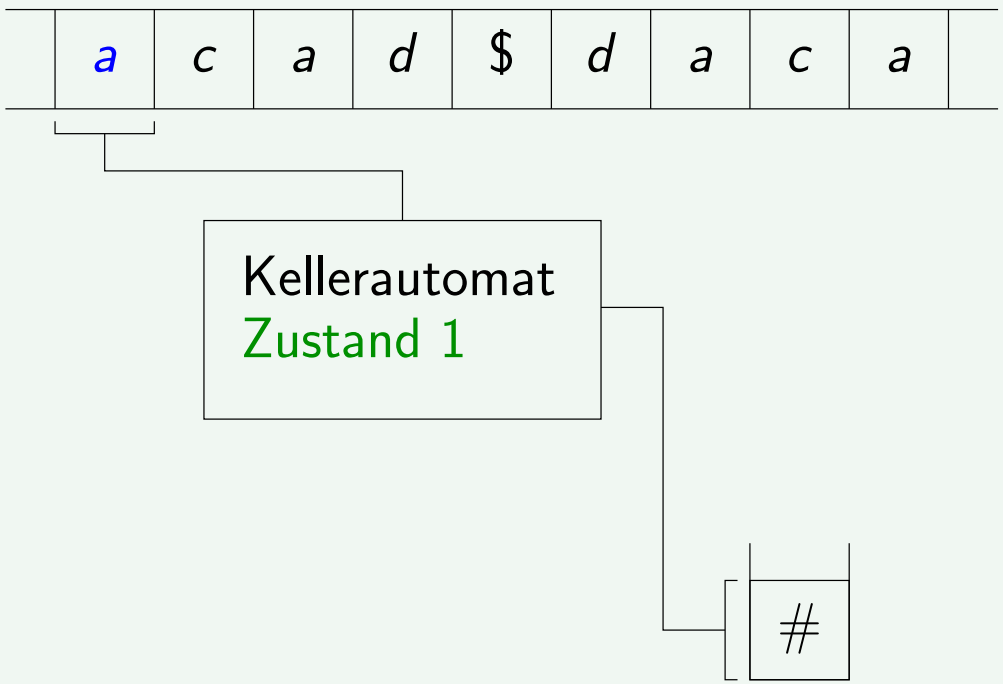
Kellerautomaten

Zustand 2:

- Für jedes neu eingelesene Zeichen wird überprüft, ob der passende Großbuchstabe auf dem Keller liegt. Dieser wird dann entfernt.
- Falls irgendwann keine Übereinstimmung festgestellt wird: Kellerautomat blockiert und das Wort wird nicht akzeptiert.
- Falls immer Übereinstimmung herrscht: auch das Kellerbodenzeichen # wird entfernt und der Automat akzeptiert mit leerem Keller.

