



FORMALE SYSTEME

11. Vorlesung: Von regulären zu kontextfreien Sprachen

Markus Krötzsch

TU Dresden, 17. November 2016

Probleme auf Automaten

Wiederholung

Einige wichtige Probleme für Automaten:

Problem	Fragestellung	Komplexität
Leerheit	$L(\mathcal{M}) \stackrel{?}{=} \emptyset$	polynomiell
Inklusion	$L(\mathcal{M}_1) \stackrel{?}{\subseteq} L(\mathcal{M}_2)$	polynomiell falls \mathcal{M}_2 DFA exponentiell falls \mathcal{M}_2 NFA
Äquivalenz	$L(\mathcal{M}_1) \stackrel{?}{=} L(\mathcal{M}_2)$	polynomiell falls \mathcal{M}_1 und \mathcal{M}_2 DFA exponentiell falls \mathcal{M}_1 oder \mathcal{M}_2 NFA

Das Wortproblem für reguläre Sprachen

Das Wortproblem für DFAs kann in polynomieller Zeit entschieden werden.

Das Wortproblem für reguläre Sprachen

Das Wortproblem für DFAs kann in polynomieller Zeit entschieden werden.

Beweis: Es genügt, den DFA für $|w|$ Schritte zu simulieren und zu prüfen, ob danach ein Endzustand erreicht ist. Die Berechnung von $\delta(q, \mathbf{a})$ ist in polynomieller Zeit möglich – die Details hängen davon ab, wie genau \mathcal{M} in der Eingabe kodiert wurde. \square

Das Wortproblem für reguläre Sprachen

Das Wortproblem für DFAs kann in polynomieller Zeit entschieden werden.

Beweis: Es genügt, den DFA für $|w|$ Schritte zu simulieren und zu prüfen, ob danach ein Endzustand erreicht ist. Die Berechnung von $\delta(q, \mathbf{a})$ ist in polynomieller Zeit möglich – die Details hängen davon ab, wie genau \mathcal{M} in der Eingabe kodiert wurde. \square

Das Wortproblem für reguläre Sprachen kann in linearer Zeit $O(|w|)$ entschieden werden.

Das Wortproblem für reguläre Sprachen

Das Wortproblem für DFAs kann in polynomieller Zeit entschieden werden.

Beweis: Es genügt, den DFA für $|w|$ Schritte zu simulieren und zu prüfen, ob danach ein Endzustand erreicht ist. Die Berechnung von $\delta(q, \mathbf{a})$ ist in polynomieller Zeit möglich – die Details hängen davon ab, wie genau \mathcal{M} in der Eingabe kodiert wurde. \square

Das Wortproblem für reguläre Sprachen kann in linearer Zeit $O(|w|)$ entschieden werden.

Beweis: Man kann den DFA als gegeben annehmen, so dass die Berechnung von $\delta(q, \mathbf{a})$ in konstanter Zeit erfolgen kann. Die Simulation benötigt daher insgesamt $|w|$ Rechenschritte. \square

Das Wortproblem für NFAs

Was tun, wenn ein NFA gegeben ist?

Das Wortproblem für NFAs

Was tun, wenn ein NFA gegeben ist?

Variante 1: NFA in DFA umwandeln (Potenzmengenkonstruktion),
Wortproblem für DFA lösen

Das Wortproblem für NFAs

Was tun, wenn ein NFA gegeben ist?

Variante 1: NFA in DFA umwandeln (Potenzmengenkonstruktion),
Wortproblem für DFA lösen

Exponentieller Algorithmus: Potenzmengen-DFA ist
exponentiell groß

Das Wortproblem für NFAs

Was tun, wenn ein NFA gegeben ist?

Variante 1: NFA in DFA umwandeln (Potenzmengenkonstruktion),
Wortproblem für DFA lösen

Exponentieller Algorithmus: Potenzmengen-DFA ist
exponentiell groß

Variante 2: NFA direkt mit Zustandsmengen simulieren
(vergleichbar „on-the-fly Version von Variante 1“)

Details: Variante 2

Eingabe: NFA $\mathcal{M} = \langle Q, \Sigma, \delta, Q_0, F \rangle$ und Wort w

Ausgabe: Ist $w \in \mathbf{L}(\mathcal{M})$?

- Initialisiere $Z := Q_0$
- Für jedes Symbol σ_i in $w = \sigma_1 \cdots \sigma_{|w|}$:
Berechne $Z := \bigcup_{q \in Z} \delta(q, \sigma_i)$
- Für alle $q \in F$:
Falls $q \in Z$: das Ergebnis ist „ja“
- Falls kein $q \in F \cap Z$ gefunden wurde: das Ergebnis ist „nein“

Alle Teilberechnungen können in polynomieller Zeit ausgeführt werden, sofern \mathcal{M} „vernünftig“ kodiert wird

Das Wortproblem für NFAs

Was tun, wenn ein NFA gegeben ist?

