Grundbegriffe der Informatik

Kapitel 15: Reguläre Ausdrücke und rechtslineare Grammatiken

Mattias Ulbrich
(basierend auf Folien von Thomas Worsch)

KIT · Institut für Theoretische Informatik

Wintersemester 2023/2024

Was können endliche Akzeptoren?

- manche Sprachen mit EA erkennbar: z. B. $\{\mathtt{a}\}^+\{\mathtt{b}\} \cup \{\mathtt{b}\}^+\{\mathtt{a}\}$
- manche Sprachen *nicht* mit EA erkennbar: z. B. $\{\mathtt{a}^k\mathtt{b}^k \mid k \in \mathbb{N}_0\}$

Was können endliche Akzeptoren?

- manche Sprachen mit EA erkennbar: z. B. $\{\mathtt{a}\}^+\{\mathtt{b}\} \cup \{\mathtt{b}\}^+\{\mathtt{a}\}$
- manche Sprachen *nicht* mit EA erkennbar: z. B. $\{\mathtt{a}^k\mathtt{b}^k \mid k \in \mathbb{N}_0\}$
- **Charakterisierung** der erkennbaren Sprachen
  - d. h. Beschreibung ohne Benutzung endlicher Akzeptoren
  - Drei Formalismen die dieselbe Klasse an formalen Sprachen beschreiben:

    1. Endliche Akzeptoren
    2. Reguläre Ausdrücke
    3. Rechtslineare Grammatiken
  - Jede formale Sprache kann in einem Formalismus ausgedrückt werden, gdw. sie in den anderen ausgedrückt werden kann.

Überblick

Karlsruher Institut für Technologie

Reguläre Ausdrücke
- Definition
- Semantik regulärer Ausdrücke
- Beispiel: `grep`
- Zusammenhang mit Automaten und Grammatiken

Rechtslineare Grammatiken (Typ 3)

Kantorowitsch-Bäume und strukturelle Induktion

Wo sind wir?

Karlsruher Institut für Technologie

Reguläre Ausdrücke

Rechtslineare Grammatiken (Typ 3)

Kantorowitsch-Bäume und strukturelle Induktion

Wo sind wir?

Der Begriff regulärer Ausdruck hat heute verschiedene Bedeutungen

- **Wichtige Beschreibungssprache für formaler Sprachen**

- **Ursprung:**
  Stephen Kleene
  Representation of Events in Nerve Nets and Finite Automata
  in: Shannon, McCarthy, Ashby (eds.): *Automata Studies*, 1956
  (Research Memorandum “Project RAND” from 1951)

- **heute:** theoretische und praktische Bedeutung
  - *in dieser Vorlesung*: die «klassische», grundlegende Definition
  - Verallgemeinerung: *regular expressions*
    - *im täglichen Leben* sehr nützlich (emacs, grep, sed, …, Java, Python, …)

Definition regulärer Ausdrücke (1)

- Alphabet $Z = \{|, (,), *, \emptyset\}$ von «Hilfssymbolen»
- Alphabet A enthalte kein Zeichen aus Z
- *regulärer Ausdruck* über A ist eine Zeichenfolge über dem Alphabet $A \cup Z$, die gewissen Vorschriften genügt.
- Menge der regulären Ausdrücke (RA) ist wie folgt festgelegt:
  - $\emptyset$ ist ein RA
  - für jedes $x \in A$ ist x RA
  - wenn R_1 und R_2 RA sind, dann auch $(R_1|R_2)$ und (R_1R_2)
  - wenn R ein RA ist, dann auch $(R*)$
  - Nichts anderes sind reguläre Ausdrücke.

(vgl. Definition von Formeln in Aussagen- und Prädikatenlogik)

Definition regulärer Ausdrücke (1)

- Alphabet $Z = \{|, (,), *, \emptyset\}$ von «Hilfssymbolen»
- Alphabet A enthalte kein Zeichen aus Z
- *regulärer Ausdruck* über A ist eine Zeichenfolge über dem Alphabet $A \cup Z$, die gewissen Vorschriften genügt.
- Menge der regulären Ausdrücke (RA) ist wie folgt festgelegt:
  - $\emptyset$ ist ein RA
  - für jedes $x \in A$ ist x RA
  - wenn R_1 und R_2 RA sind, dann auch $(R_1|R_2)$ und (R_1R_2)
  - wenn R ein RA ist, dann auch $(R*)$
  - Nichts anderes sind reguläre Ausdrücke.

(vgl. Definition von Formeln in Aussagen- und Prädikatenlogik)

RA kleinste Menge mit:

- $\emptyset \in RA$
- $R_1, R_2 \in RA \implies (R_1|R_2), (R_1R_2) \in RA$
- $R \in RA \implies (R*) \in RA$

Beispiele

- sei $A = \{\texttt{a}, \texttt{b}\}$

- Beispiele regulärer Ausdrücke:
  `(Øb)` `((ab)(aa))` `(a|(b|(a|a)))`
  `(((((ab)b)*)*)|(Ø*))`

- **keine** Beispiele für reguläre Ausdrücke:
  `(|b)` vor `|` fehlt ein regulärer Ausdruck
  `()ab` zwischen `(` und `)` fehlt ein regulärer Ausdruck
  `((ab)` Klammern müssen «gepaart» auftreten
  `*(ab)` vor `*` fehlt ein regulärer Ausdruck