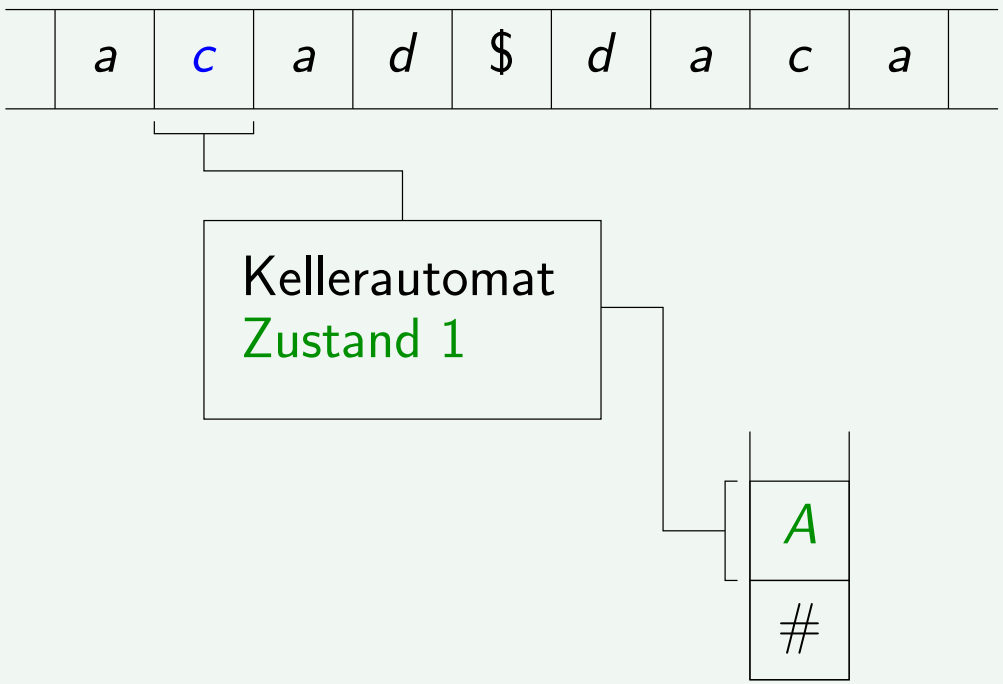
Kellerautomaten

Simulation



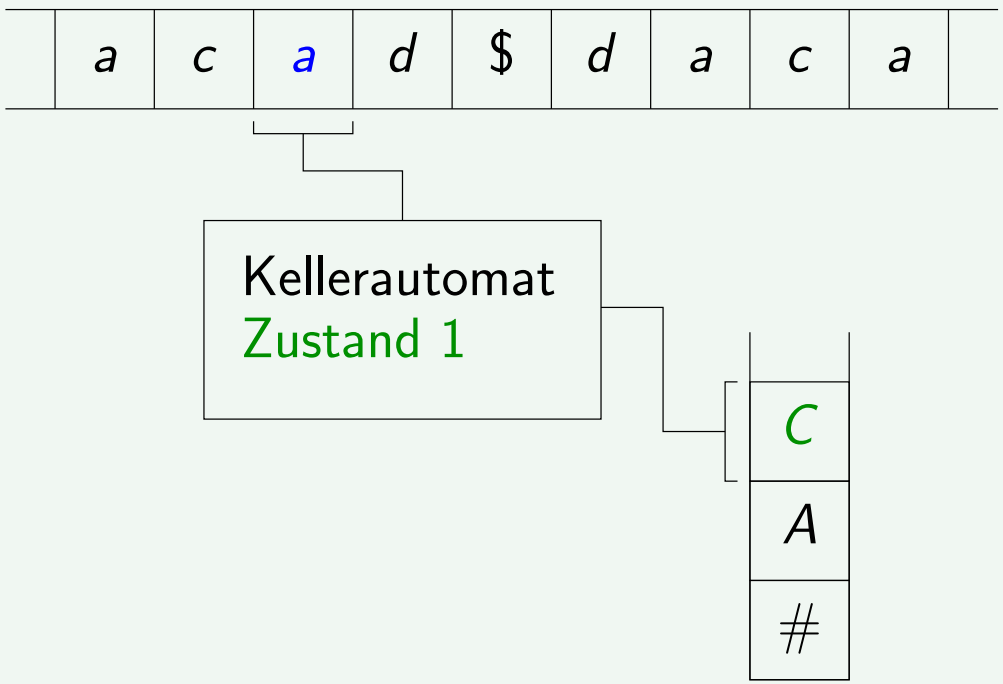
Kellerautomaten

Simulation



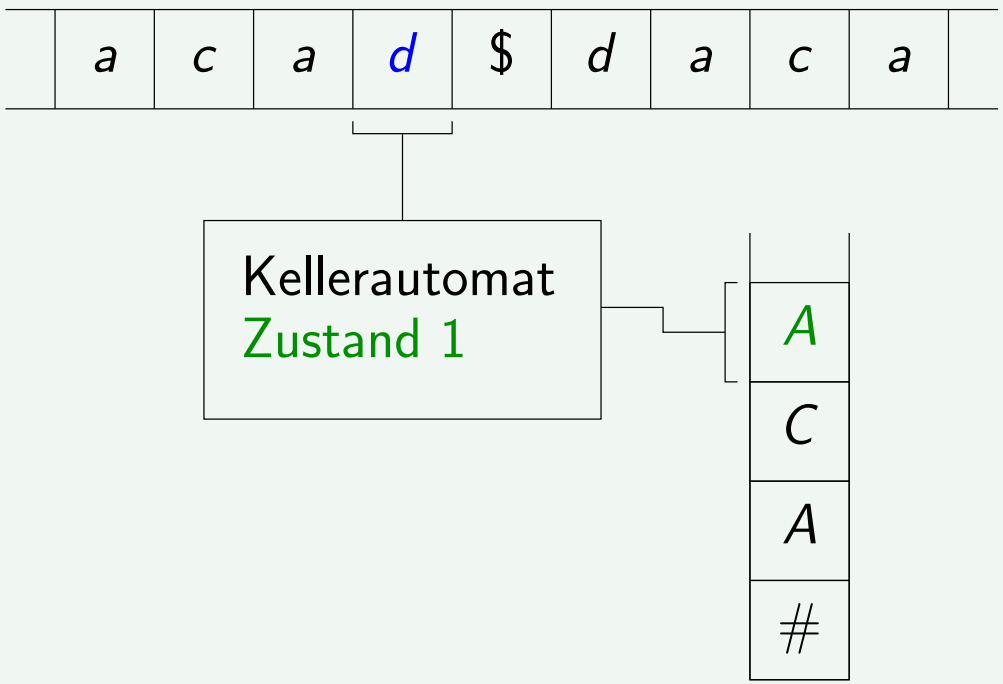
Kellerautomaten

Simulation



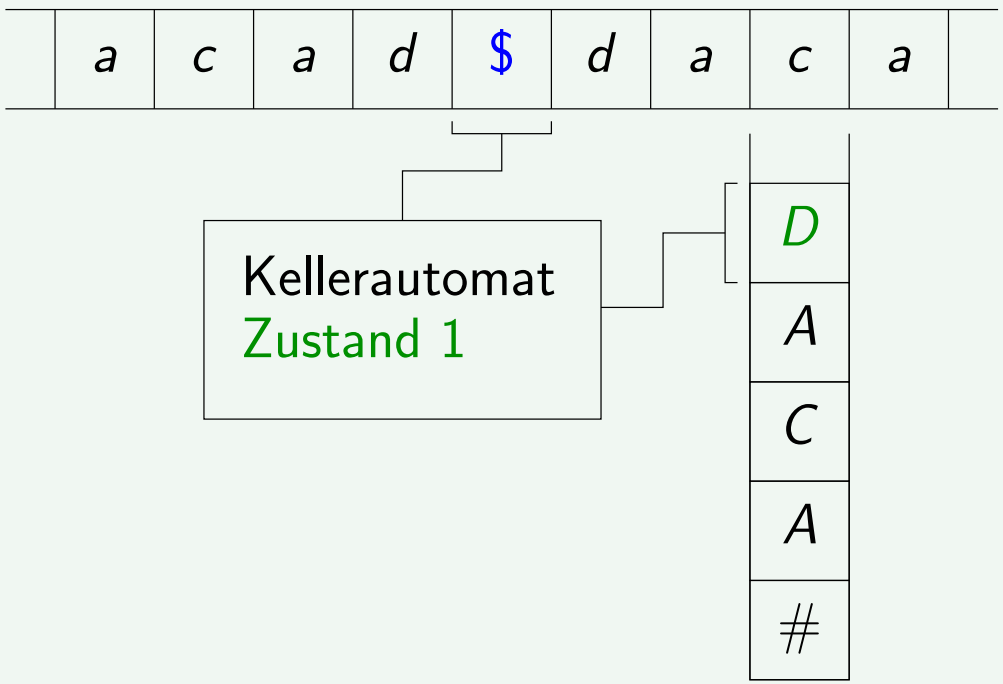
Kellerautomaten

Simulation



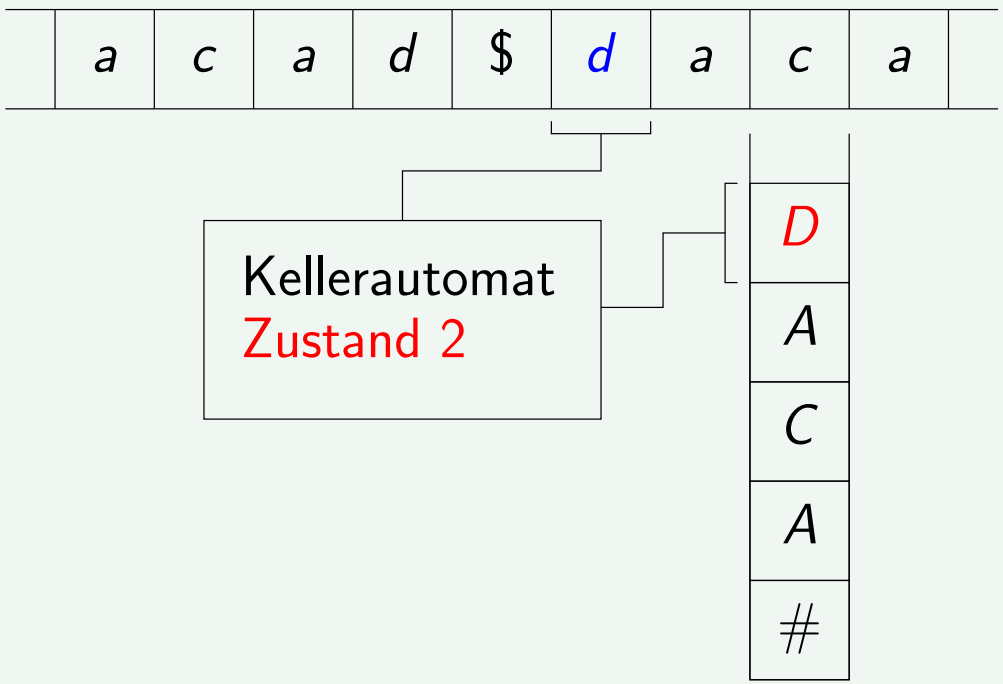
Kellerautomaten

Simulation



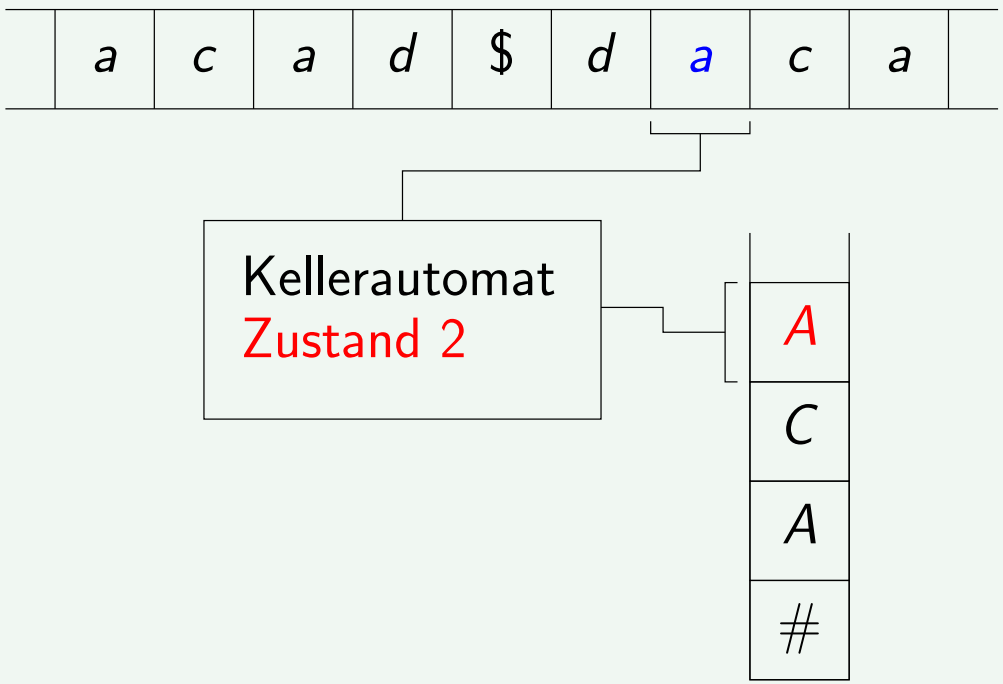
Kellerautomaten

Simulation



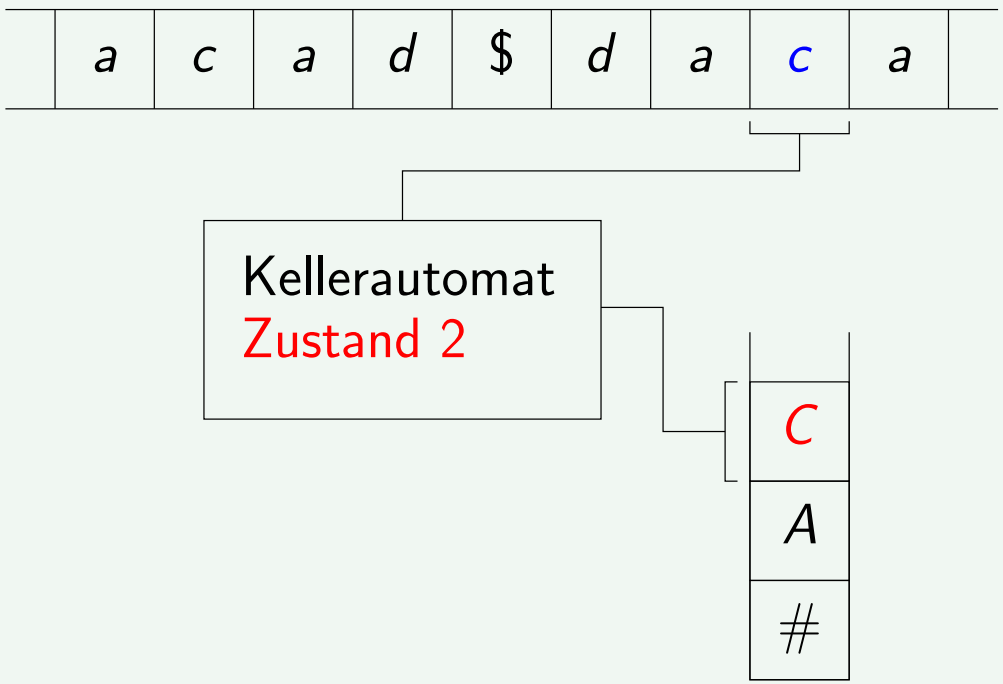
Kellerautomaten

Simulation



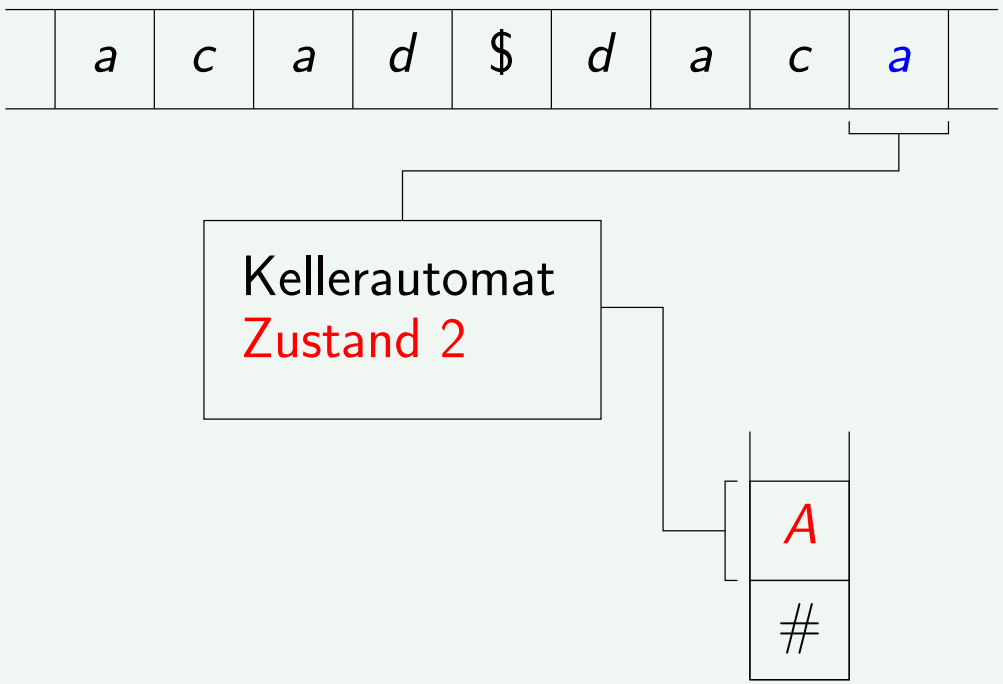
Kellerautomaten

Simulation



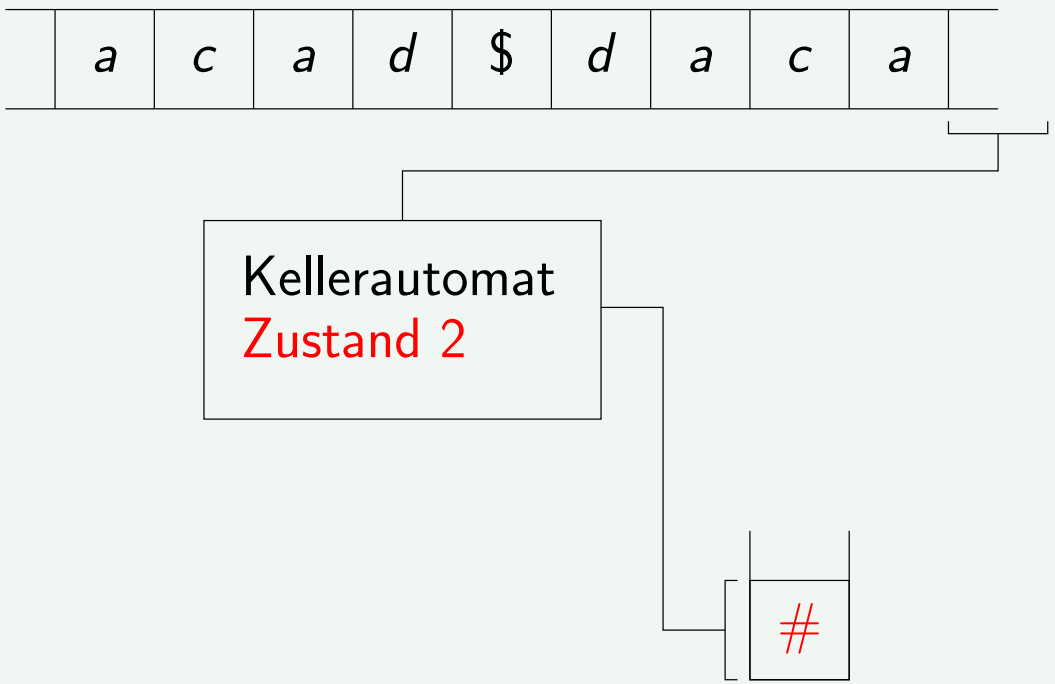
Kellerautomaten

Simulation



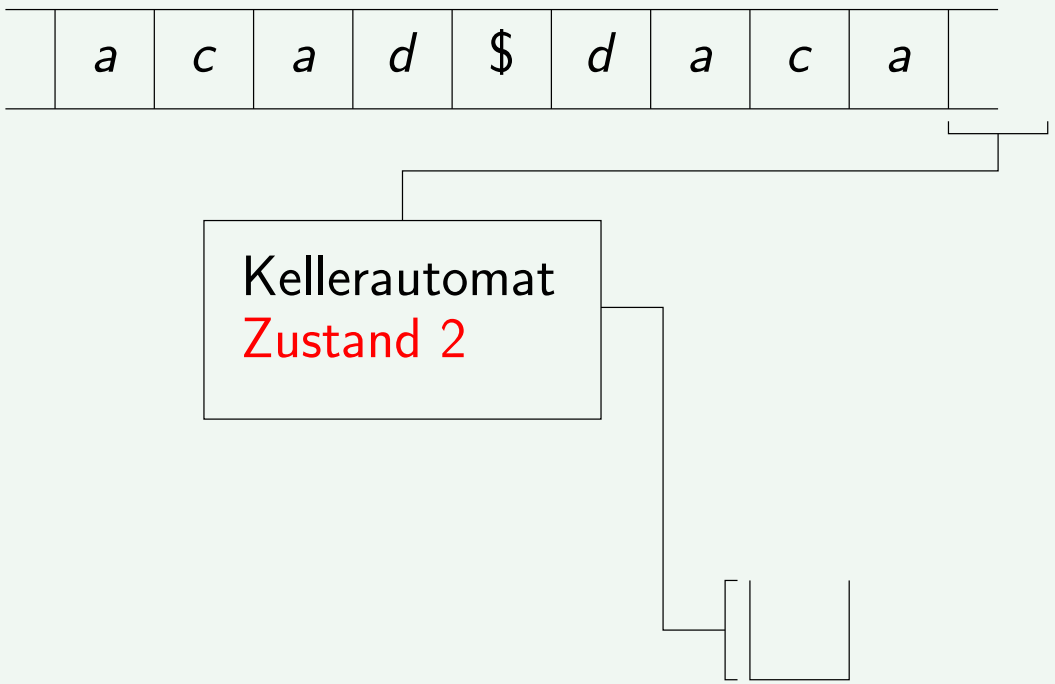
Kellerautomaten

Simulation



Kellerautomaten

Simulation



Kellerautomaten

Kellerautomat (Definition)

Ein (nichtdeterministischer) Kellerautomat M ist ein 6-Tupel $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$, wobei

- Z die Menge der Zustände,
- Σ das Eingabealphabet (mit $Z \cap \Sigma = \emptyset$),
- Γ das Kelleralphabet,
- $z_0 \in Z$ der Startzustand,
- $\delta\colon Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathcal{P}_e(Z \times \Gamma^*)$ die Überführungsfunktion *und*
- $\# \in \Gamma$ das unterste Kellerzeichen oder Kellerbodenzeichen ist.

Kellerautomaten

Bemerkungen zu Kellerautomaten:

- Z, Σ müssen wiederum endliche Mengen sein.
- $\mathcal{P}_e(Z \times \Gamma^*)$ bezeichnet die Menge aller *endlichen* Teilmengen von $Z \times \Gamma^*$.
- Abkürzung: KA (Kellerautomat) oder PDA (pushdown automaton).

Kellerautomaten

- Wir betrachten die Überführungsfunktion

$$\delta\colon Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathcal{P}_e(Z \times \Gamma^*)$$

Falls $(z', B_1 \dots B_k) \in \delta(z, a, A)$, so bedeutet das:

- wenn im Zustand z das Eingabesymbol a gelesen wird und das Zeichen A als oberstes auf dem Keller liegt, *dann*
- wird A vom Keller entfernt und durch $B_1 \dots B_k$ ersetzt (B_1 liegt zuoberst) und der Automat geht in den Zustand z' über.