Variante 1: NFA in DFA umwandeln (Potenzmengenkonstruktion),
Wortproblem für DFA lösen

Exponentieller Algorithmus: Potenzmengen-DFA ist
exponentiell groß

Variante 2: NFA direkt mit Zustandsmengen simulieren
(vergleichbar „on-the-fly Version von Variante 1“)
Polynomieller Algorithmus: Zustandsmengen sind von
linearer Größe; Berechnung der Nachfolgemenge als
Vereinigung linear vieler δ -Ergebnismengen

Das Wortproblem für NFAs

Was tun, wenn ein NFA gegeben ist?

Variante 1: NFA in DFA umwandeln (Potenzmengenkonstruktion),
Wortproblem für DFA lösen

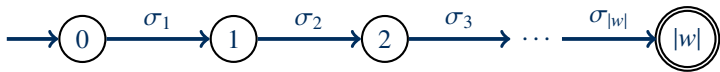
Exponentieller Algorithmus: Potenzmengen-DFA ist
exponentiell groß

Variante 2: NFA direkt mit Zustandsmengen simulieren
(vergleichbar „on-the-fly Version von Variante 1“)
Polynomieller Algorithmus: Zustandsmengen sind von
linearer Größe; Berechnung der Nachfolgemenge als
Vereinigung linear vieler δ -Ergebnismengen

Variante 3: Konstruiere einen DFA \mathcal{M}_w mit $\mathbf{L}(\mathcal{M}_w) = \{w\}$ und prüfe
ob $\mathbf{L}(\mathcal{M}) \cap \mathbf{L}(\mathcal{M}_w) \neq \emptyset$

Details: Variante 3

Der Automat \mathcal{M}_w für $w = \sigma_1 \cdots \sigma_{|w|}$ ist leicht gefunden:



Eingabe: NFA $\mathcal{M} = \langle Q, \Sigma, \delta, Q_0, F \rangle$ und Wort w

Ausgabe: Ist $w \in \mathbf{L}(\mathcal{M})$?

- (1) Konstruiere \mathcal{M}_w
- (2) Berechne Produktautomat $\mathcal{M} \otimes \mathcal{M}_w$
- (3) Entscheide ob $(\mathcal{M} \otimes \mathcal{M}_w) \neq \emptyset$

Das Wortproblem für NFAs

Was tun, wenn ein NFA gegeben ist?

Variante 1: NFA in DFA umwandeln (Potenzmengenkonstruktion),
Wortproblem für DFA lösen

Exponentieller Algorithmus: Potenzmengen-DFA ist
exponentiell groß

Variante 2: NFA direkt mit Zustandsmengen simulieren
(vergleichbar „on-the-fly Version von Variante 1“)

Polynomieller Algorithmus: Zustandsmengen sind von
linearer Größe; Berechnung der Nachfolgemenge als
Vereinigung linear vieler δ -Ergebnismengen

Variante 3: Konstruiere einen DFA \mathcal{M}_w mit $\mathbf{L}(\mathcal{M}_w) = \{w\}$ und prüfe
ob $\mathbf{L}(\mathcal{M}) \cap \mathbf{L}(\mathcal{M}_w) \neq \emptyset$

Polynomieller Algorithmus: \mathcal{M}_w ist linear in $|w|$;
Schnittmengen-DFA ist quadratisch groß; Leerheitstest
ist polynomiell in dieser Größe

Wortproblem für NFAs – Komplexität

Variante 1: NFA in DFA umwandeln (Potenzmengenkonstruktion), Wortproblem für DFA lösen

Variante 2: NFA direkt mit Zustandsmengen simulieren (vergleichbar „on-the-fly Version von Variante 1“)

Variante 3: Konstruiere einen DFA \mathcal{M}_w mit $\mathbf{L}(\mathcal{M}_w) = \{w\}$ und prüfe ob $\mathbf{L}(\mathcal{M}) \cap \mathbf{L}(\mathcal{M}_w) \neq \emptyset$

Mit Variante 2 und 3 erhalten wir:

Satz: Das Wortproblem für NFAs kann in polynomieller Zeit entschieden werden.