Klammereinsparungsregeln

- «Stern- vor Punktrechnung»
- «Punkt- vor Strichrechnung»
- Beispiel:
  - $R_1 \mid R_2 R_3 *$ Kurzform für
  - $R_1 \mid R_2 R_3 *$
- Bei mehreren gleichen binären Operatoren ohne Klammern gilt das als links geklammert
- Beispiel
  - $R_1 \mid R_2 \mid R_3$ Kurzform für
  - $((R_1 \mid R_2) \mid R_3)$

Klammereinsparungsregeln

- «Stern- vor Punktrechnung»
- «Punkt- vor Strichrechnung»
- Beispiel:
  - $R_1 \mid R_2 R_3 *$ Kurzform für
  - $R_1 \mid R_2 (R_3 *)$
- Bei mehreren gleichen binären Operatoren ohne Klammern gilt das als links geklammert
- Beispiel
  - $R_1 \mid R_2 \mid R_3$ Kurzform für
  - $((R_1 \mid R_2) \mid R_3)$

Klammereinsparungsregeln

- «Stern- vor Punktrechnung»
- «Punkt- vor Strichrechnung»
- Beispiel:
  - $R_1 \mid R_2 R_3 *$ Kurzform für
  - $R_1 \mid (R_2 (R_3 *))$
- Bei mehreren gleichen binären Operatoren ohne Klammern gilt das als links geklammert
- Beispiel
  - $R_1 \mid R_2 \mid R_3$ Kurzform für
  - $((R_1 \mid R_2) \mid R_3)$

Klammereinsparungsregeln

- «Stern- vor Punktrechnung»
- «Punkt- vor Strichrechnung»
- Beispiel:
  - $R_1 \mid R_2 R_3 *$ Kurzform für
  - $(R_1 \mid (R_2 (R_3 *)))$
- Bei mehreren gleichen binären Operatoren ohne Klammern gilt das als links geklammert
- Beispiel
  - $R_1 \mid R_2 \mid R_3$ Kurzform für
  - $((R_1 \mid R_2) \mid R_3)$

Wo sind wir?

Durch reg. Ausdruck R beschriebene formale Sprache $L(R)$

Sei $R \in RA$ über einem Alphabet A.
Dann ist die durch R beschriebene Sprache $L(R)$ induktiv definiert durch:

- $L(\emptyset) = \{\}$
- $L(x) = \{x\}$ für jedes $x \in A$
- $L(R_1 \mid R_2) = L(R_1) \cup L(R_2)$
- $L(R_1 R_2) = L(R_1) \cdot L(R_2)$
- $L(R*) = L(R)^*$

- Die Sprache RA der regulären Ausdrücke ist eine Sprache zur Beschreibung von Sprachen.
- Definition folgt der für reguläre Ausdrücke

Wiedererkennungseffekt: Mengenausdrücke für Sprachen

Beispiele für $L(R)$

- $R = \mathtt{a} \mid \mathtt{b}$: dann ist
  $L(R) = L(\mathtt{a} \mid \mathtt{b}) = L(\mathtt{a}) \cup L(\mathtt{b}) = \{\mathtt{a}\} \cup \{\mathtt{b}\} = \{\mathtt{a}, \mathtt{b}\}$

Beispiele für $L(R)$

- $R = \texttt{a|b}$: dann ist
  $L(R) = L(\texttt{a|b}) = L(\texttt{a}) \cup L(\texttt{b}) = \{\texttt{a}\} \cup \{\texttt{b}\} = \{\texttt{a}, \texttt{b}\}$

- $R = \texttt{(a|b)*}$: dann ist
  $L(R) = L(\texttt{(a|b)*}) = L(\texttt{a|b})^* = \{\texttt{a}, \texttt{b}\}^*$

Beispiele für $L(R)$

- $R = \texttt{a|b}$: dann ist
  $L(R) = L(\texttt{a|b}) = L(\texttt{a}) \cup L(\texttt{b}) = \{\texttt{a}\} \cup \{\texttt{b}\} = \{\texttt{a}, \texttt{b}\}$

- $R = \texttt{(a|b)*}$: dann ist
  $L(R) = L(\texttt{(a|b)*}) = L(\texttt{a|b})^* = \{\texttt{a}, \texttt{b}\}^*$

- $R = \texttt{(a*b*)*}$: dann ist
  $L(R) = L(\texttt{(a*b*)*}) = L(\texttt{a*b*})^*$
  $= (L(\texttt{a*})L(\texttt{b*}))^* = (L(\texttt{a})^* L(\texttt{b})^*)^* = (\{\texttt{a}\}^* \{\texttt{b}\}^*)^*$

Beispiele für $L(R)$

- $R = \texttt{a|b}$: dann ist
  $L(R) = L(\texttt{a|b}) = L(\texttt{a}) \cup L(\texttt{b}) = \{\texttt{a}\} \cup \{\texttt{b}\} = \{\texttt{a}, \texttt{b}\}$

- $R = \texttt{(a|b)*}$: dann ist
  $L(R) = L(\texttt{(a|b)*}) = L(\texttt{a|b})^* = \{\texttt{a}, \texttt{b}\}^*$

- $R = \texttt{(a*b*)*}$: dann ist
  $L(R) = L(\texttt{(a*b*)*}) = L(\texttt{a*b*})^*$
  $= (L(\texttt{a*})L(\texttt{b*}))^* = (L(\texttt{a})^*L(\texttt{b})^*)^* = (\{\texttt{a}\}^*\{\texttt{b}\}^*)^*$