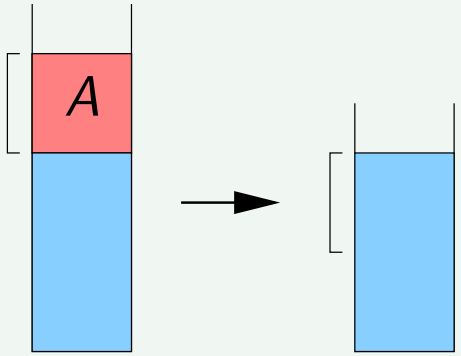
Es kann auch $a = \varepsilon$ gelten. In diesem Fall wird kein Eingabesymbol eingelesen.

Kellerautomaten

Wir betrachten verschiedene Fälle von Werten der Überführungsfunktion δ:

$(z', \varepsilon) \in \delta(z, a, A)$

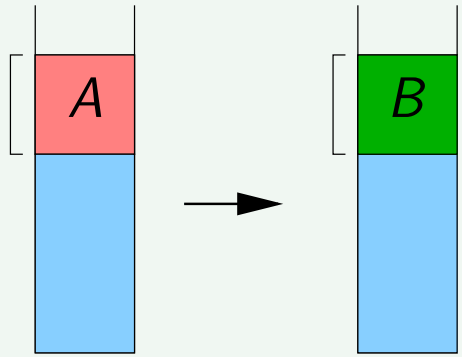
- Zeichen a wird gelesen
- Zustand ändert sich von z nach z'
- Symbol A wird vom Keller entfernt:



Kellerautomaten

$(z', B) \in \delta(z, a, A)$

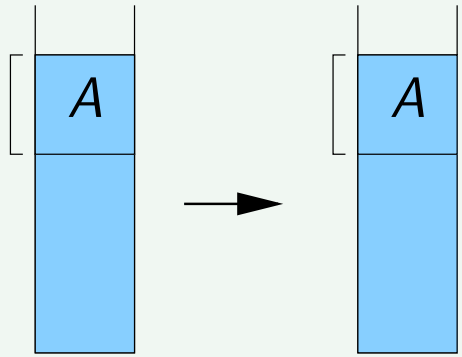
- Zeichen a wird gelesen
- Zustand ändert sich von z nach z'
- Symbol A auf dem Keller wird durch B ersetzt:



Kellerautomaten

$(z', A) \in \delta(z, a, A)$

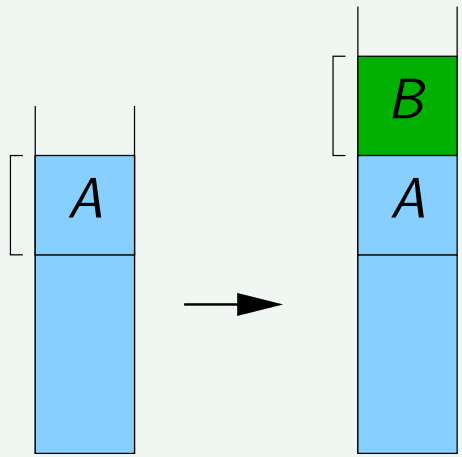
- Zeichen a wird gelesen
- Zustand ändert sich von z nach z'
- Symbol A bleibt auf dem Keller:



Kellerautomaten

$(z', BA) \in \delta(z, a, A)$

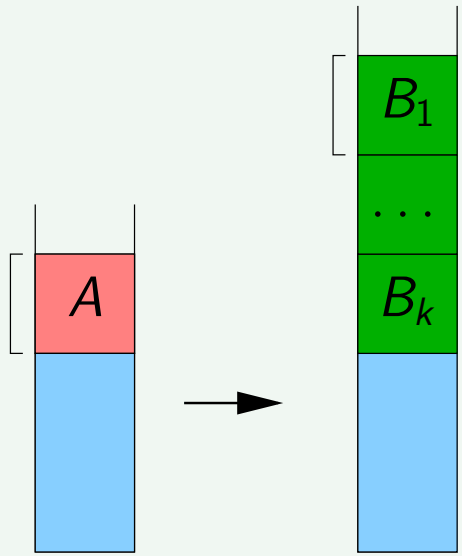
- Zeichen a wird gelesen
- Zustand ändert sich von z nach z'
- Symbol B wird neu auf den Keller gelegt:



Kellerautomaten

$(z', B_1 \dots B_k) \in \delta(z, a, A)$

- Zeichen a wird gelesen
- Zustand ändert sich von z nach z'
- Symbol A wird durch mehrere neue Symbole ersetzt:



Kellerautomaten

- Zu Beginn einer jeden Berechnung enthält der Keller genau das Kellerbodenzeichen #.
- Der Keller ist *nicht* beschränkt und kann beliebig wachsen. Es gibt unendlich viele mögliche Kellerinhalte, das unterscheidet Kellerautomaten von endlichen Automaten.
- Die von uns betrachteten Kellerautomaten akzeptieren immer mit leerem Keller (in diesem Fall gibt es auch keine Übergangsmöglichkeiten mehr). Es gibt aber auch andere Varianten von Kellerautomaten, die mit Endzustand akzeptieren.

Kellerautomaten

Kellerautomat für die Sprache
$L = \{a_1 a_2 \dots a_n \$ a_n \dots a_2 a_1 \mid a_i \in \{a, b\}\}$:

$$M = (\{z_1, z_2\}, \{a, b, \$\}, \{\#, A, B\}, \delta, z_1, \#),$$

wobei δ folgendermaßen definiert ist (wir schreiben $(z, a, A) \to (z', x)$, falls $(z', x) \in \delta(z, a, A)$):

$$\begin{array}{lll}
(z_1, a, \#) \to (z_1, A\#) & (z_1, a, A) \to (z_1, AA) & (z_1, a, B) \to (z_1, AB) \\
(z_1, b, \#) \to (z_1, B\#) & (z_1, b, A) \to (z_1, BA) & (z_1, b, B) \to (z_1, BB) \\
(z_1, \$, \#) \to (z_2, \#) & (z_1, \$, A) \to (z_2, A) & (z_1, \$, B) \to (z_2, B) \\
(z_2, a, A) \to (z_2, \varepsilon) & (z_2, b, B) \to (z_2, \varepsilon) & (z_2, \varepsilon, \#) \to (z_2, \varepsilon)
\end{array}$$

Kellerautomaten

Konfiguration (Definition)

Eine Konfiguration eines Kellerautomaten ist gegeben durch ein Tripel

$$k \in Z \times \Sigma^* \times \Gamma^*.$$

Bedeutung der Komponenten von $k = (z, w, \gamma) \in Z \times \Sigma^* \times \Gamma^*$:

- $z \in Z$ ist der aktuelle Zustand des Kellerautomaten.
- $w \in \Sigma^*$ ist der noch zu lesende Teil der Eingabe.
- $\gamma \in \Gamma^*$ ist der aktuelle Kellerinhalt. Dabei steht das oberste Kellerzeichen ganz links.

Kellerautomaten

Übergänge zwischen Konfigurationen ergeben sich aus der Überführungsfunktion δ:

Konfigurationsübergänge (Definition)

Es gilt

$$(z, aw, A\gamma) \vdash (z', w, B_1 \dots B_k\gamma),$$

falls $(z', B_1 \dots B_k) \in \delta(z, a, A)$, und es gilt

$$(z, w, A\gamma) \vdash (z', w, B_1 \dots B_k\gamma),$$

falls $(z', B_1 \dots B_k) \in \delta(z, \varepsilon, A)$.

Im ersten Fall wird ein Zeichen der Eingabe gelesen, im zweiten jedoch nicht.

Kellerautomaten

Wir definieren $\vdash^*$ als die reflexive and transitive Hülle von $\vdash$.

Damit kann jetzt die von einem Kellerautomaten akzeptierte Sprache definiert werden:

Akzeptierte Sprache (Definition)

Sei $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$ ein Kellerautomat. Dann ist die von M akzeptierte Sprache:

$$N(M) = \{x \in \Sigma^* \mid (z_0, x, \#) \vdash^* (z, \varepsilon, \varepsilon) \text{ für ein } z \in Z\}.$$

Das heißt die akzeptierte Sprache enthält alle Wörter, mit Hilfe derer es möglich ist, den Keller vollständig zu leeren. Da Kellerautomaten jedoch nicht-deterministisch sind, kann es auch Berechnungen für dieses Wort geben, die den Keller nicht leeren.

Kellerautomaten

Ein weiteres Beispiel: ein Kellerautomat für die Sprache

$$L = \{a_1 a_2 \dots a_n a_n \dots a_2 a_1 \mid a_i \in \{a, b\}\}.$$

Idee: anstatt auf das Zeichen \$ zu warten, kann sich der Automat nicht-deterministisch entscheiden, in den Zustand z_2 (= Keller abbauen) überzugehen, sobald das aktuelle Zeichen auf dem Band mit dem Zeichen auf dem Keller übereinstimmt (oder wenn der Keller leer ist).

Kellerautomaten

Veränderte Überführungsfunktion δ (3. Zeile ist geändert):

$$
\begin{array}{lll}
(z_1, a, \#) \to (z_1, A\#) & (z_1, a, A) \to (z_1, AA) & (z_1, a, B) \to (z_1, AB) \\
(z_1, b, \#) \to (z_1, B\#) & (z_1, b, A) \to (z_1, BA) & (z_1, b, B) \to (z_1, BB) \\
(z_1, \varepsilon, \#) \to (z_2, \#) & (z_1, a, A) \to (z_2, \varepsilon) & (z_1, b, B) \to (z_2, \varepsilon) \\
(z_2, a, A) \to (z_2, \varepsilon) & (z_2, b, B) \to (z_2, \varepsilon) & (z_2, \varepsilon, \#) \to (z_2, \varepsilon)
\end{array}
$$

Anmerkung: dieser Kellerautomat nutzt (im Gegensatz zum vorherigen) Nichtdeterminismus, d.h., eine Konfiguration kann mehrere mögliche Nachfolger haben. (Und möglicherweise enden einige Konfigurationsfolgen als Sackgassen und führen nicht dazu, dass der Keller geleert wird.)

Beispiel: Kellerautomat erhält die Eingabe *aabbaa*.

Kellerautomaten

$$(z_1, aabba, \#) \vdash (z_1, abbaa, A\#) \vdash (z_1 bbaa, AA\#)$$

$$\vdash (z_1, baa, BAA\#) \vdash (z_2, baa, BAA\#)$$

An dieser Stelle haben wir uns nichtdeterministisch entscheiden, in den Zustand z_2 zu wechseln, denn wir haben die Mitte des Wortes erreicht. Das kann ein Kellerautomat natürlich nicht ersehen, es gibt also noch eine Reihe weiterer möglicher Folgekonfigurationen, nur mit einem Wechsel zu Zustand z_2 an dieser Stelle können wir aber sicherstellen, dass der Kellerautomat den leeren Keller am Ende der Eingabe erreicht, sie also akzeptiert.

$$\cdots \vdash (z_2, aa, AA\#) \vdash (z_1, a, A\#) \vdash (z_1, \varepsilon, \#) \vdash (z_1, \varepsilon, \varepsilon)$$

Think-Pair-Share: Kellerautomaten

Betrachten Sie den Kellerautomaten
$M = (\{z_1, z_2\}, \{a, b\}, \{A, B, \#\}, \delta, z_1, \#)$ mit folgender Überführungsfunktion:

$$(z_1, a, \#) \to (z_1, A\#) \quad (z_1, a, A) \to (z_1, AA) \quad (z_1, b, A) \to (z_2, B)$$
$$(z_2, b, B) \to (z_2, \varepsilon) \quad (z_2, b, A) \to (z_2, B) \quad (z_2, \varepsilon, \#) \to (z_2, \varepsilon)$$

Welche Sprache akzeptiert M? Erarbeiten Sie zunächst fünf Minuten in Einzelarbeit eine Lösung. Anschließend tauschen Sie sich für weitere fünf Minuten mit ihrem Sitznachbarn aus. Schlussendlich besprechen wir die Lösung im Plenum.
Hinweis: Es kann hilfreich sein, den Kellerautomaten auf einigen Eingaben zu simulieren.

Lösungsvorschlag zu Think-Pair-Share: Kellerautomaten

Die Sprache ist

$$L = \{a^n b^{2n} \mid n > 0\}$$

Deterministisch kontextfreie Sprachen

Wir betrachten nun eine Unterklasse von Kellerautomaten, die dazu verwendet werden können, Sprachen deterministisch und damit effizient zu erkennen.

Deterministischer Kellerautomat (Definition)

En deterministischer Kellerautomat M ist ein 7-Tupel $M = (Z, \Sigma, \Gamma, \delta, z_0, \#, E)$, wobei

- $(Z, \Sigma, \Gamma, \delta, z_0, \#)$ ein Kellerautomat ist,
- $E \subseteq Z$ eine Menge von Endzuständen ist *und*
- die Überführungsfunktion δ deterministisch ist, das heißt: für alle $z \in Z$, $a \in \Sigma$ und $A \in \Gamma$ gilt:

$$|\delta(z, a, A)| + |\delta(z, \varepsilon, A)| \leq 1.$$

Deterministisch kontextfreie Sprachen

Unterschiede zwischen Kellerautomaten und deterministischen Kellerautomaten:

- Deterministische Kellerautomaten haben eine Menge von Endzuständen und akzeptieren mit Endzustand – und *nicht* mit leerem Keller.