Weitere Probleme für Automaten

Das **Endlichkeitsproblem** für FAs über Alphabet Σ besteht darin, die folgende Funktion zu berechnen:

Eingabe: ein FA \mathcal{M}

Ausgabe: „ja“ wenn $L(\mathcal{M})$ endlich ist; andernfalls „nein“

Weitere Probleme für Automaten

Das **Endlichkeitsproblem** für FAs über Alphabet Σ besteht darin, die folgende Funktion zu berechnen:

Eingabe: ein FA \mathcal{M}

Ausgabe: „ja“ wenn $L(\mathcal{M})$ endlich ist; andernfalls „nein“

Idee wie Pumping-Lemma: unendliche Sprachen erfordern Zyklus
 \leadsto suche nach Zyklen, die auf einem Pfad von einem Start- zu einem Endzustand liegen (polynomiell)

Weitere Probleme für Automaten

Das **Endlichkeitsproblem** für FAs über Alphabet Σ besteht darin, die folgende Funktion zu berechnen:

Eingabe: ein FA \mathcal{M}

Ausgabe: „ja“ wenn $L(\mathcal{M})$ endlich ist; andernfalls „nein“

Idee wie Pumping-Lemma: unendliche Sprachen erfordern Zyklus
 \leadsto suche nach Zyklen, die auf einem Pfad von einem Start- zu einem Endzustand liegen (polynomiell)

Das **Universalitätsproblem** für FAs über Alphabet Σ besteht darin, die folgende Funktion zu berechnen:

Eingabe: ein FA \mathcal{M}

Ausgabe: „ja“ wenn $L(\mathcal{M}) = \Sigma^*$; andernfalls „nein“

Weitere Probleme für Automaten

Das **Endlichkeitsproblem** für FAs über Alphabet Σ besteht darin, die folgende Funktion zu berechnen:

Eingabe: ein FA \mathcal{M}

Ausgabe: „ja“ wenn $L(\mathcal{M})$ endlich ist; andernfalls „nein“

Idee wie Pumping-Lemma: unendliche Sprachen erfordern Zyklus
 \leadsto suche nach Zyklen, die auf einem Pfad von einem Start- zu einem Endzustand liegen (polynomiell)

Das **Universalitätsproblem** für FAs über Alphabet Σ besteht darin, die folgende Funktion zu berechnen:

Eingabe: ein FA \mathcal{M}

Ausgabe: „ja“ wenn $L(\mathcal{M}) = \Sigma^*$; andernfalls „nein“

Komplement des Leerheitsproblems: $L(\mathcal{M}) = \Sigma^*$ wenn $L(\overline{\mathcal{M}}) = \emptyset$

\leadsto Komplexität abhängig von FA-Komplementierung

Zusammenfassung

Einige wichtige Probleme für Automaten:

Problem	Fragestellung	Komplexität
Leerheit	$\mathbf{L}(\mathcal{M}) \stackrel{?}{=} \emptyset$	polynomiell
Inklusion	$\mathbf{L}(\mathcal{M}_1) \stackrel{?}{\subseteq} \mathbf{L}(\mathcal{M}_2)$	polynomiell falls \mathcal{M}_2 DFA exponentiell falls \mathcal{M}_2 NFA
Äquivalenz	$\mathbf{L}(\mathcal{M}_1) \stackrel{?}{=} \mathbf{L}(\mathcal{M}_2)$	polynomiell falls \mathcal{M}_1 und \mathcal{M}_2 DFA exponentiell falls \mathcal{M}_1 oder \mathcal{M}_2 NFA
Wortproblem	$w \stackrel{?}{\in} \mathbf{L}(\mathcal{M})$	polynomiell
Universalität	$\mathbf{L}(\mathcal{M}) \stackrel{?}{=} \Sigma^*$	polynomiell falls \mathcal{M} DFA exponentiell falls \mathcal{M} NFA
Endlichkeit	$\mathbf{L}(\mathcal{M})$ endlich?	polynomiell

Viereinhalb Wochen reguläre Sprachen

auf sieben Folien

Die Chomsky-Hierarchie

Die **Chomsky-Hierarchie** unterteilt Grammatiken in vier Stufen:

- **Typ 0:** beliebige Grammatiken
- **Typ 1: kontextsensitive Grammatiken:**
 - (a) Alle Regeln $w \rightarrow v$ erfüllen die Bedingung $|w| \leq |v|$.
 - (b) Es gibt eine Regel $S \rightarrow \epsilon$ und alle anderen Regeln $w \rightarrow v$ enthalten kein S in v und erfüllen $|w| \leq |v|$.
- **Typ 2: kontextfreie Grammatiken:**