- Nachdenken: $(\{\texttt{a}\}^*\{\texttt{b}\}^*)^* =$

Beispiele für $L(R)$

- $R = \texttt{a|b}$: dann ist
  $L(R) = L(\texttt{a|b}) = L(\texttt{a}) \cup L(\texttt{b}) = \{\texttt{a}\} \cup \{\texttt{b}\} = \{\texttt{a}, \texttt{b}\}$

- $R = \texttt{(a|b)*}$: dann ist
  $L(R) = L(\texttt{(a|b)*}) = L(\texttt{a|b})^* = \{\texttt{a}, \texttt{b}\}^*$

- $R = \texttt{(a*b*)*}$: dann ist
  $L(R) = L(\texttt{(a*b*)*}) = L(\texttt{a*b*})^*$
  $= (L(\texttt{a*})L(\texttt{b*}))^* = (L(\texttt{a})^*L(\texttt{b})^*)^* = (\{\texttt{a}\}^*\{\texttt{b}\}^*)^*$

- Nachdenken: $(\{\texttt{a}\}^*\{\texttt{b}\}^*)^* = \{\texttt{a}, \texttt{b}\}^*$

Wie ist das denn eigentlich?

- Kann man «allgemein» von regulären Ausdrücken R_1, R_2 feststellen, ob $L(R_1) = L(R_2)$ ist?
- Geht das algorithmisch?

- Welche formalen Sprachen sind denn durch reguläre Ausdrücke beschreibbar?

Äquivalenz regulärer Ausdrücke

- Es gibt Algorithmen, um für reguläre Ausdrücke R_1, R_2 festzustellen, ob $L(R_1) = L(R_2)$ ist.
  - sogar konzeptionell ziemlich einfache
- **Aber:** Dieses Problem ist PSPACE-vollständig.
  - Definition: in einer anderen Vorlesung
  - alle bisher bekannten (!) Algorithmen sind sehr sehr sehr sehr langsam
- Man weiß nicht, ob es vielleicht doch Algorithmen mit polynomieller Laufzeit für das Problem gibt.

Zwei sinnvolle Erweiterungen

- $L(\mathtt{aa*}) = \{\mathtt{a}\} \cdot \{\mathtt{a}\}^* \overset{\text{Kap. } 7}{=} \{\mathtt{a}\}^+$

Zwei sinnvolle Erweiterungen

- $L(\texttt{aa}*) = \{\texttt{a}\} \cdot \{\texttt{a}\}^* \overset{\text{Kap. } 7}{=} \{\texttt{a}\}^+$
  Definiere $R+$ als Abkürzung für $(RR*)$ mit $L(R+) = L(R)^+$

Zwei sinnvolle Erweiterungen

- $L(\mathtt{aa*}) = \{\mathtt{a}\} \cdot \{\mathtt{a}\}^* \overset{\text{Kap. 7}}{=} \{\mathtt{a}\}^+$
  Definiere $R\mathtt{+}$ als Abkürzung für $(RR\mathtt{*})$ mit $L(R\mathtt{+}) = L(R)^+$

- $L(\mathtt{a}\,|\,\mathtt{\emptyset*}) = \{\mathtt{a}\} \cup \{\ \}^* = \{\mathtt{a}\} \cup \{\varepsilon\}$

Zwei sinnvolle Erweiterungen

- $L(\texttt{a}\texttt{a}\texttt{*}) = \{\texttt{a}\} \cdot \{\texttt{a}\}^* \overset{\text{Kap. 7}}{=} \{\texttt{a}\}^+$
  Definiere $R\texttt{+}$ als Abkürzung für $(RR\texttt{*})$ mit $L(R\texttt{+}) = L(R)^+$

- $L(\texttt{a}\texttt{|}\emptyset\texttt{*}) = \{\texttt{a}\} \cup \{\ \}^* = \{\texttt{a}\} \cup \{\varepsilon\}$
  Definiere $R\texttt{?}$ als Abkürzung für $(R\texttt{|}\emptyset\texttt{*})$ mit $L(R\texttt{?}) = L(R) \cup \{\varepsilon\}$

Wo sind wir?

`grep` in action

`grep` = **g**lobal search for **r**egular **e**xpressions and **p**rint

grep in action

Karlsruher Institut für Technologie

`grep` = **g**lobal search for **r**egular **e**xpressions and **p**rint

- `' i = '` Suche nach Zuweisungen an die Variable `i`
- `' (i|j) = '` Suche nach Zw. an die Variable `i` oder `j`
- `' [a-z] = '` Suche nach Zw. an eine Einbuchstaben-Variable
- `' [a-z][a-z] = '` Suche nach Zw. an eine Zweibuchstaben-Variable
- `' [a-z]+ *= *'` Suche nach Zw. an Variable (Kleinbchstaben)
- `'"[a-z]+"'` Suche nach Zeichenkette aus Kleinbuchstaben
- `'".*"'` Falsche Suche nach Zeichenketten
- `'"[^"]*"'` Suche nach Zeichenketten

Kommando: `grep -P -r 'reg. Ausdruck' *`

Wo sind wir?