  (Bei deterministischen Kellerautomaten ist dies ein Unterschied, für nicht-deterministische Kellerautomaten sind beide Akzeptanzmöglichkeiten gleichwertig.)
- Für jeden Zustand z und jedes Kellersymbol A gilt:
  - *entweder* gibt es höchstens einen ε-Übergang
  - *oder* es gibt für jedes Alphabetsymbol höchstens einen Übergang.

Deterministisch kontextfreie Sprachen

Konfigurationen und Übergänge zwischen Konfiguration bleiben gleich definiert. Konfigurationsfolgen werden jedoch zu linearen Ketten, d.h., es gibt immer höchstens eine Folgekonfiguration.

Diese Tatsache kann für die effiziente Lösung des Wortproblems ausgenutzt werden.

Deterministisch kontextfreie Sprachen

Akzeptierte Sprache bei det. Kellerautomaten (Definition)

Sei $M(Z, \Sigma, \Gamma, \delta, z_0, \#, E)$ ein deterministischer Kellerautomat. Dann ist die von M akzeptierte Sprache:

$$D(M) = \{x \in \Sigma^* \mid (z_0, x, \#) \vdash^* (z, \varepsilon, \gamma) \text{ für ein } z \in E, \gamma \in \Gamma^*\}.$$

Vergleiche dies mit der Definition für nicht-deterministische Kellerautomaten! Bei deterministischen Kellerautomaten ist folgendes anders:

- Der erreichte Zustand z muss ein Endzustand sein.
- Es darf ein Kellerinhalt γ übrigbleiben.

Deterministisch kontextfreie Sprachen

Deterministisch kontextfreie Sprachen

Eine Sprache heißt deterministisch kontextfrei genau dann, wenn sie von einem deterministischen Kellerautomaten akzeptiert wird.

Beispiele:

- Die Sprache $L = \{a_1 a_2 \dots a_n \$ a_n \dots a_2 a_1 \mid a_i \in \Delta\}$ ist deterministisch kontextfrei. (Siehe den entsprechenden Kellerautomaten.)
- Die Sprache $L = \{a_1 a_2 \dots a_n a_n \dots a_2 a_1 \mid a_i \in \Delta\}$ ist jedoch nicht deterministisch kontextfrei. (Ohne Beweis.)

Deterministisch kontextfreie Sprachen

Weitere Bemerkungen:

- Effizienz: Mit Hilfe von deterministischen Kellerautomaten hat man jetzt ein Verfahren zur Lösung des Wortproblems, das die Komplexität $O(n)$ hat. (n ist die Länge des Wortes.)

  Dazu lässt man einfach den Automaten auf dem Wort arbeiten und überprüft, ob man in einen Endzustand gelangt.
- Deterministisch kontextfreie Grammatiken: Da die Syntax von Sprachen einfacher mit Hilfe von Grammatiken als mit Hilfe von Kellerautomaten definiert werden kann, ist es notwendig, die zu deterministischen Kellerautomaten passende Klasse von deterministisch kontextfreien Grammatiken zu definieren.

  Da dies nicht ganz trivial ist, gibt es hierzu mehrere Ansätze. Der bekannteste davon sind die sogenannten $LR(k)$-Grammatiken (siehe Compilerbau und Syntaxanalyse).

Deterministisch kontextfreie Sprachen

Die Abschlusseigenschaften bei deterministisch kontextfreien Sprachen sehen etwas anders aus als bei kontextfreien Sprachen.

Abschluss unter Komplement

Wenn L eine deterministisch kontextfreie Sprache ist, dann ist auch $\overline{L} = \Sigma^* \backslash L$ deterministisch kontextfrei.

(Ohne Beweis)

Deterministisch kontextfreie Sprachen

Kein Abschluss unter Schnitt

Wenn L_1 und L_2 deterministisch kontextfreie Sprachen sind, dann ist $L_1 \cap L_2$ nicht notwendigerweise deterministisch kontextfrei.

Begründung: Die Beispiel-Sprachen aus dem Argument, dass die kontextfreien Sprachen unter Schnitt nicht abgeschlossen sind, sind sogar deterministisch kontextfrei, ihr Schnitt jedoch noch nicht einmal kontextfrei:

$$\begin{aligned} L_1 &= \{a^j b^k c^k \mid j \geq 0, k \geq 0\} \\ L_2 &= \{a^k b^k c^j \mid j \geq 0, k \geq 0\} \end{aligned}$$

Deterministisch kontextfreie Sprachen

Kein Abschluss unter Vereinigung

Wenn L_1 und L_2 deterministisch kontextfreie Sprachen sind, dann ist $L_1 \cup L_2$ nicht notwendigerweise deterministisch kontextfrei.

Begründung: Aus dem Abschluss unter Vereinigung und Komplement würde auch der Abschluss unter Schnitt folgen (wegen $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$).

Deterministisch kontextfreie Sprachen

Deterministisch kontextfreie Sprachen sind unter Schnitt mit regulären Sprachen abgeschlossen.

Abschluss unter Schnitt mit regulären Sprachen

Sei L eine deterministisch kontextfreie Sprache und R eine reguläre Sprache. Dann gilt, dass $L \cap R$ eine deterministisch kontextfreie Sprache ist.

Deterministisch kontextfreie Sprachen

Beweisidee:

Konstruktion eines Kellerautomaten M' für $L \cap R$ aus einem deterministischen Kellerautomaten $M = (Z_1, \Sigma, \Gamma, \delta_1, z_0^1, \#, E_1)$ für L und einem deterministischen endlichen Automaten $A = (Z_2, \Sigma, \delta_2, z_0^2, E_2)$ für R:

$$M' = (Z_1 \times Z_2, \Sigma, \Gamma, \delta', (z_0^1, z_0^2), \#, E_1 \times E_2)$$

mit

- $((z_1', z_2'), B_1 \dots B_k) \in \delta((z_1, z_2), a, A)$, falls $(z_1', B_1 \dots B_k) \in \delta_1(z_1, a, A)$ und $\delta_2(z_2, a) = z_2'$
- $((z_1', z_2), B_1 \dots B_k) \in \delta((z_1, z_2), \varepsilon, A)$, falls $(z_1', B_1 \dots B_k) \in \delta_1(z_1, \varepsilon, A)$

(Analog der Kreuzprodukt-Konstruktion für endliche Automaten.)

Nochmal Abschlusseigenschaften

Mit einer ähnlichen Technik und unter Ausnutzung der Tatsache, dass für allgemeine (nicht-deterministische) Kellerautomaten die Akzeptanz mit leerem Keller analog zur Akzeptanz mit Endzustand ist, lässt sich auch folgendes zeigen:

Abschluss unter Schnitt mit regulären Sprachen

Sei L eine kontextfreie Sprache und R eine reguläre Sprache. Dann gilt, dass $L \cap R$ eine kontextfreie Sprache ist.

Nochmal Abschlusseigenschaften

Zusammenfassung Abschlusseigenschaften:

Abgeschlossen unter	Reguläre Spr.	Det. kfr. Spr.	Kfr. Sprachen
Vereinigung	✓	✗	✓
Konkatenation	✓	✗	✓
Kleene-Stern	✓	✗	✓
Schnitt	✓	✗	✗
Schnitt mit reg. Spr.	✓	✓	✓
Komplement	✓	✓	✗

Fragestellungen zu dieser Vorlesungseinheit

Zum Ende einer jeden Vorlesungseinheit betrachten wir im Regelfall drei Fragestellungen, die mithilfe der in dieser Einheit besprochenen Inhalte beantwortet werden sollen. In der darauffolgenden Einheit können zu Beginn mögliche Antworten gesammelt werden.

Fragen zur zehnten Vorlesungseinheit

- Wovon ist der Übergang eines Kellerautomaten abhängig?
- Wann akzeptiert ein nichtdeterministischer Kellerautomat ein Wort, wann ein deterministischer?
- Wie zeigt man, dass deterministisch kontextfreie Sprachen unter Schnitt mit regulären Sprachen abgeschlossen sind?

Kellerautomaten

Wir müssen nun noch zeigen, dass man mit Kellerautomaten wirklich *genau* die kontextfreien Sprachen akzeptieren kann.

Kontextfreie Grammatiken $\rightarrow$ Kellerautomaten (Satz)

Zu jeder kontextfreien Grammatik G gibt es einen Kellerautomaten M mit $L(G) = N(M)$.

Kellerautomaten

Beweisidee:

1. Verwende den Keller zur Simulation der Grammatik. Leite ein Wort der Sprache auf dem Keller ab (nicht-deterministisches Raten) und überprüfe dann, ob dieses Wort mit dem Wort in der Eingabe übereinstimmt.
2. Problem: der Keller darf nicht beliebig verwendet werden, man kann immer nur das oberste Kellersymbol ersetzen.

   Lösung: Entferne die bereits fertig abgeleiteten Teile des Wortes auf dem Keller, indem sie mit der Eingabe verglichen und bei Übereinstimmung weggenommen werden.
3. Damit kann man erreichen, dass immer wieder eine Variable zuoberst auf dem Keller liegt und abgeleitet werden kann.

Kellerautomaten

Formaler:

sei $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik. Dann definieren wir einen Kellerautomaten

$$M = (\{z\}, \Sigma, V \cup \Sigma, \delta, z, S)$$

mit einem Zustand z und Kelleralphabet $V \cup \Sigma$. Das Startsymbol S ist das Kellerbodenzeichen.

Überführungsfunktion δ:

- Für jede Regel $(A \to \alpha) \in P$ mit $\alpha \in (V \cup \Sigma)^*$ nehme (z, α) in die Menge $\delta(z, \varepsilon, A)$ auf.
  (Ableitungsschritt auf dem Keller ohne Lesen der Eingabe)
- Außerdem nehme (z, ε) in $\delta(z, a, a)$ auf.
  (Vergleichen von Kellerinhalt und Eingabe)

Kellerautomaten

Wir betrachten folgende kontextfreie Grammatik mit dem zweielementigen Alphabet $\Sigma = \{[,]\}$, die korrekte Klammerstrukturen erzeugt:

$$S \to [S]S \mid \varepsilon$$

Aufgabe: wandle diese Grammatik in einen Kellerautomaten um und akzeptiere damit das Wort [[ ]] [ ].

Kellerautomaten

Kellerautomat für Beispiel-Grammatik:

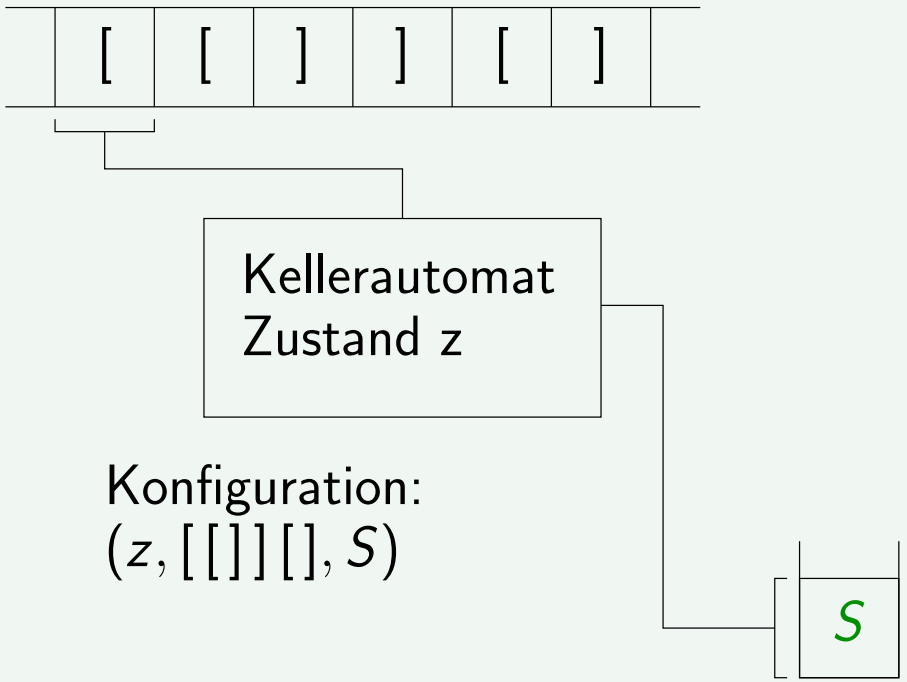
$$M = (\{z\}, \Sigma, \{S\} \cup \Sigma, \delta, z, S)$$

mit folgender Überführungsfunktion δ:

$$\begin{aligned}(z, \varepsilon, S) &\to (z, [S]S)\\ (z, \varepsilon, S) &\to (z, \varepsilon)\\ (z, [\,, [\,) &\to (z, \varepsilon)\\ (z,]\,,]\,) &\to (z, \varepsilon)\end{aligned}$$