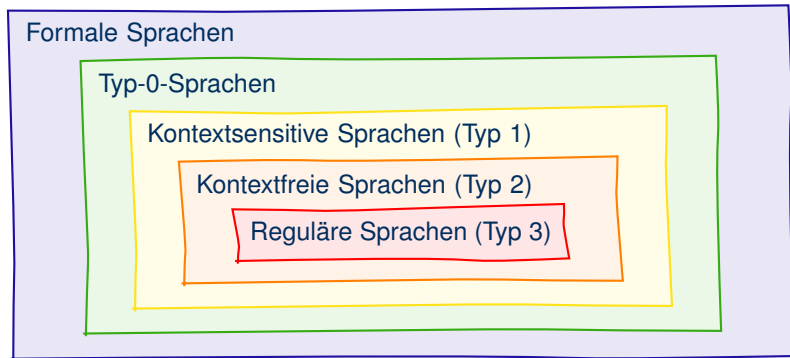
Alle Regeln haben die Form $A \rightarrow v$ für eine Variable A .
- **Typ 3: reguläre Grammatiken:**

Alle Regeln haben eine der Formen

$$A \rightarrow cB \quad A \rightarrow c \quad A \rightarrow \epsilon$$

wobei A und B Variablen sind und c ein Terminalsymbol ist.

Chomsky's Hierarchie ist eine Hierarchie



(Dafür mussten wir Typ-1 erweitern und ϵ -Regeln bei Typ-2 eliminieren.)

Automaten

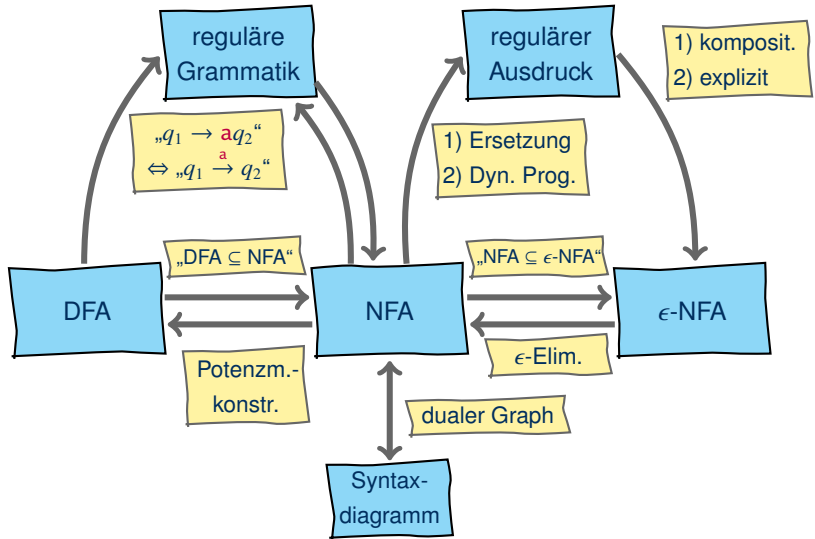
Wir kennen mehrere Varianten endlicher Automaten:

- **Deterministischer endlicher Automat (DFA)**
 - mit totaler Übergangsfunktion
- **Nichtdeterministischer endlicher Automat (NFA)**
 - mit ϵ -Übergängen
 - mit Wortübergängen
 - mit Übergängen für reguläre Ausdrücke (nur für Umwandlung reg. Ausdruck \rightarrow ϵ -NFA)

Die Sprache eines Automaten haben wir auf zwei Arten definiert

- Mithilfe einer verallgemeinerten Übergangsfunktion, die ganze Wörter einliest
- Durch akzeptierende Läufe, die einem Wort zugeordnet werden können

Darstellungen von Typ-3-Sprachen



Umformungsalgorithmen (1)

	Eingabe	Ausgabe	V.
	kontextfreie Grammatik	ϵ -freie kontextfreie Grammatik	2
	DFA \mathcal{M}	totaler DFA $\mathcal{M}_{\text{total}}$	3
	DFA \mathcal{M}	reguläre Grammatik $G_{\mathcal{M}}$	3
	Syntaxdiagramm	NFA	4
	NFA \mathcal{M}	Potenzmengen-DFA \mathcal{M}_{DFA}	4
	reguläre Grammatik G	NFA \mathcal{M}_G	5
	NFA mit Wortübergängen	ϵ -NFA	5
	ϵ -NFA \mathcal{M}	NFA $\text{elim}_{\epsilon}(\mathcal{M})$	5

Umformungsalgorithmen (2)