Karlsruher Institut für Technologie

Reguläre Ausdrücke
- Definition
- Semantik regulärer Ausdrücke
- Beispiel: `grep`
- Zusammenhang mit Automaten und Grammatiken

Rechtslineare Grammatiken (Typ 3)

Kantorowitsch-Bäume und strukturelle Induktion

Charakterisierungen regulärer Sprachen

Satz

Für jede formale Sprache L sind äquivalent:

1. L kann von einem endlichen Akzeptor erkannt werden.
2. L kann durch einen regulären Ausdruck beschrieben werden.
3. L kann von einer rechtslinearen Grammatik erzeugt werden.

Sprachen mit diesen Eigenschaften heißen regulär.

- Beweis in «Theoretische Grundlagen der Informatik»
- rechtslineare Grammatiken (kommen gleich)
  - sind Spezialfall kontextfreier Grammatiken
  - also ist jede reguläre Sprache auch kontextfrei
  - *aber nicht umgekehrt*

Was ist wichtig

- Das sollten Sie mitnehmen:
  - Definition «klassischer» regulärer Ausdrücke
    - atomare: $\emptyset$, $a \in A$
    - zusammengesetzte: $(R_1 \mid R_2)$, $(R_1 R_2)$, $(R)*$
  - wissen: reguläre Ausdrücke und die Verallgemeinerung Regular Expressions z. B. bei Textverarbeitungsaufgaben manchmal nützlich
- Das sollten Sie üben:
  - zu L ein R mit $L(R) = L$ finden
  - zu R das $L(R)$ finden

Wo sind wir?

Karlsruher Institut für Technologie

Reguläre Ausdrücke

Rechtslineare Grammatiken (Typ 3)

Kantorowitsch-Bäume und strukturelle Induktion

Motivation

- kontextfreie Grammatiken erzeugen mehr formale Sprachen, als man mit endlichen Akzeptoren erkennen kann.
- Beispiel:
  - $G = (\{X\}, \{\mathtt{a}, \mathtt{b}\}, X, \{X \to \mathtt{a}X\mathtt{b} \mid \varepsilon\})$ erzeugt $\{\mathtt{a}^k\mathtt{b}^k \mid k \in \mathbb{N}_0\}$
  - und diese Sprache ist nicht regulär.
- Kann man kontextfreie Grammatiken so einschränken, dass sie zu endlichen Akzeptoren passen?

Rechtslineare Grammatiken: Definition

- *rechtslineare Grammatik* ist eine kontextfreie Grammatik $G = (N, T, S, P)$ mit folgenden Einschränkungen:
  Jede Produktion ist
  - entweder von der Form $X \to w$ mit $w \in T^*$
  - oder von der Form $X \to wY$ mit $w \in T^*$ und $X, Y \in N$.
- also auf jeder rechten Seite
  - höchstens ein Nichterminalsymbol
  - und wenn dann nur als letztes Symbol

Rechtslineare Grammatiken: Beispiele

- $G = (\{X\}, \{\mathtt{a}, \mathtt{b}\}, X, \{X \to \mathtt{ab}X \mid \mathtt{bba}X \mid \varepsilon\}$

Rechtslineare Grammatiken: Beispiele

- $G = (\{X\}, \{\texttt{a}, \texttt{b}\}, X, \{X \to \texttt{ab}X \mid \texttt{bba}X \mid \varepsilon\}$

  $L(G) = L(\texttt{(ab|bba)*})$

Rechtslineare Grammatiken: Beispiele

- $G = (\{X\}, \{\mathtt{a}, \mathtt{b}\}, X, \{X \to \mathtt{ab}X \mid \mathtt{bba}X \mid \varepsilon\}$

  $L(G) = L(\mathtt{(ab|bba)*})$

- $G = (\{X, Y\}, \{\mathtt{a}, \mathtt{b}\}, X,$
  $\{X \to \mathtt{a}X \mid \mathtt{b}X \mid \mathtt{ababb}Y, Y \to \mathtt{a}Y \mid \mathtt{b}Y \mid \varepsilon\}$

Rechtslineare Grammatiken: Beispiele

- $G = (\{X\}, \{\texttt{a}, \texttt{b}\}, X, \{X \to \texttt{ab}X \mid \texttt{bba}X \mid \varepsilon\}$

  $L(G) = L(\texttt{(ab|bba)*})$

- $G = (\{X, Y\}, \{\texttt{a}, \texttt{b}\}, X,$
  $\{X \to \texttt{a}X \mid \texttt{b}X \mid \texttt{ababb}Y, Y \to \texttt{a}Y \mid \texttt{b}Y \mid \varepsilon\}$

  $L(G) = L(\texttt{(a|b)*ababb(a|b)*})$

Rechtslineare Grammatiken: Nichtbeispiel

- $G = (\{X\}, \{\mathtt{a}, \mathtt{b}\}, X, \{X \to \mathtt{a}X\mathtt{b} \mid \varepsilon\})$ ist nicht rechtslinear,
  - denn in $X \to \mathtt{a}X\mathtt{b}$ steht das Nichtterminalsymbol X nicht am rechten Ende.
- Da die erzeugte formale Sprache $\{\mathtt{a}^k\mathtt{b}^k \mid k \in \mathbb{N}_0\}$ von keinem endlichen Akzeptor erkannt wird, kann es auch gar keine rechtslineare Grammatik geben.