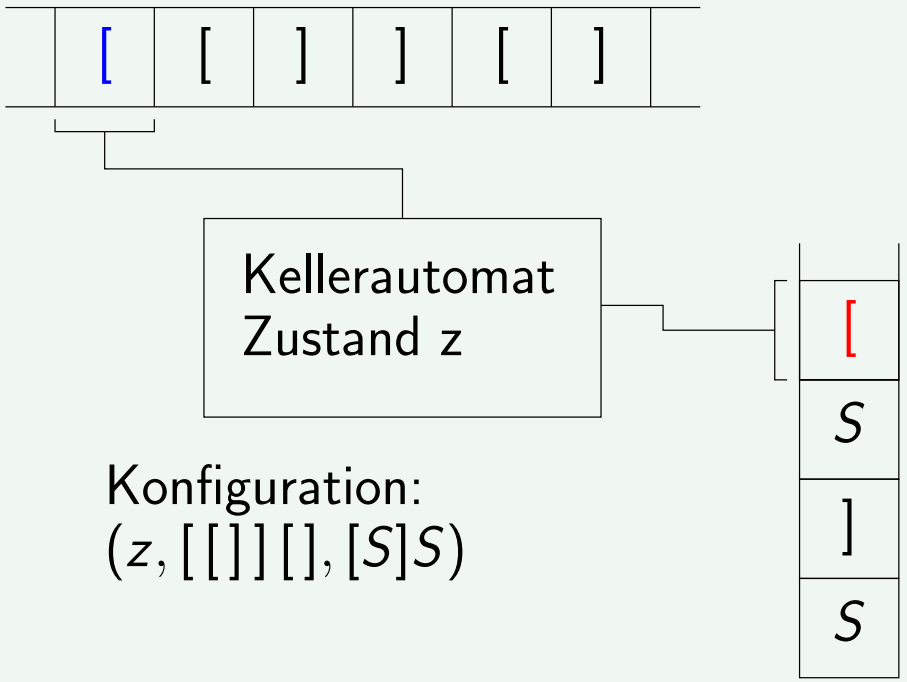
Kellerautomaten

Simulation des KA auf dem Wort [[]][]



Kellerautomaten

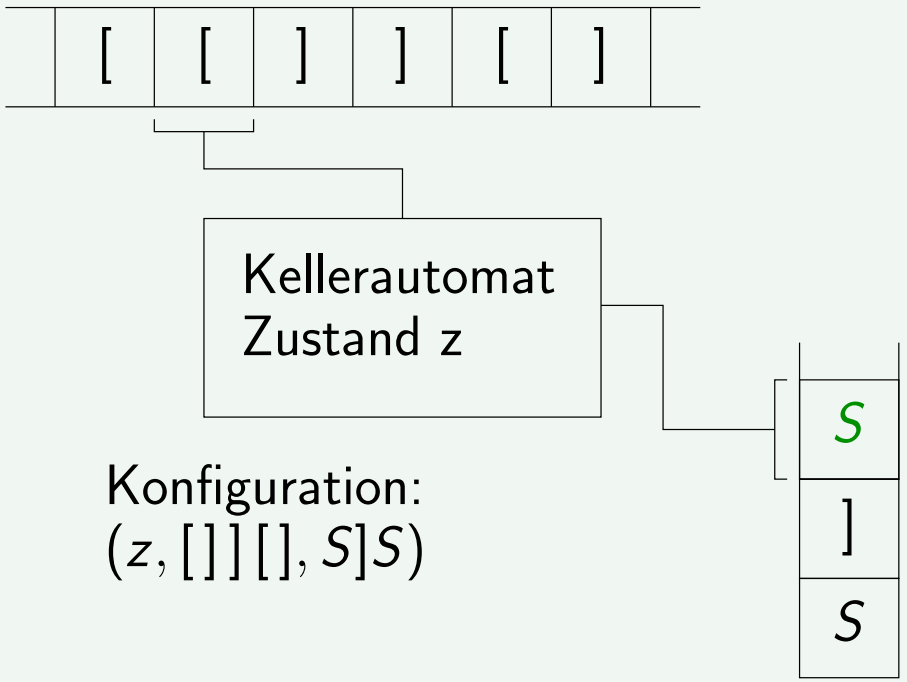
Simulation des KA auf dem Wort [[]][]



Konfiguration:
$(z, [[]][], [S]S)$

Kellerautomaten

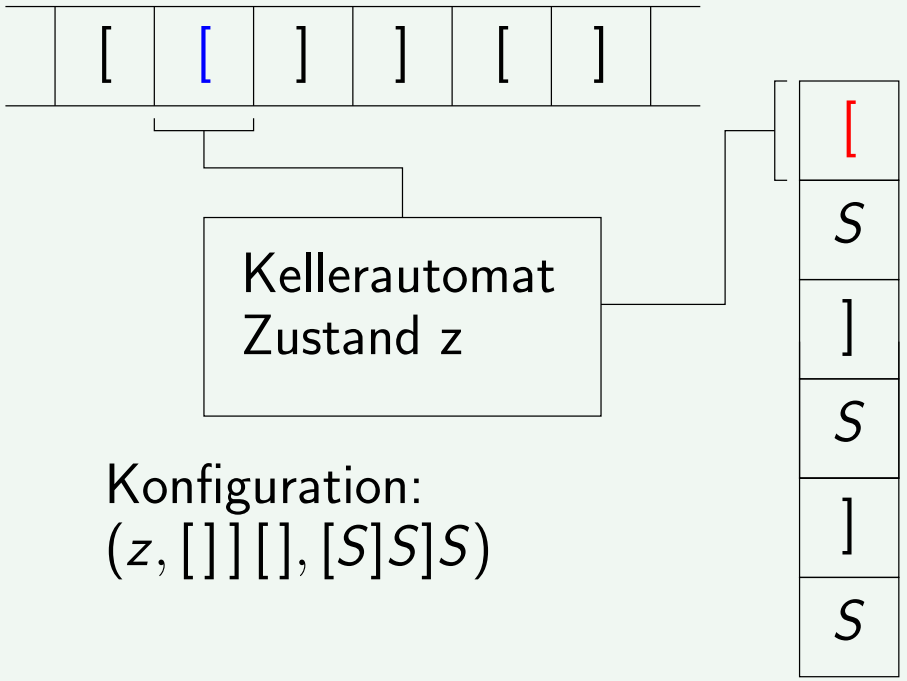
Simulation des KA auf dem Wort [[]][]



Konfiguration:
$(z, [][][], S]S)$

Kellerautomaten

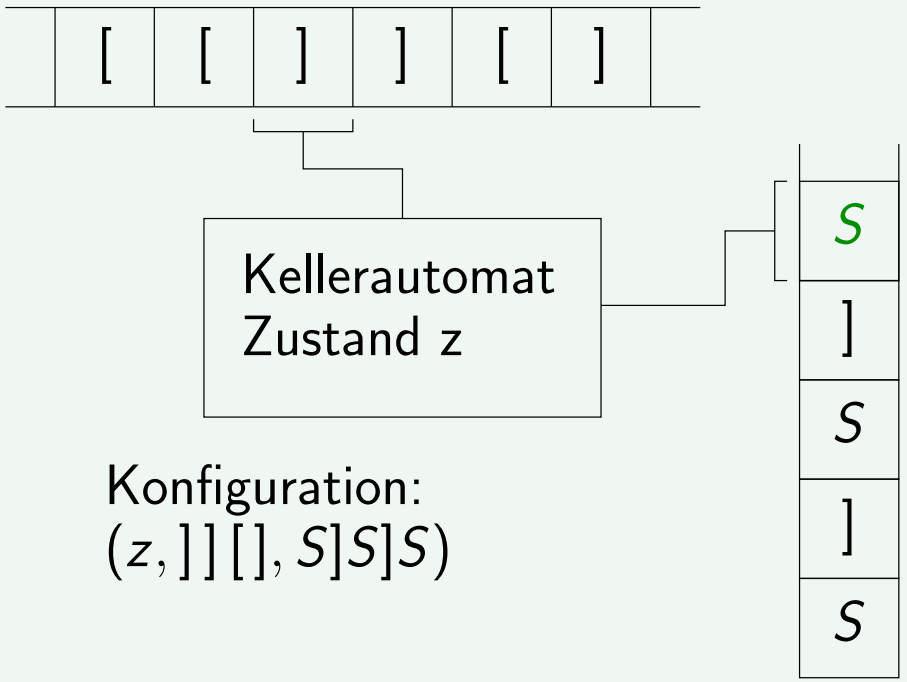
Simulation des KA auf dem Wort [[]][]



Konfiguration:
$(z, [][], [S]S]S)$

Kellerautomaten

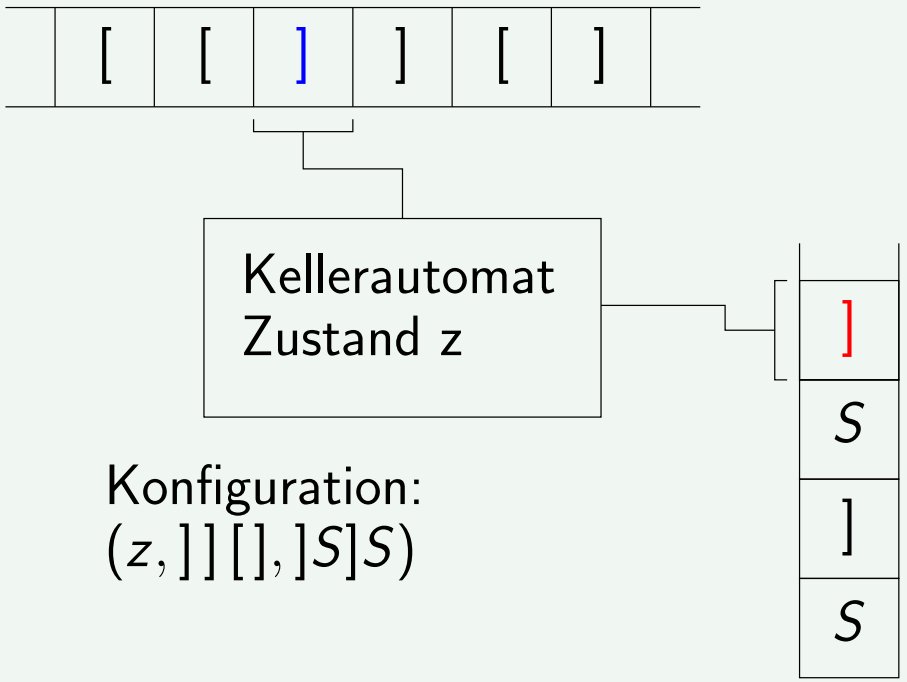
Simulation des KA auf dem Wort [[]][]



Konfiguration:
$(z,]][], S]S]S)$

Kellerautomaten

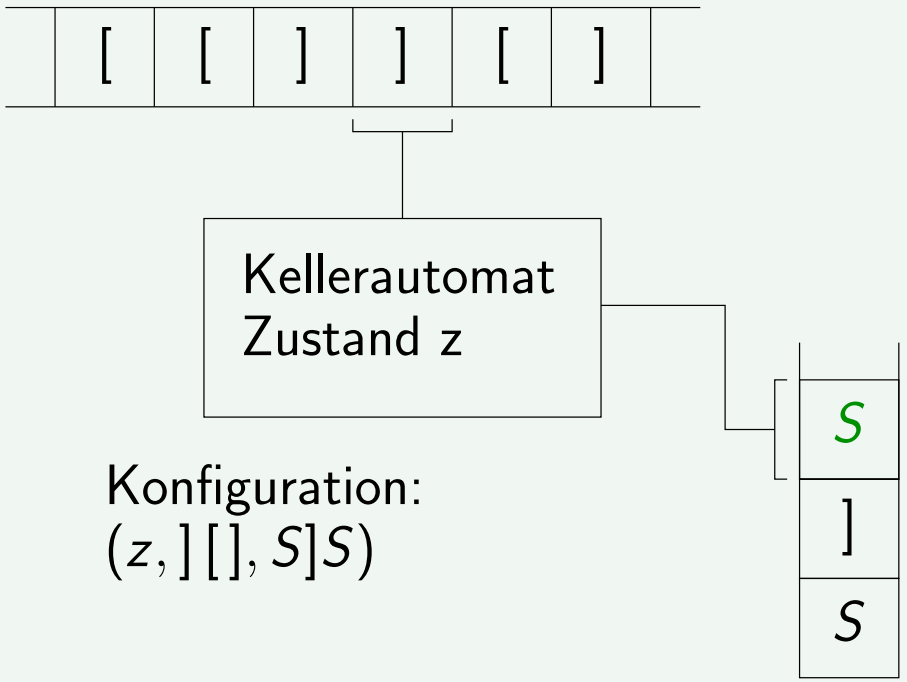
Simulation des KA auf dem Wort [[]][]