Eingabe	Ausgabe	V.
NFAs $\mathcal{M}_1, \mathcal{M}_2$	Vereinigungs-NFA $\mathcal{M}_1 \oplus \mathcal{M}_2$	5
NFAs/DFAs $\mathcal{M}_1, \mathcal{M}_2$	Produkt-NFA/DFA $\mathcal{M}_1 \otimes \mathcal{M}_2$	5
totaler DFA \mathcal{M}	Komplement-DFA $\overline{\mathcal{M}}$	5
NFAs $\mathcal{M}_1, \mathcal{M}_2$	ϵ -NFA $\mathcal{M}_1 \odot \mathcal{M}_2$ für Konkatination	5
NFA \mathcal{M}	ϵ -NFA \mathcal{M}^* für Kleene-Abschluss	5
regulärer Ausdruck	ϵ -NFA (Komposition)	6
regulärer Ausdruck	ϵ -NFA (explizit)	6
NFA	regulärer Ausdruck (Gleichungssystem)	7
NFA	regulärer Ausdruck (dyn. Programmierung)	7
totaler DFA \mathcal{M}	Quotienten-DFA \mathcal{M}/\sim	8
totaler DFA \mathcal{M}	reduzierter DFA \mathcal{M}_r	8

Reguläre Sprachen

Die Menge der regulären Sprachen ist ...

- die Menge genau all jener Sprachen ...
 - die von einer Typ-3-Grammatik beschrieben werden
 - die von einem DFA erkannt werden
 - die von einem NFA erkannt werden
 - die durch einen regulären Ausdruck beschrieben werden
 - die endlich viele Myhill-Nerode-Kongruenzklassen haben
- die kleinste Menge von Sprachen ...
 - welche alle endlichen Sprachen enthält und unter \cap , \cup , $\bar{}$, \circ und $*$ abgeschlossen ist
 - welche die Sprachen \emptyset , $\{\epsilon\}$ und $\{a\}$ ($a \in \Sigma$) enthält und unter \cup , \circ und $*$ abgeschlossen ist

Alle endlichen Sprachen sind regulär (aber nicht umgekehrt)

Alle regulären Sprachen erlauben Pumping (aber nicht umgekehrt)

Kontextfreie Sprachen

Kontextfreie Sprachen

Wir hatten kontextfreie Sprachen wie folgt definiert:

Eine **kontextfreie Grammatik** (oder **Typ-2-Grammatik** oder **CFG**) enthält nur Regeln der Form $A \rightarrow v$, wobei A eine Variable ist.

Eine Sprache ist **kontextfrei** (oder **Typ 2**), wenn sie durch eine kontextfreie Grammatik dargestellt werden kann.

Kontextfreie Sprachen

Wir hatten kontextfreie Sprachen wie folgt definiert:

Eine **kontextfreie Grammatik** (oder **Typ-2-Grammatik** oder **CFG**) enthält nur Regeln der Form $A \rightarrow v$, wobei A eine Variable ist.

Eine Sprache ist **kontextfrei** (oder **Typ 2**), wenn sie durch eine kontextfreie Grammatik dargestellt werden kann.

Das genügt, um nichtreguläre Sprachen darzustellen:

Beispiel: Die Sprache $\{a^n b^n \mid n \geq 0\}$ ist kontextfrei, da sie durch die folgende CFG dargestellt werden kann:

$$S \rightarrow \epsilon \mid aSb$$

(Übung: Beweise, dass die Grammatik wirklich diese Sprache darstellt.)

Beispiel

CFGs eignen sich zur Darstellung vollständig geklammerter Ausdrücke.

Beispiel: Vollständig geklammerte reguläre Ausdrücke über Alphabet $\Sigma = \{\sigma_1, \dots, \sigma_n\}$ sind als CFG über dem Alphabet $\Sigma \cup \{\emptyset, \epsilon, (,), |, *\}$ darstellbar:

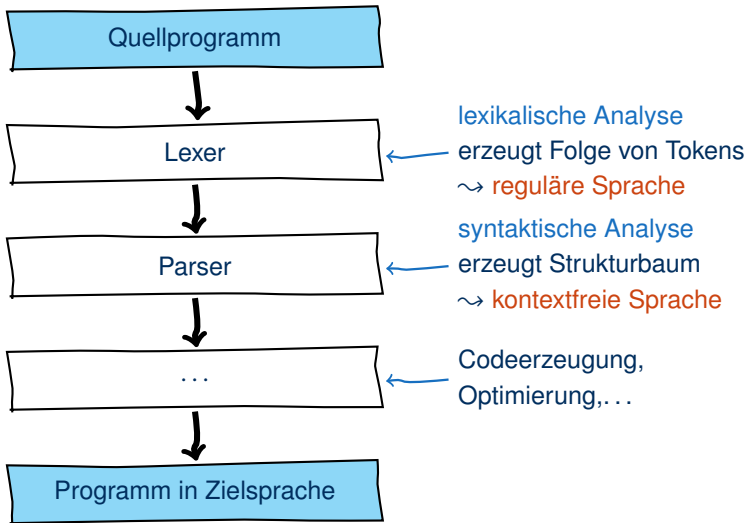
$$S \rightarrow \emptyset \mid \epsilon \mid A \mid (SS) \mid (S|S) \mid (S)^*$$