Rechtslineare Grammatiken: Dochbeispiel

- $G = (\{X\}, \{\mathtt{a}\}, X, \{X \to \mathtt{a}X\mathtt{a} \mid \varepsilon\})$ ist nicht rechtslinear,
  - denn in $X \to \mathtt{a}X\mathtt{a}$ steht das Nichtterminalsymbol X nicht am rechten Ende.
- Dennoch ist die erzeugte formale Sprache $\{\mathtt{a}^{2k} \mid k \in \mathbb{N}_0\}$ aber ist regulär! …
- …denn es gibt eine rechtslineare Grammatik $G' = (\{X\}, \{\mathtt{a}\}, X, \{X \to \mathtt{a}\mathtt{a}X \mid \varepsilon\})$ mit $L(G) = L(G')$.
- Außerdem $L((\mathtt{aa})*) = L(G)$ und ein endl. Akzeptor für $L(G)$ ist eine leichte Übung.

Sprechweisen

- Rechtslineare Grammatiken heißen auch *Typ-3-Grammatiken* (T3G).
- Kontextfreie Grammatiken heißen auch *Typ-2-Grammatiken* (T2G).
- Es gibt auch noch
  - *Typ-1-Grammatiken* und
  - *Typ-0-Grammatiken*,

  die wir hier nicht weiter betrachten werden.
- Wenn für ein $i \in \{0, 1, 2, 3\}$ eine formale Sprache L von einer Typ-i-Grammatik erzeugt wird, dann heißt auch L eine *Typ-i-Sprache* oder kurz *vom Typ i*.

Vorteil rechtslinearer Grammatiken

- gegenüber deterministischen endlichen Akzeptoren:
  manchmal deutlich kürzer und übersichtlicher hinzuschreiben
- genaueres Verständnis dafür: im 3. Semester
  bei nichtdeterministischen endliche Akzeptoren

Wo sind wir?

Reguläre Ausdrücke

Rechtslineare Grammatiken (Typ 3)

Kantorowitsch-Bäume und strukturelle Induktion

Ziel dieses Abschnittes

Beweisskizze für das folgende

Lemma

Zu jedem regulären Ausdruck R gibt es eine rechtslineare Grammatik G mit $L(G) = L(R)$.

- Wie beweist man, dass eine Aussage für alle regulären Ausdrücke gilt?
- eine Möglichkeit: *strukturelle Induktion*
  - Variante/Verallgemeinerung vollständiger Induktion,
  - ohne explizit über natürliche Zahlen zu sprechen
- darauf arbeiten wir jetzt in mehreren Schritten hin:
  - Darstellung regulärer Ausdrücke mit Bäumen
  - eine Variante «normaler vollständiger Induktion»
  - strukturelle Induktion

Induktive Strukturen

- Wir untersuchen eine Menge H.
- Gegeben seien eine Menge von Basisfällen $B \subseteq H$
- und eine Menge von Konstruktoren (Funktionen)
  $K = \{k_1 : H^{n_1} \to H, \ \ldots, \ k_r : H^{n_r} \to H\}$
- H heißt induktiv generiert aus B und K, wenn jedes Element aus H durch einen Ausdruck über B und K dargestellt werden kann.

Beispiele:

- Natürliche Zahlen: $B_{\mathbb{N}} = \{0\}, \quad K_{\mathbb{N}} = \{(+1) : \mathbb{N}_0 \to \mathbb{N}_0\}$
- Aussagenlogik: $B_{AL} = Var_{AL}, \quad K_{AL} = \{\wedge, \vee, \to, \neg\}$

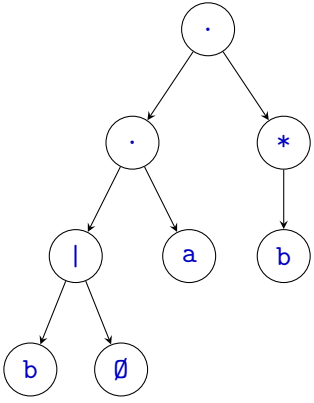
Induktive Strukturen

- Wir untersuchen eine Menge H.
- Gegeben seien eine Menge von Basisfällen $B \subseteq H$
- und eine Menge von Konstruktoren (Funktionen)
  $K = \{k_1 : H^{n_1} \to H, \ \ldots, \ k_r : H^{n_r} \to H\}$
- H heißt induktiv generiert aus B und K, wenn jedes Element aus H durch einen Ausdruck über B und K dargestellt werden kann.

Beispiele:

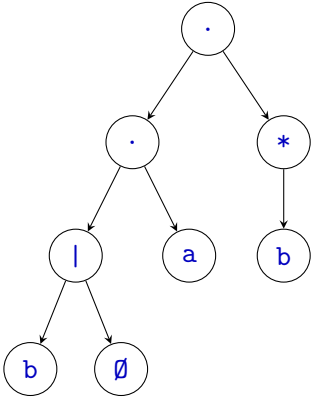
- Natürliche Zahlen: $B_{\mathbb{N}} = \{0\}, \quad K_{\mathbb{N}} = \{(+1) : \mathbb{N}_0 \to \mathbb{N}_0\}$
- Aussagenlogik: $B_{AL} = Var_{AL}, \quad K_{AL} = \{\wedge, \vee, \to, \neg\}$
- Reguläre Ausdrücke: $B_{RE} = \{\emptyset\} \cup A, \quad K_{RE} = \{*, |, \cdot\}$