Konfiguration:
$(z,]][],]S]S)$

Kellerautomaten

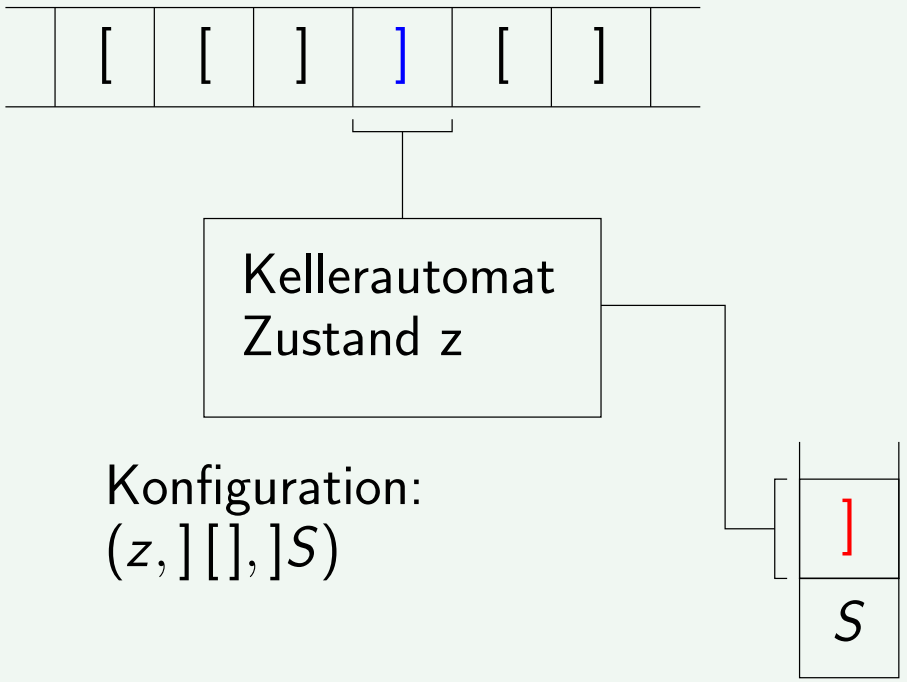
Simulation des KA auf dem Wort [[]][]



Konfiguration:
$(z,][], S]S)$

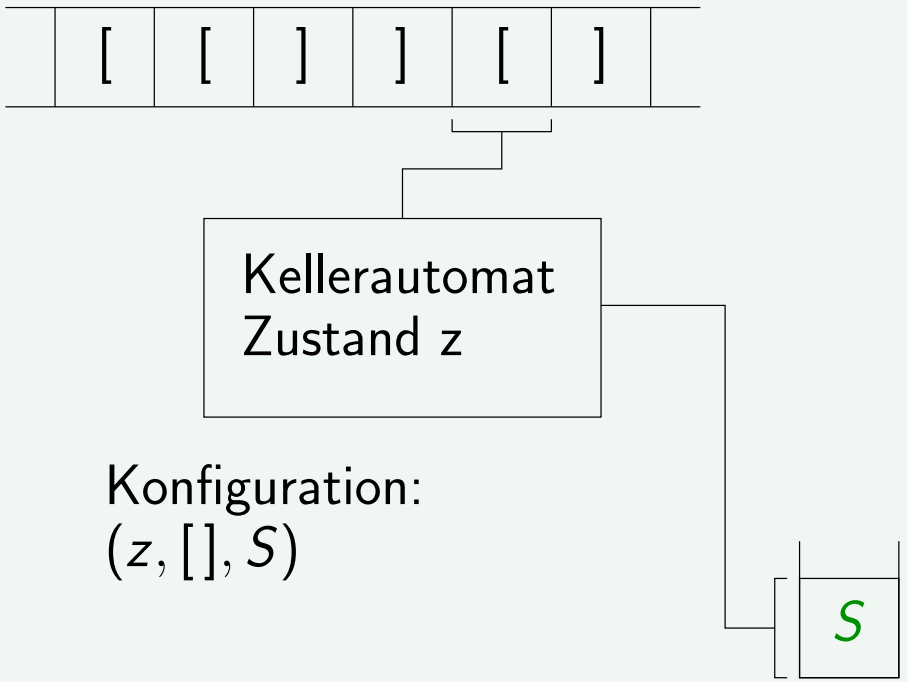
Kellerautomaten

Simulation des KA auf dem Wort [[]][]



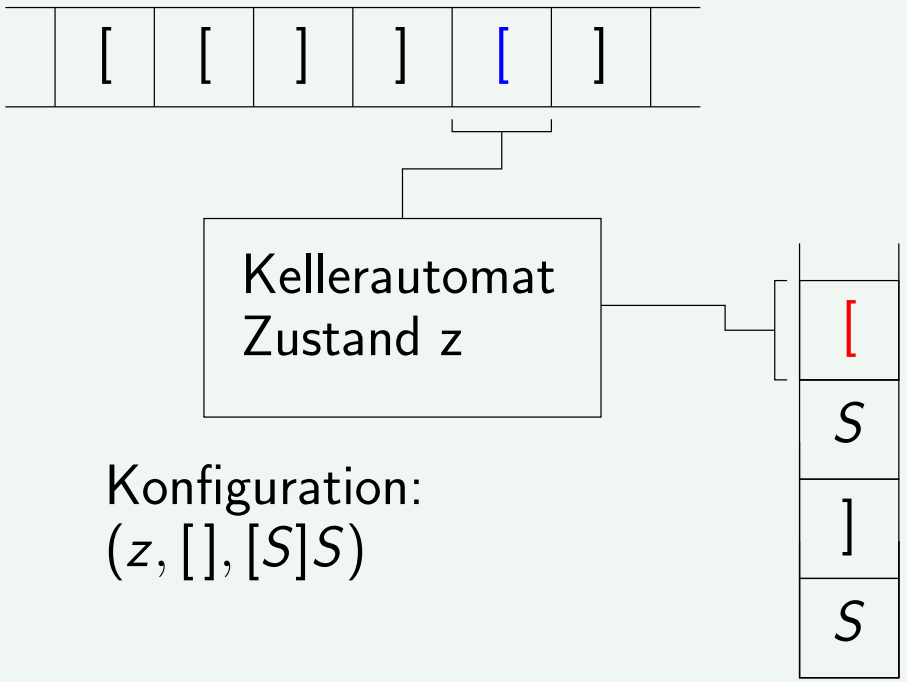
Kellerautomaten

Simulation des KA auf dem Wort [[]][]



Kellerautomaten

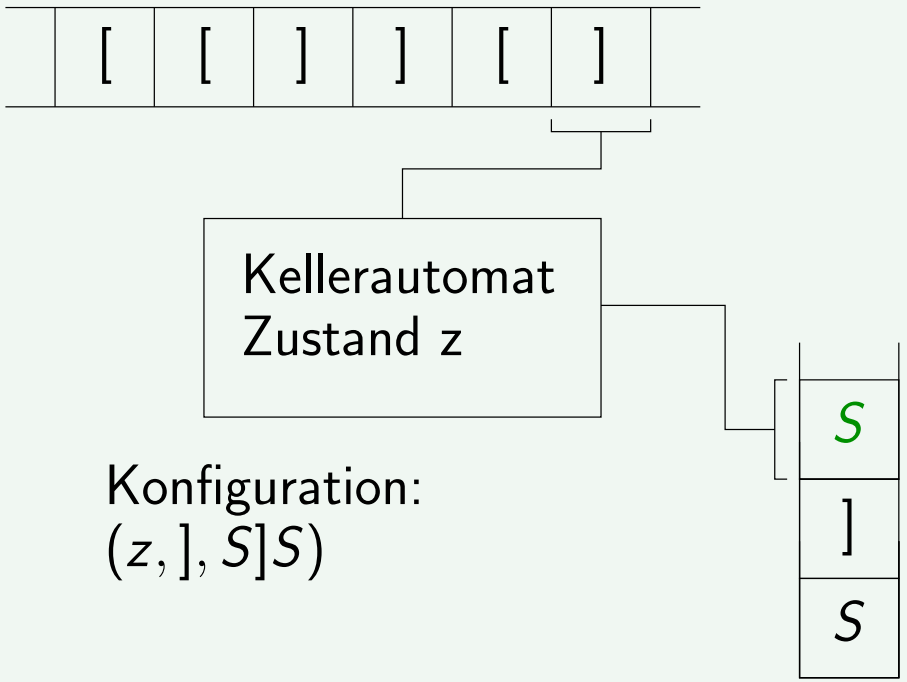
Simulation des KA auf dem Wort [[]][]



Konfiguration:
$(z, [], [S]S)$

Kellerautomaten

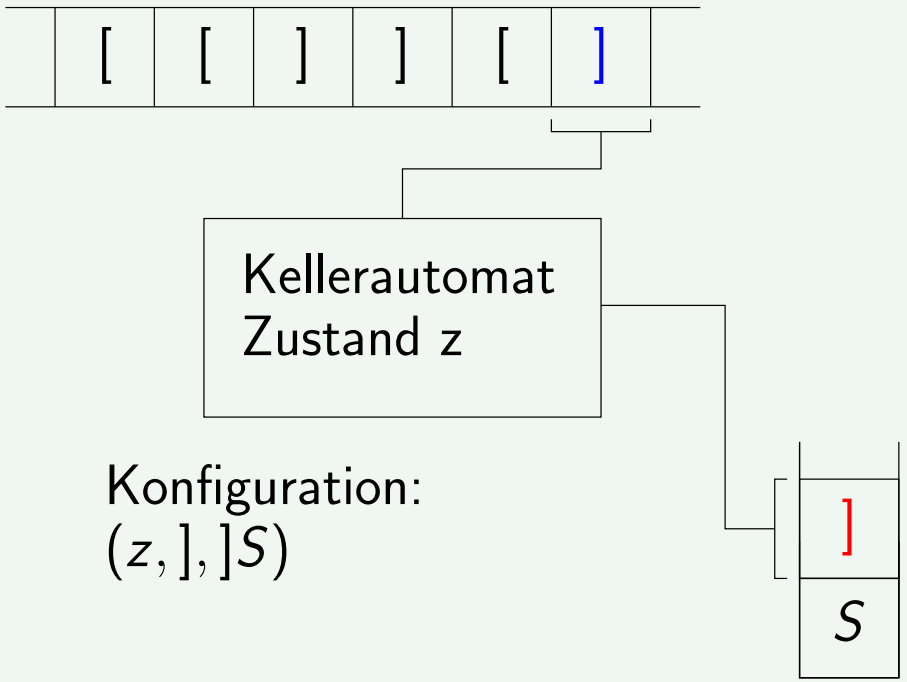
Simulation des KA auf dem Wort [[]][]



Konfiguration:
$(z,], S]S)$

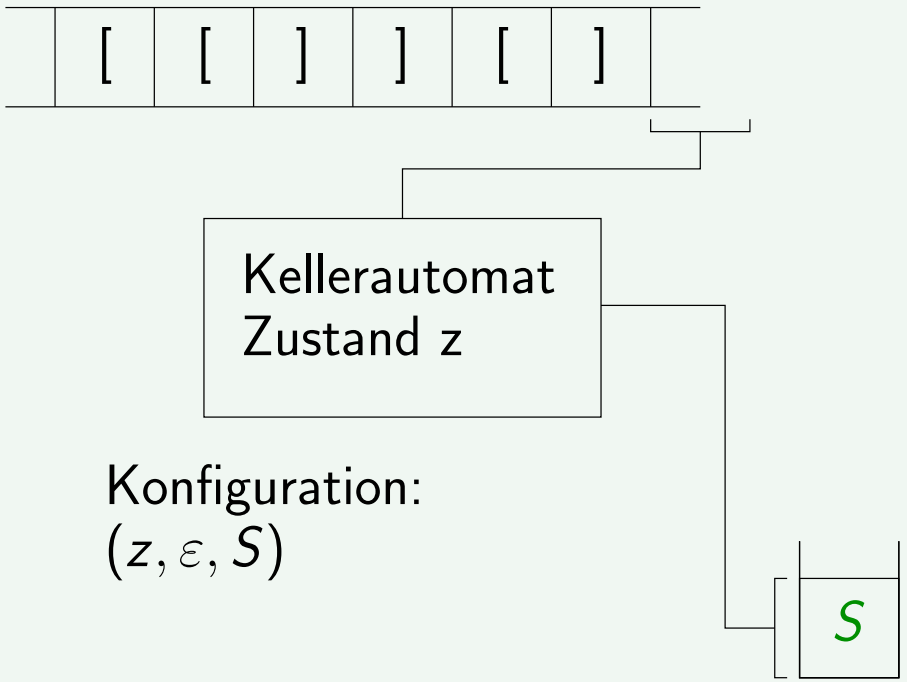
Kellerautomaten

Simulation des KA auf dem Wort [[]][]



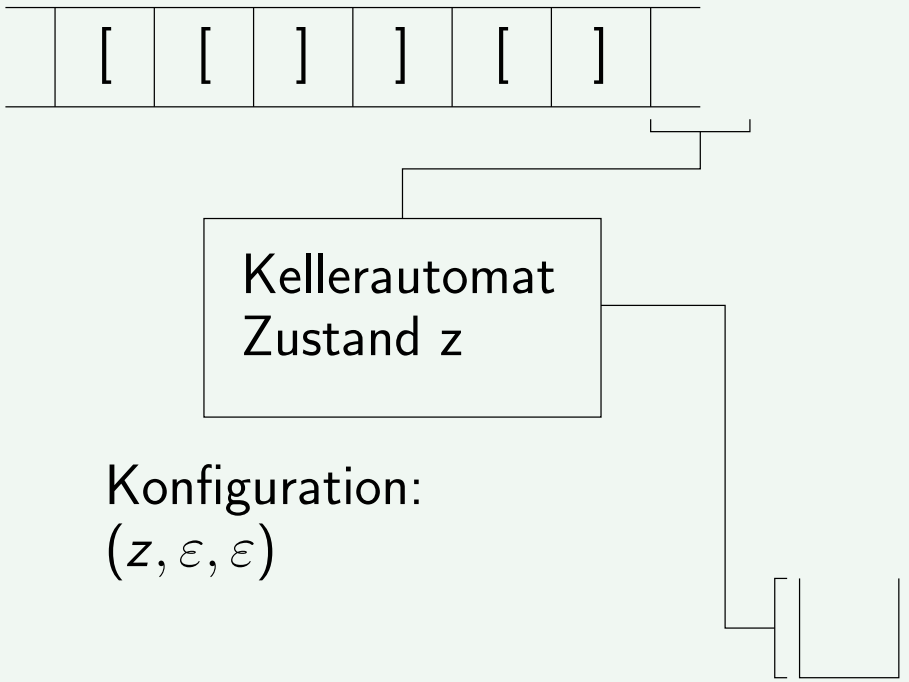
Kellerautomaten

Simulation des KA auf dem Wort [[]][]



Kellerautomaten

Simulation des KA auf dem Wort [[]][]



Think-Pair-Share: Transformation von Grammatik in Kellerautomat

Wandeln Sie die folgende Grammatik in einen Kellerautomaten um:

$$G = (\{S, A\}, \{a, b, c\}, P, S)$$

mit folgender Produktionenmenge P:

$$\begin{aligned} S &\rightarrow aAb \mid ab \\ A &\rightarrow S \mid aaSc \end{aligned}$$

Erarbeiten Sie zunächst vier Minuten in Einzelarbeit eine Lösung. Anschließend tauschen Sie sich für weitere vier Minuten mit ihrem Sitznachbarn aus. Schlussendlich besprechen wir die Lösung im Plenum.