$$A \rightarrow \sigma_1 \mid \dots \mid \sigma_n$$

Allgemein ist die Beschreibung korrekt geklammerter Ausdrücke für viele Sprachen sehr wichtig, nicht zuletzt für Programmiersprachen

Ein praktisches Beispiel . . .

Beispiel Compiler



Wiederholung: Ableitung

Sei $\langle V, \Sigma, P, S \rangle$ eine Grammatik. Die **1-Schritt-Ableitungsrelation** ist eine binäre Relation \Rightarrow zwischen Wörtern aus $(V \cup \Sigma)^*$, so dass $u \Rightarrow v$ genau dann wenn:

$$u = w_1 x w_2 \text{ und } v = w_1 y w_2 \text{ und es gibt eine Regel } x \rightarrow y \in P$$

wobei $w_1, w_2, x, y \in (V \cup \Sigma)^*$ beliebige Wörter sind.

Die **Ableitungsrelation** \Rightarrow^* ist der reflexive, transitive Abschluss von \Rightarrow , das heißt $u \Rightarrow^* v$ genau dann wenn:

$$u = w_1 \Rightarrow w_2 \Rightarrow \dots \Rightarrow w_{n-1} \Rightarrow w_n = v$$

wobei $n \geq 1$ und $w_1, \dots, w_n \in (V \cup \Sigma)^*$ beliebige Wörter sind. Insbesondere gilt $u \Rightarrow^* u$ für alle $u \in (V \cup \Sigma)^*$ (Fall $n = 1$).

Anmerkung: Der Begriff „Herleitungsrelation“ ist auch gebräuchlich. Wir verwenden „Ableitung“ und „Herleitung“ synonym.

Anmerkung 2: Manche Autoren schreiben \vdash statt \Rightarrow .

Beispiel

Die Grammatik

$$S \rightarrow A \mid M \mid V$$

$$A \rightarrow (S+S)$$

$$M \rightarrow (S*S)$$

$$V \rightarrow x \mid y \mid z$$

erzeugt zum Beispiel das Wort $(x * (y + z))$ über die Ableitung:

$$\begin{aligned} S &\Rightarrow M \Rightarrow (S*S) \Rightarrow (V*S) \Rightarrow (x*S) \Rightarrow (x*A) \Rightarrow (x*(S+S)) \\ &\Rightarrow (x*(V+S)) \Rightarrow (x*(y+S)) \Rightarrow (x*(y+V)) \Rightarrow (x*(y+z)) \end{aligned}$$

Ableitungen als Bäume

Grammatik:

S

$S \rightarrow A \mid M \mid V$ $A \rightarrow (S+S)$

$M \rightarrow (S*S)$ $V \rightarrow x \mid y \mid z$

Ableitung:

S

Ableitungen als Bäume

Grammatik:

$S \rightarrow A \mid M \mid V$ $A \rightarrow (S+S)$

$M \rightarrow (S*S)$ $V \rightarrow x \mid y \mid z$

S
|
M

Ableitung:

$S \Rightarrow M$

Ableitungen als Bäume

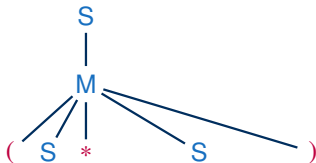
Grammatik:

$S \rightarrow A \mid M \mid V$ $A \rightarrow (S+S)$

$M \rightarrow (S*S)$ $V \rightarrow x \mid y \mid z$

Ableitung:

$S \Rightarrow M \Rightarrow (S*S)$



Ableitungen als Bäume

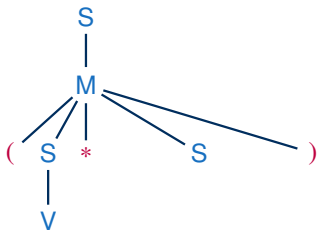
Grammatik:

$S \rightarrow A \mid M \mid V$ $A \rightarrow (S+S)$

$M \rightarrow (S*S)$ $V \rightarrow x \mid y \mid z$

Ableitung:

$S \Rightarrow M \Rightarrow (S*S) \Rightarrow (V*S)$



Ableitungen als Bäume

Grammatik:

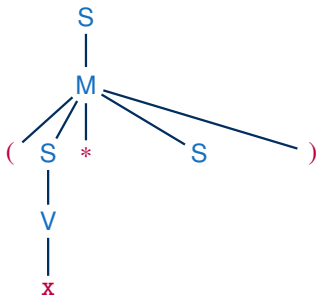
$$S \rightarrow A \mid M \mid V \quad A \rightarrow (S+S)$$

$$M \rightarrow (S*S) \quad V \rightarrow x \mid y \mid z$$

Ableitung:

$$S \Rightarrow M \Rightarrow (S*S) \Rightarrow (V*S)$$

$$\Rightarrow (x*S)$$



Ableitungen als Bäume

Grammatik:

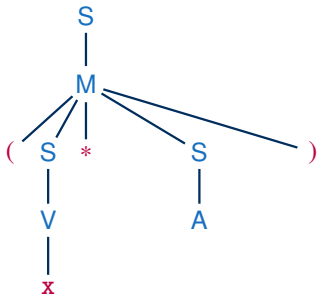
$$S \rightarrow A \mid M \mid V \quad A \rightarrow (S+S)$$

$$M \rightarrow (S*S) \quad V \rightarrow x \mid y \mid z$$

Ableitung:

$$S \Rightarrow M \Rightarrow (S*S) \Rightarrow (V*S)$$

$$\Rightarrow (x*S) \Rightarrow (x*A)$$



Ableitungen als Bäume

Grammatik:

$S \rightarrow A \mid M \mid V$ $A \rightarrow (S+S)$

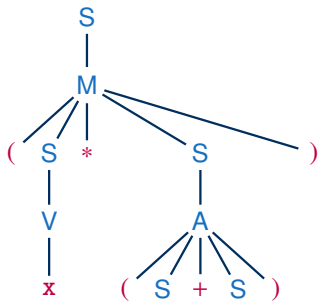
$M \rightarrow (S*S)$ $V \rightarrow x \mid y \mid z$

Ableitung:

$S \Rightarrow M \Rightarrow (S*S) \Rightarrow (V*S)$

$\Rightarrow (x*S) \Rightarrow (x*A)$

$\Rightarrow (x*(S+S))$



Ableitungen als Bäume

Grammatik:

$S \rightarrow A \mid M \mid V$ $A \rightarrow (S+S)$

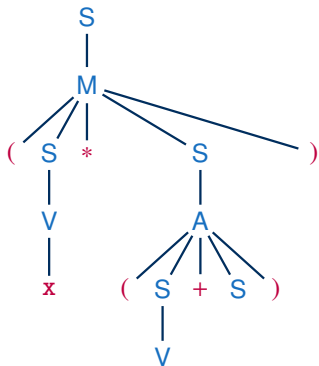
$M \rightarrow (S*S)$ $V \rightarrow x \mid y \mid z$

Ableitung:

$S \Rightarrow M \Rightarrow (S*S) \Rightarrow (V*S)$

$\Rightarrow (x*S) \Rightarrow (x*A)$

$\Rightarrow (x*(S+S)) \Rightarrow (x*(V+S))$



Ableitungen als Bäume

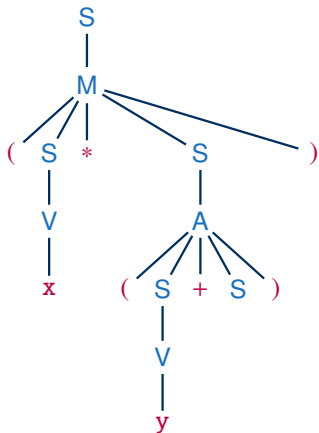
Grammatik:

$S \rightarrow A \mid M \mid V$ $A \rightarrow (S+S)$

$M \rightarrow (S*S)$ $V \rightarrow x \mid y \mid z$

Ableitung:

$S \Rightarrow M \Rightarrow (S*S) \Rightarrow (V*S)$
 $\Rightarrow (x*S) \Rightarrow (x*A)$
 $\Rightarrow (x*(S+S)) \Rightarrow (x*(V+S))$
 $\Rightarrow (x*(y+S))$



Ableitungen als Bäume

Grammatik:

$S \rightarrow A \mid M \mid V$ $A \rightarrow (S+S)$

$M \rightarrow (S*S)$ $V \rightarrow x \mid y \mid z$

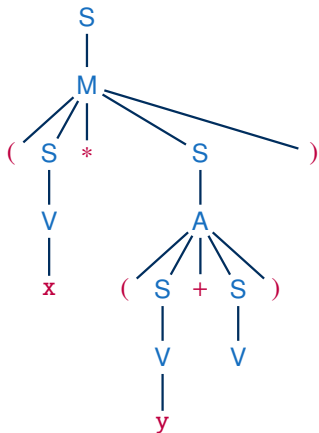
Ableitung:

$S \Rightarrow M \Rightarrow (S*S) \Rightarrow (V*S)$

$\Rightarrow (x*S) \Rightarrow (x*A)$

$\Rightarrow (x*(S+S)) \Rightarrow (x*(V+S))$

$\Rightarrow (x*(y+S)) \Rightarrow (x*(y+V))$



Ableitungen als Bäume

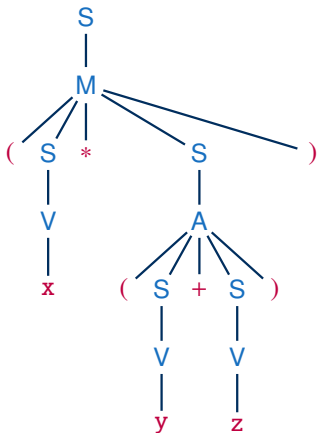
Grammatik:

$S \rightarrow A \mid M \mid V$ $A \rightarrow (S+S)$

$M \rightarrow (S*S)$ $V \rightarrow x \mid y \mid z$

Ableitung:

$S \Rightarrow M \Rightarrow (S*S) \Rightarrow (V*S)$
 $\Rightarrow (x*S) \Rightarrow (x*A)$
 $\Rightarrow (x*(S+S)) \Rightarrow (x*(V+S))$
 $\Rightarrow (x*(y+S)) \Rightarrow (x*(y+V))$
 $\Rightarrow (x*(y+z))$



Von Ableitung zu Ableitungsbaum

Sei $G = \langle V, \Sigma, P, S \rangle$ eine Grammatik und sei $S = w_0 \Rightarrow w_1 \Rightarrow \dots \Rightarrow w_n$ eine Ableitung (mit $w_i \in (V \cup \Sigma)^*$ für alle $i \in \{1, \dots, n\}$).

Wir erhalten den entsprechenden **Ableitungsbaum** wie folgt:

- Der Ableitungsbaum wird initialisiert mit einem einzigen Wurzelknoten S
- Der Baum wird schrittweise konstruiert. Nach i Schritten ergeben die Blätter des Baumes – gelesen von links nach rechts – immer genau w_i .
- Wenn in einem Ableitungsschritt $w_i \Rightarrow w_{i+1}$ die Regel $V \rightarrow u$ angewendet wurde, dann erhält der Knoten für V genau $|u|$ Kindknoten, die – von links nach rechts – mit den Symbolen aus u beschriftet werden.

Ableitungsbäume sind auch als **Syntaxbäume** oder **Parsebäume** bekannt

Anwendung Ableitungsbaum

Der Ableitungsbaum ist von großer praktischer Bedeutung, da er die „innere Struktur“ eines Wortes einer kontextfreien Sprache repräsentiert

In der Praxis geht es meist nicht darum, zu prüfen, ob ein Wort in einer Sprache liegt, sondern darum, seine syntaktische Struktur zu ermitteln

Beispiele:

- Parsebäume in der **Verarbeitung natürlicher Sprache** können Aufschluss über die Bedeutung eines Satzes geben
- Syntaxbäume in **Programmiersprachen** sind die Grundlage zur inhaltlichen Interpretation des Codes
- Ableitungsbäume in **Mark-Up-Sprachen** wie HTML oder XML sind entscheidend für die Adressierung von Elementen („DOM-Tree“)

Zusammenfassung und Ausblick

Algorithmische Probleme zu Automaten sind **Leerheit, Inklusion, Äquivalenz, Universalität, Endlichkeit** und das **Wortproblem**

für einige von ihnen sind bei Verwendung von NFAs nur exponentielle Algorithmen bekannt

Wir kennen **viele Charakterisierungen für reguläre Sprachen**, die man mit zahlreichen Umformungen in Beziehung setzen kann

Wörter in **kontextfreien Sprachen** haben eine interessante innere Struktur, die wir durch **Ableitungsbäume** darstellen können

Offene Fragen:

- Wie kann das Wortproblem bei kontextfreien Grammatiken gelöst werden?
- Haben kontextfreie Sprachen ein Berechnungsmodell?
- Wie sehen nicht-kontextfreie Sprachen aus und wie erkennt man sie?