Mit Kantorowitsch-Bäumen kann man z. B. reguläre Ausdrücke repräsentieren

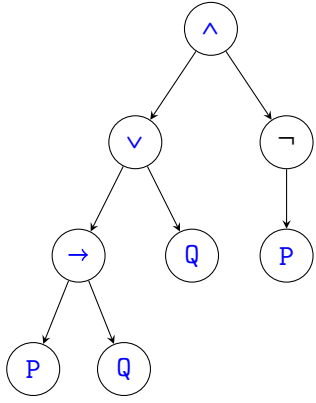


- regulärer Ausdruck: `((b|∅)a)(b*)`
- Darstellung als sogenannter Kantorowitsch-Baum
  - innere Knoten: Konstruktoren (entspr. Ausgangsgrad)
  - Blätter: Basisfälle
- «Abstrakter Syntaxbaum»
- hier: «Regex-Baum»
- Beachte:
  - das ist *kein* Ableitungsbaum gemäß einer Grammatik
  - aber «genauso gut» und kompakter

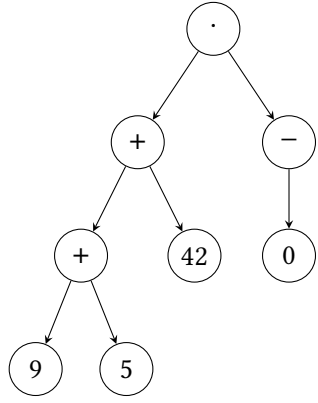
Kantorowitsch-Bäumen für induktive Strukturen



regulärer Ausdruck

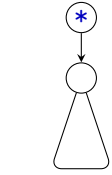


aussagenlog. Formel



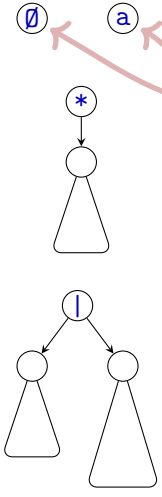
arithm. Ausdruck

Regex-Bäume — etwas genauer



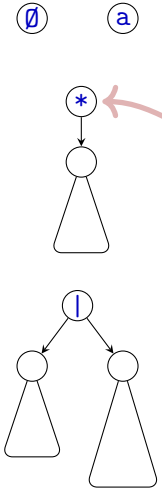
- Es sei A Alphabet.
- Ein Baum ist ein Regex-Baum, wenn gilt:
  - entweder Wurzel ist Blatt, mit $x \in A$ oder $\emptyset$ beschriftet,
  - oder Wurzel mit $*$ beschriftet und hat genau einen Nachfolger, der Wurzel eines Regex-Baumes ist
  - oder Wurzel mit $\cdot$ oder $|$ beschriftet und hat genau zwei Nachfolger, die Wurzeln zweier Regex-Bäume sind.

Regex-Bäume — etwas genauer



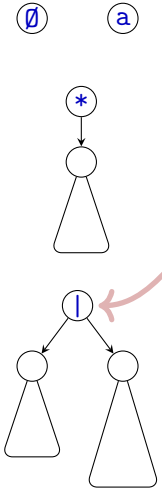
- Es sei A Alphabet.
- Ein Baum ist ein Regex-Baum, wenn gilt:
  - entweder Wurzel ist Blatt, mit $x \in A$ oder $\emptyset$ beschriftet,
  - oder Wurzel mit $*$ beschriftet und hat genau einen Nachfolger, der Wurzel eines Regex-Baumes ist
  - oder Wurzel mit $\cdot$ oder $|$ beschriftet und hat genau zwei Nachfolger, die Wurzeln zweier Regex-Bäume sind.

Regex-Bäume — etwas genauer



- Es sei A Alphabet.
- Ein Baum ist ein Regex-Baum, wenn gilt:
  - entweder Wurzel ist Blatt, mit $x \in A$ oder $\emptyset$ beschriftet,
  - oder Wurzel mit $*$ beschriftet und hat genau einen Nachfolger, der Wurzel eines Regex-Baumes ist
  - oder Wurzel mit $\cdot$ oder $|$ beschriftet und hat genau zwei Nachfolger, die Wurzeln zweier Regex-Bäume sind.

Regex-Bäume — etwas genauer



- Es sei A Alphabet.
- Ein Baum ist ein Regex-Baum, wenn gilt:
  - entweder Wurzel ist Blatt, mit $x \in A$ oder $\emptyset$ beschriftet,
  - oder Wurzel mit $*$ beschriftet und hat genau einen Nachfolger, der Wurzel eines Regex-Baumes ist
  - oder Wurzel mit $\cdot$ oder $|$ beschriftet und hat genau zwei Nachfolger, die Wurzeln zweier Regex-Bäume sind.

Das Prinzip der strukturellen (vollständigen) Induktion

Sei H eine induktive Struktur generiert aus Basisfällen B und Konstruktoren K.

Eine Aussage $\mathcal{B}$ für alle $h \in H$ kann mit struktureller (vollständiger) Induktion bewiesen werden durch:

$b \in B \implies \mathcal{B}(b)$

1. **Induktionsanfang** *zu zeigen:* Für alle $b \in B$ gilt $\mathcal{B}(b)$

$\mathcal{B}(h_1) \wedge \ldots \wedge \mathcal{B}(h_n)$
$\implies$
$\mathcal{B}(k(h_1, h_2, \ldots, h_n))$

2. **Induktionsschritt** *zu zeigen:* Für alle $k : H^n \to H \in B$ gilt:
   Für beliebige $h_1, h_2, \ldots, h_n \in H$ gilt
   $\bigwedge_{i=1}^{n} \mathcal{B}(h_i) \implies \mathcal{B}(k(h_1, h_2, \ldots, h_n))$

Beispiele für strukturelle Induktion

- Man sagt meistens «strukturelle Induktion».