Lösungsvorschlag zu Think-Pair-Share: Transformation von Grammatik in Kellerautomat

$$M = (\{z\}, \{a, b, c\}, \{S, A, a, b, c, \}, \delta, z, S)$$

$$\begin{aligned}
(z, \varepsilon, S) &\to (z, aAb) \\
(z, \varepsilon, S) &\to (z, ab) \\
(z, \varepsilon, A) &\to (z, S) \\
(z, \varepsilon, A) &\to (z, aaSc) \\
(z, \varepsilon, S) &\to (z, \varepsilon) \\
(z, a, a) &\to (z, \varepsilon) \\
(z, b, b) &\to (z, \varepsilon) \\
(z, c, c) &\to (z, \varepsilon)
\end{aligned}$$

Kellerautomaten

Nun geht es darum zu zeigen, dass es zu jedem Kellerautomaten eine entsprechende kontextfreie Grammatik gibt. (Das ist die schwierigere Richtung.)

Kellerautomaten $\rightarrow$ Kontextfreie Grammatiken (Satz)

Zu jedem Kellerautomaten M gibt es eine kontextfreie Grammatik G mit $N(M) = L(G)$.

Kellerautomaten

Beweisidee:

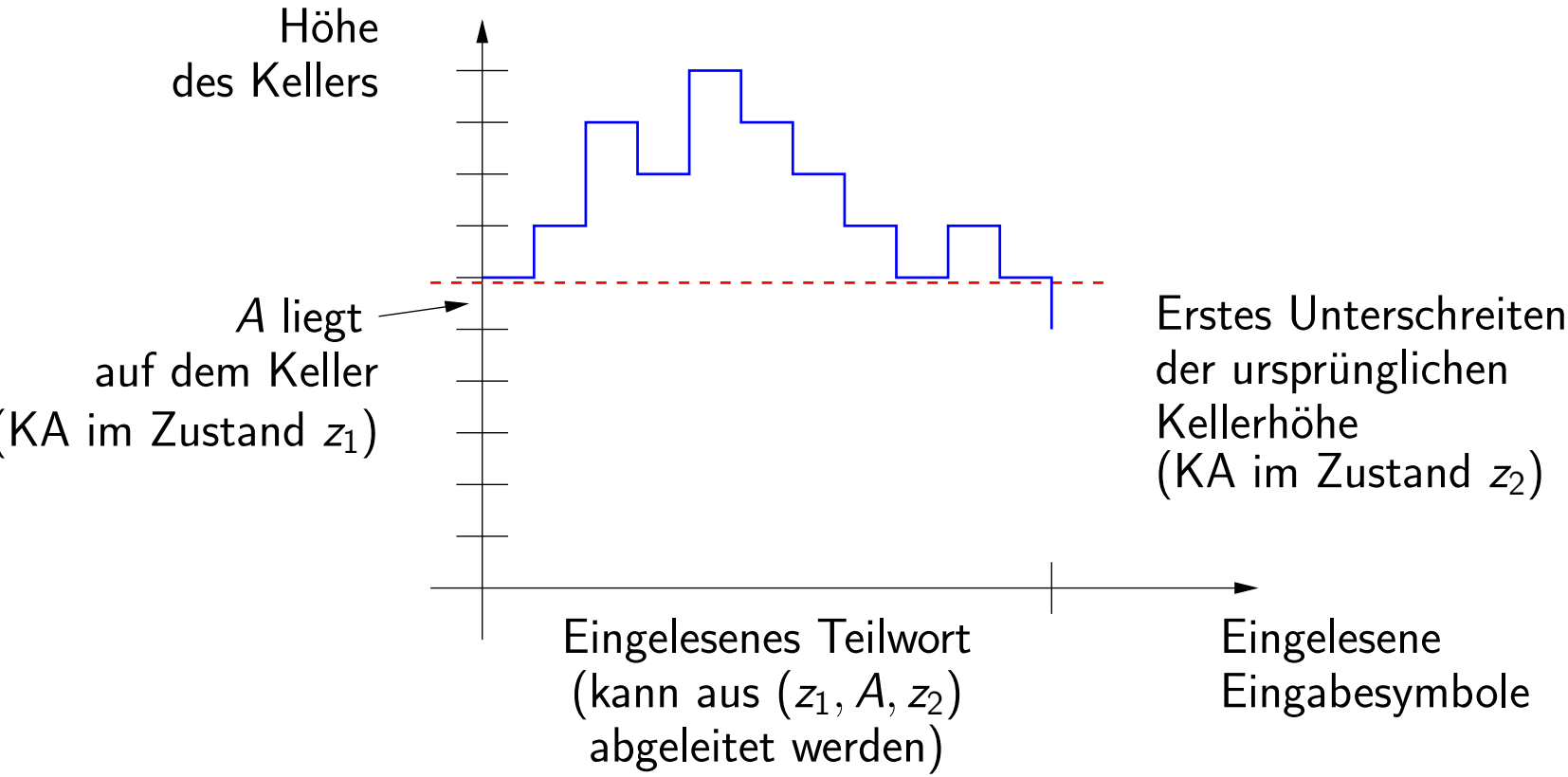
1. Wir wollen beschreiben, welche Wörter man durch Abbauen eines bestimmten Kellersymbols akzeptieren kann. Die vom Automaten akzeptierte Sprache besteht nämlich aus allen Wörtern, die man durch Abbauen von # erzeugen kann.

   Abbauen bedeutet: zwischendurch dürfen weitere Symbole auf den Keller gelegt werden, aber zuletzt muss der Keller um dieses eine Symbol kürzer geworden sein.

2. Die zu erstellende kontextfreie Grammatik besitzt Variablen der Form (z_1, A, z_2) mit der Bedeutung:

   > Aus (z_1, A, z_2) kann man genau die Wörter ableiten, die der Kellerautomat einliest, wenn er im Zustand z_1 startet, A vom Keller abbaut und im Zustand z_2 aufhört.

Kellerautomaten



Zwischendurch kann A durch ein anderes Symbol ersetzt werden.
Die ursprüngliche Kellerhöhe wird jedoch nicht unterschritten.

Kellerautomaten

Bedeutung der Nicht-Terminale (z_1, A, z_2):

$$(z_1, A, z_2) \Rightarrow^* x \qquad \Longleftrightarrow \qquad (z_1, x, A) \vdash^* (z_2, \varepsilon, \varepsilon)$$

Gegeben sei ein Kellerautomat $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$. Wir definieren eine Grammatik $G = (V, \Sigma, P, S)$ wie folgt:

Variable: $V = \{S\} \cup Z \times \Gamma \times Z$ (Eigene Startvariable und Variablen der Form (z_1, A, z_2))

Kellerautomaten

Produktionen folgender Form:

$$S \to (z_0, \#, z) \qquad \text{für alle } z \in Z$$

(Entfernen des Kellerbodenzeichens)

$$(z, A, z') \to a \qquad \text{falls } (z', \varepsilon) \in \delta(z, a, A)$$

(Symbol A kann – bei Einlesen von a – sofort entfernt werden)

$$(z, A, z') \to a(z_1, B_1, z_2)(z_2, B_2, z_3)\dots(z_k, B_k, z')$$

falls $(z_1, B_1 \dots B_k) \in \delta(z, a, A)$, $z', z_2, \dots, z_k \in Z$

(Symbol A wird bei Einlesen von a durch $B_1 \dots B_k$ ersetzt, diese müssen über Zwischenzustände $z_1, \dots, z_k$ entfernt werden)

Kellerautomaten

Produktionen folgender Form:

$$(z, A, z') \quad \rightarrow \quad a(z_1, B_1, z_2)(z_2, B_2, z_3) \dots (z_k, B_k, z')$$

Die Idee ist hier die folgende: Wenn das Symbol A auf dem Keller entfernt und durch die Symbole $B_1 \dots B_k$ ersetzt wird, ist es notwendig, $B_1 \dots B_k$ durch weitere Transitionen abzubauen. Dabei werden irgendwelche Zwischenzustände $z_2 \dots z_k$ erreicht, bevor schlussendlich B_k abgebaut und der Zustand z' erreicht wird. Es gibt daher eine Überführungsregel für jede mögliche Wahl von $z_2 \dots z_k$.
Wenn man ein Wort ableiten möchte, muss man also bei der Simulation eines Schrittes des Kellerautomaten bereits „raten“, welche Zwischenzustände bei der Elimination der neuen Kellerzeichen erreicht werden.

Kellerautomaten

Beispiel: Wir betrachten den Kellerautomaten

$$M = (\{z_1, z_2\}, \{a, b\}, \{A, \#\}, \delta, z_1, \#)$$

mit folgender Überführungsfunktion δ:

$$\begin{array}{rcl} (z_1, \varepsilon, \#) & \to & (z_2, \varepsilon) \\ (z_1, a, \#) & \to & (z_1, AA) \\ (z_1, a, A) & \to & (z_1, AAA) \\ (z_1, b, A) & \to & (z_2, \varepsilon) \\ (z_2, b, A) & \to & (z_2, \varepsilon) \end{array}$$

Es gilt: $N(M) = \{a^n b^{2n} \mid n \geq 0\}$.

Aufgabe: Umwandlung von M in eine kontextfreie Grammatik.

Kellerautomaten

Kontextfreie Grammatik für den Beispiel-Kellerautomaten:

$$G = (V, \Sigma, P, S)$$

mit folgender Variablenmenge

$$\begin{aligned} V &= \{S, (z_1, \#, z_1), (z_1, \#, z_2), (z_2, \#, z_1), (z_2, \#, z_2), \\ &\quad (z_1, A, z_1), (z_1, A, z_2), (z_2, A, z_1), (z_2, A, z_2)\} \end{aligned}$$

...

Kellerautomaten

. . . und mit folgender Produktionenmenge P:

$$
\begin{aligned}
S &\to (z_1, \#, z_1) \mid (z_1, \#, z_2) \\
(z_1, \#, z_2) &\to \varepsilon \\
(z_1, A, z_2) &\to b \\
(z_2, A, z_2) &\to b \\
(z_1, \#, z_1) &\to a(z_1, A, z)(z, A, z_1) \\
(z_1, \#, z_2) &\to a(z_1, A, z)(z, A, z_2) \\
(z_1, A, z_1) &\to a(z_1, A, z)(z, A, z')(z', A, z_1) \\
(z_1, A, z_2) &\to a(z_1, A, z)(z, A, z')(z', A, z_2)
\end{aligned}
$$

$z, z' \in \{z_1, z_2\}$ können jeweils beliebig gewählt werden.

Kellerautomaten

Beispiel-Ableitung des Wortes $aabbbb$:

$$
\begin{aligned}
S &\Rightarrow (z_1, \#, z_2) \\
&\Rightarrow a(z_1, A, z_2)(z_2, A, z_2) \\
&\Rightarrow aa(z_1, A, z_2)(z_2, A, z_2)(z_2, A, z_2)(z_2, A, z_2) \\
&\Rightarrow aab(z_2, A, z_2)(z_2, A, z_2)(z_2, A, z_2) \\
&\Rightarrow aabb(z_2, A, z_2)(z_2, A, z_2) \\
&\Rightarrow aabbb(z_2, A, z_2) \\
&\Rightarrow aabbbb
\end{aligned}
$$

Kellerautomaten

Bemerkung zu den Umwandlungen "Kontextfreie Grammatik $\leftrightarrow$ Kellerautomat":

- Zu jedem Kellerautomaten gibt es immer einen äquivalenten Kellerautomaten mit nur einem Zustand.

  Dazu wandelt man ihn in eine kontextfreie Grammatik um und dann wieder zurück in einen Kellerautomaten. Es wird ausgenutzt, dass bei der Umwandlung in Kellerautomaten immer nur Automaten mit einem Zustand konstruiert werden.

Entscheidbarkeit

Wir betrachten nun noch Probleme für kontextfreie Sprachen und stellen fest, ob sie entscheidbar sind, d.h., ob es entsprechende Verfahren zu ihrer Lösung gibt.

Folgende Probleme sind für kontextfreie Sprachen (repräsentiert durch eine kontextfreie Grammatik oder einen Kellerautomaten) entscheidbar:

Wortproblem bei kontextfreien Sprachen ist entscheidbar

- Wortproblem: Gegeben eine kontextfreie Sprache L und $w \in \Sigma^*$. Gilt $w \in L$?

  Mit dem CYK-Algorithmus in $O(|w|^3)$ Zeit.

Entscheidbarkeit

Leerheitsproblem bei kontextfreien Sprachen ist entscheidbar

- Leerheitsproblem: Gegeben eine kontextfreie Sprache L. Gilt $L = \emptyset$?

  Bestimme alle produktiven Variablen, d.h., alle Variablen A, für die es ein $x \in \Sigma^*$ gibt mit $A \Rightarrow^* x$ (siehe Übungsaufgabe). Die Sprache L ist leer, genau dann wenn das Startsymbol S nicht produktiv ist.

Entscheidbarkeit

Endlichkeitsproblem bei kontextfreien Sprachen ist entscheidbar

- Endlichkeitsproblem: Gegeben eine kontextfreie Sprache L. Ist L endlich?

  Wir gehen davon aus, dass die Grammatik in CNF gegeben ist (sonst überführen wir sie in CNF).

  1. Entferne alle nicht produktiven oder nicht erreichbaren Variablen aus der Grammatik (vgl. Übungen)
  2. Ermittle für jede Variable, welche Variablen in einem oder mehr Schritten ableitbar sind (Fixpunktiteration analog zur Bestimmung erreichbarer Variablen)
  3. Gibt es eine Variable, die von sich selbst aus abgeleitet werden kann, ist die Sprache unendlich, sonst endlich (vgl. Pumping-Lemma für kontextfreie Sprachen: Ist eine Variable von sich selbst aus erreichbar, kann man an dieser Variable pumpen).

Entscheidbarkeit

Folgende Probleme sind für kontextfreie Sprachen nicht entscheidbar, d.h., man kann zeigen, dass es kein entsprechendes Verfahren gibt:

Unentscheidbare Probleme bei kontextfreien Sprachen

- Äquivalenzproblem: Gegeben zwei kontextfreie Sprachen L_1, L_2. Gilt $L_1 = L_2$?
- Schnittproblem: Gegeben zwei kontextfreie Sprachen L_1, L_2. Gilt $L_1 \cap L_2 = \emptyset$?

Bemerkung: In der Vorlesung "Berechenbarkeit und Komplexität" wird es darum gehen, wie man solche Unentscheidbarkeitsresultate zeigen kann.

Entscheidbarkeit

Schnittproblem mit regulären Sprachen ist entscheidbar

Das Schnittproblem ist jedoch entscheidbar, wenn von einer der beiden Sprachen L_1, L_2 bekannt ist, dass sie regulär ist und sie als endlicher Automat gegeben ist.

Entscheidungsverfahren:

1. In diesem Fall kann ein Kellerautomat M konstruiert werden (Konstruktion siehe weiter oben), der $L_1 \cap L_2$ akzeptiert.
2. Der Kellerautomat M kann dann in eine kontextfreie Grammatik G umgewandelt werden.
3. Durch Bestimmung der produktiven Variablen von G kann dann ermittelt werden, ob S nicht-produktiv ist und damit, ob $L_1 \cap L_2$ leer ist.

Entscheidbarkeit

Folgende Probleme sind für deterministisch kontextfreie Sprachen (repräsentiert durch einen deterministischen Kellerautomaten) entscheidbar:

Entscheidbarkeit bei deterministisch kontextfreien Sprachen

- Wortproblem: Gegeben eine deterministisch kontextfreie Sprache L und $w \in \Sigma^*$. Gilt $w \in L$?

  Mit einem deterministischen Kellerautomaten in $O(|w|)$ Zeit.
- Leerheitsproblem: Gegeben eine deterministisch kontextfreie Sprache L. Gilt $L = \emptyset$?

  Siehe das entsprechende Entscheidungsverfahren für kontextfreie Sprachen.

Entscheidbarkeit

Entscheidbarkeit bei deterministisch kontextfreien Sprachen

- Endlichkeitsproblem: Gegeben eine kontextfreie Sprache L. Ist L endlich?

  Siehe das entsprechende Entscheidungsverfahren für kontextfreie Sprachen.

- Äquivalenzproblem: Gegeben zwei deterministisch kontextfreie Sprachen L_1, L_2. Gilt $L_1 = L_2$?

  War lange offen und die Entscheidbarkeit wurde erst 1997 von Sénizergues gezeigt.

Entscheidbarkeit

Folgende Problem ist für deterministisch kontextfreie Sprachen nicht entscheidbar, d.h., man kann zeigen, dass es kein entsprechendes Verfahren gibt:

Unentscheidbarkeit bei deterministisch kontextfreien Sprachen

- Schnittproblem: Gegeben zwei deterministisch kontextfreie Sprachen L_1, L_2. Gilt $L_1 \cap L_2 = \emptyset$?

Wie bei kontextfreien Sprachen ist dieses Problem jedoch entscheidbar, wenn eine der beiden Sprachen regulär ist.

Entscheidbarkeit

Zusammenfassung Entscheidbarkeit:

Problem entscheidbar	Reguläre Spr.	Det. kfr. Spr.	Kfr. Sprachen
Wortproblem	✓	✓	✓
Leerheit	✓	✓	✓
Endlichkeit	✓	✓	✓
Schnittproblem	✓	✗	✗
Schnittp. mit reg. Spr.	✓	✓	✓
Äquivalenz	✓	✓	✗

Entscheidbarkeit

Wir betrachten als eine Anwendung von Kellerautomaten das Adventure-Problem, Level 2.

Die Schatz-Regel

Man muss mindestens zwei Schätze finden.

Die Drachen-Regel

Unmittelbar nach der Begegnung mit einem Drachen muss man in einen Fluss springen, da uns der Drache in Brand stecken wird. Dies gilt nicht mehr, sobald man ein Schwert besitzt, mit dem man den Drachen vorher töten kann.

Neue Tür-Regel

Die Schlüssel sind magisch und verschwinden sofort, nachdem eine Tür mit ihnen geöffnet wurde. Sobald man eine Tür durchschritten hat, schließt sie sich sofort wieder.

Entscheidbarkeit

Es gilt:

- Die Schatz- und die Drachen-Regel sowie die Menge aller möglichen Pfade im Adventure können durch endliche Automaten A, D, M beschrieben werden.
- Es gibt einen Kellerautomaten T (siehe nächste Folie), der alle Wörter akzeptiert, die die neue Tür-Regel erfüllen.

  Idee: Lege Schlüssel (L) auf den Keller und entferne sie wieder, sobald eine Tür (T) in der Eingabe auftaucht.

Entscheidbarkeit

(Nicht-deterministischer) Kellerautomat für die Tür-Regel:

$$T = (\{z_0\}, \{D, B, T, W, A, F, L\}, \{\#, L\}, \delta, z_0, \#),$$

mit folgendem δ:

$$\begin{array}{rcll}
(z_0, L, \#) & \to & (z_0, L\#) & \\
(z_0, a, \#) & \to & (z_0, \#), & \text{für } a \in \{D, B, W, A, F\} \\
(z_0, L, L) & \to & (z_0, LL) & \\
(z_0, T, L) & \to & (z_0, \varepsilon) & \\
(z_0, a, L) & \to & (z_0, L), & \text{für } a \in \{D, B, W, A, F\} \\
(z_0, \varepsilon, X) & \to & (z_0, \varepsilon), & \text{für } X \in \{L, \#\}
\end{array}$$

Die letzte Regel dient dazu, den Keller am Ende zu leeren, um mit leerem Keller zu akzeptieren. Sie kann auch in Sackgassen führen.

Entscheidbarkeit

Es gibt auch einen deterministischen Kellerautomaten für die Sprache aller Wörter, die die Tür-Regel erfüllen:

$$T = (\{z_0\}, \{D, B, T, W, A, F, L\}, \{\#, L\}, \delta, z_0, \#, \{z_0\}),$$

mit folgendem δ:

$$\begin{array}{rcll}
(z_0, L, \#) & \to & (z_0, L\#) & \\
(z_0, a, \#) & \to & (z_0, \#), & \text{für } a \in \{D, B, W, A, F\} \\
(z_0, L, L) & \to & (z_0, LL) & \\
(z_0, T, L) & \to & (z_0, \varepsilon) & \\
(z_0, a, L) & \to & (z_0, L), & \text{für } a \in \{D, B, W, A, F\}
\end{array}$$

Entscheidbarkeit

Verfahren zur Lösung des Adventure-Problems (Level 2):

1. Kreuzprodukt der endlichen Automaten, A, D, M bilden. Der entstehende Automat heißt ADM und akzeptiert $T(A) \cap T(D) \cap T(M)$.
2. Kreuzprodukt des Kellerautomaten T mit dem endlichen Automaten ADM bilden (siehe Abschluss von kontextfreien Sprachen unter Schnitt mit regulären Sprachen). Daraus entsteht ein Kellerautomat $TADM$.
3. Kellerautomat $TADM$ in eine kontextfreie Grammatik G umwandeln und überprüfen, ob das Startsymbol S produktiv ist. Genau in diesem Fall gibt es eine Lösung. Aufbauend auf dem Verfahren zur Überprüfung der Produktivität kann man eine solche Lösung auch explizit angeben.

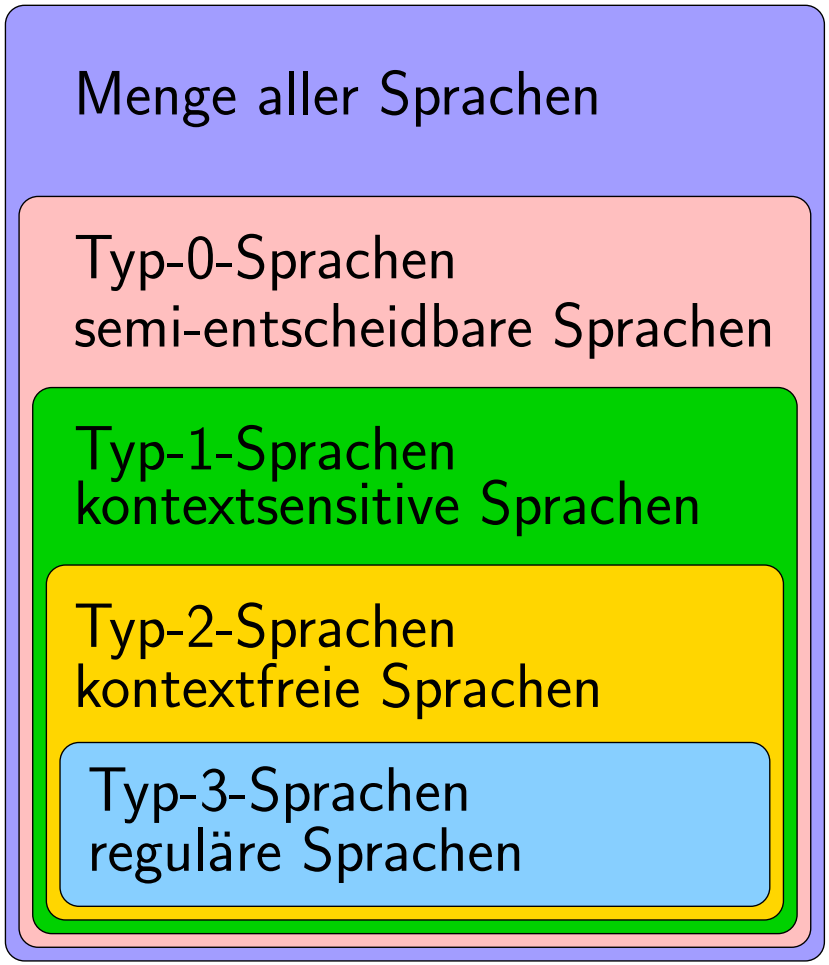
Fragestellungen zu dieser Vorlesungseinheit

Zum Ende einer jeden Vorlesungseinheit betrachten wir im Regelfall drei Fragestellungen, die mithilfe der in dieser Einheit besprochenen Inhalte beantwortet werden sollen. In der darauffolgenden Einheit können zu Beginn mögliche Antworten gesammelt werden.

Fragen zur elften Vorlesungseinheit

- Wie transformiert man einen Kellerautomaten in eine kontextfreie Grammatik?
- Wie transformiert man eine kontextfreie Grammatik in einen Kellerautomaten?
- Wie entscheidet man das Schnittproblem zwischen kontextfreien und regulären Sprachen?

Zusammenfassung



Wir haben uns bis jetzt mit den untersten beiden Stufen der Chomsky-Hierarchie beschäftigt: den regulären und kontextfreien Sprachen.

Mit den weiter oben befindlichen Stufen beschäftigt sich die Vorlesung "Berechenbarkeit und Komplexität". Inbesondere geht es darin um die Frage, was berechenbar und was nicht mehr berechenbar ist.

Weiterer Ablauf

Im Moodle werden alle Fragen zur Vorlesungseinheit zur Wahl gestellt. Diejenigen Fragen, die die meisten Stimmen erhalten, werden in der kommenden Vorlesungseinheit beantwortet.
In der darauffolgenden Woche wird die Vorlesung durch Christine Mika vertreten und als Fragestunde fungieren, bitte senden Sie zeitig Ihre Fragen per E-Mail an Christine Mika.
Das Tutorium wird bis zum Vorlesungsende wie gewohnt im Wechsel Donnerstags und Freitags stattfinden.