Beispiele für strukturelle Induktion

- Man sagt meistens «strukturelle Induktion».
- über **natürliche Zahlen** ($B = \{0\}, K = (+1)$)
  - Induktionsanfang: $\mathcal{B}(0)$
  - Induktionsschritt: $\mathcal{B}(n) \implies \mathcal{B}(n+1)$
  - = „herkömmliche” vollständige Induktion

Beispiele für strukturelle Induktion

- Man sagt meistens «strukturelle Induktion».
- über **natürliche Zahlen** ($B = \{0\}, K = (+1)$)
- über **aussagenlogische Formeln**
  - Induktionsanfang: $\mathcal{B}(\mathsf{P}_i)$ für alle $\mathsf{P}_i \in \Sigma$
  - Induktionsschritt: $\mathcal{B}(F_1)$ und $\mathcal{B}(F_2) \implies$:
    1. $\mathcal{B}(F_1 \wedge F_2)$
    2. $\mathcal{B}(F_1 \vee F_2)$
    3. $\mathcal{B}(F_1 \to F_2)$
    4. $\mathcal{B}(\neg F_1)$

Beispiele für strukturelle Induktion

- Man sagt meistens «strukturelle Induktion».
- über **natürliche Zahlen** ($B = \{0\}, K = (+1)$)
- über **aussagenlogische Formeln**
- über **reguläre Ausdrücke**
  - Induktionsanfang:
    1. $\mathcal{B}(\mathtt{x})$ für alle $\mathtt{x} \in A$
    2. $\mathcal{B}(\emptyset)$
  - Induktionsschritt: $\mathcal{B}(R_1)$ und $\mathcal{B}(R_2) \implies$:
    1. $\mathcal{B}(R_1 R_2)$
    2. $\mathcal{B}(R_1 \mid R_2)$
    3. $\mathcal{B}(R_1 *)$

Beispiele für strukturelle Induktion

- Man sagt meistens «strukturelle Induktion».
- über **natürliche Zahlen** ($B = \{0\}, K = (+1)$)
- über **aussagenlogische Formeln**
- über **reguläre Ausdrücke**

Strukturelle Induktion über Regex-Bäume

Aussage $\mathcal{B}$:

Zu jedem regulären Ausruck R über Alphabet A gibt es eine rechtslineare Grammatik (T3G) G mit $L(R) = L(G)$.

Strukturelle Induktion über Regex-Bäume

Aussage $\mathcal{B}$:

Zu jedem regulären Ausruck R über Alphabet A gibt es eine rechtslineare Grammatik (T3G) G mit $L(R) = L(G)$.

- **Induktionsanfang**
  1. $R = \emptyset$, also $L(R) = \emptyset$
  2. $R = \mathtt{x}$ für $\mathtt{x} \in A$, also $L(R) = \{\mathtt{x}\}$.
     T3G hierfür: $\longrightarrow$ Übung

Strukturelle Induktion über Regex-Bäume

Aussage $\mathcal{B}$:

Zu jedem regulären Ausruck R über Alphabet A gibt es eine rechtslineare Grammatik (T3G) G mit $L(R) = L(G)$.

- **Induktionsanfang**
  1. $R = \emptyset$, also $L(R) = \emptyset$
  2. $R = \mathtt{x}$ für $\mathtt{x} \in A$, also $L(R) = \{\mathtt{x}\}$.
     T3G hierfür: $\longrightarrow$ Übung
- **Induktionsschritt** Seien R_1, R_2 beliebige reg. Ausdrücke über A
  Induktionsannahme: es gibt T3G G_1, G_2 mit $L(G_1) = L(R_1)$ und $L(G_2) = L(R_2)$.

Strukturelle Induktion über Regex-Bäume

Aussage $\mathcal{B}$:

Zu jedem regulären Ausruck R über Alphabet A gibt es eine rechtslineare Grammatik (T3G) G mit $L(R) = L(G)$.

- **Induktionsanfang**
  1. $R = \emptyset$, also $L(R) = \emptyset$
  2. $R = \mathtt{x}$ für $\mathtt{x} \in A$, also $L(R) = \{\mathtt{x}\}$.
     T3G hierfür: $\longrightarrow$ Übung
- **Induktionsschritt** Seien R_1, R_2 beliebige reg. Ausdrücke über A
  Induktionsannahme: es gibt T3G G_1, G_2 mit $L(G_1) = L(R_1)$ und $L(G_2) = L(R_2)$.
  zu zeigen:
  1. Es gibt T3G G_3 mit $L(G_3) = L(R_1 \mid R_2)$
  2. Es gibt T3G G_4 mit $L(G_4) = L(R_1 R_2)$ $\longrightarrow$ Übung
  3. Es gibt T3G G_5 mit $L(G_5) = L(R_1{*})$ $\longrightarrow$ Übung

Strukturelle Induktion über Regex-Bäume

Aussage $\mathcal{B}$:

Zu jedem regulären Ausruck R über Alphabet A gibt es eine rechtslineare Grammatik (T3G) G mit $L(R) = L(G)$.

$\mathcal{B}(\emptyset)$
$\mathcal{B}(\mathtt{x})$

- **Induktionsanfang**
  1. $R = \emptyset$, also $L(R) = \emptyset$
  2. $R = \mathtt{x}$ für $\mathtt{x} \in A$, also $L(R) = \{\mathtt{x}\}$.
     T3G hierfür: ⟶ Übung

IA: $\mathcal{B}(R_1), \mathcal{B}(R_2)$

z.z.:
$\mathcal{B}(R_1 \,|\, R_2)$
$\mathcal{B}(R_1 R_2)$
$\mathcal{B}(R_1 *)$

- **Induktionsschritt** Seien R_1, R_2 beliebige reg. Ausdrücke über A
  Induktionsannahme: es gibt T3G G_1, G_2 mit $L(G_1) = L(R_1)$ und $L(G_2) = L(R_2)$.
  zu zeigen:
  1. Es gibt T3G G_3 mit $L(G_3) = L(R_1 \,|\, R_2)$
  2. Es gibt T3G G_4 mit $L(G_4) = L(R_1 R_2)$ ⟶ Übung
  3. Es gibt T3G G_5 mit $L(G_5) = L(R_1 *)$ ⟶ Übung

Skizze eines Induktionsschritts

- Erster Fall: $R_1 \mid R_2$

Skizze eines Induktionsschritts

- Erster Fall: $R_1 \,|\, R_2$
- nach Induktionsvoraussetzung gibt es T3G $G_1 = (N_1, A, S_1, P_1)$ und $G_2 = (N_2, A, S_2, P_2)$ mit $L(G_1) = L(R_1)$ bzw. $L(G_2) = L(R_2)$
- es sei $N_1 \cap N_2 = \emptyset$
  - keine Beschränkung der Allgemeinheit
  - wichtig für die Konstruktion

Skizze eines Induktionsschritts

- Erster Fall: $R_1 \,|\, R_2$
- nach Induktionsvoraussetzung gibt es T3G $G_1 = (N_1, A, S_1, P_1)$ und $G_2 = (N_2, A, S_2, P_2)$ mit $L(G_1) = L(R_1)$ bzw. $L(G_2) = L(R_2)$
- es sei $N_1 \cap N_2 = \emptyset$
  - keine Beschränkung der Allgemeinheit
  - wichtig für die Konstruktion
- wähle «neues» Nichtterminalsymbol $S \notin N_1 \cup N_2$
- Behauptung: $G = (\{S\} \cup N_1 \cup N_2, A, S, \{S \to S_1 \mid S_2\} \cup P_1 \cup P_2)$ ist T3G mit $L(G) = L(R_1 \,|\, R_2)$

Skizze eines Induktionsschritts

- Erster Fall: $R_1 \,|\, R_2$
- nach Induktionsvoraussetzung gibt es T3G $G_1 = (N_1, A, S_1, P_1)$ und $G_2 = (N_2, A, S_2, P_2)$ mit $L(G_1) = L(R_1)$ bzw. $L(G_2) = L(R_2)$
- es sei $N_1 \cap N_2 = \emptyset$
  - keine Beschränkung der Allgemeinheit
  - wichtig für die Konstruktion
- wähle «neues» Nichtterminalsymbol $S \notin N_1 \cup N_2$
- Behauptung: $G = (\{S\} \cup N_1 \cup N_2, A, S, \{S \to S_1 \mid S_2\} \cup P_1 \cup P_2)$ ist T3G mit $L(G) = L(R_1 \,|\, R_2)$
  - Rechtslinearität: klar
  - $L(G) = L(G_1) \cup L(G_2) \to$ Übung

Vollständige Induktion über die Baumhöhe

- Größere Bäume sind «aus kleineren zusammengesetzt», und zwar auf eindeutige Weise.
- Bijektion Regex-Bäume ↔ reguläre Ausdrücke
  Kantorowitsch-Bäume ↔ Elemente in induktiven Strukturen

Vollständige Induktion über die Baumhöhe

- Größere Bäume sind «aus kleineren zusammengesetzt», und zwar auf eindeutige Weise.
- Bijektion Regex-Bäume ↔ reguläre Ausdrücke
  Kantorowitsch-Bäume ↔ Elemente in induktiven Strukturen
- *Höhe* $h(T)$ eines Baumes

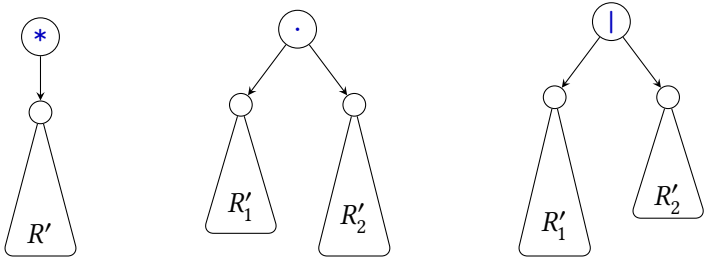
$$h(T) = \begin{cases} 0 & \text{falls die Wurzel Blatt ist} \\ 1 + \max_i h(U_i) & \text{falls die } U_i \text{ alle Unterbäume von } T \text{ sind} \end{cases}$$

Vollständige Induktion über die Baumhöhe

- zu naive Vorgehensweise:
  beim Schritt zu Bäumen der Höhe $n+1$
  Induktionsvoraussetzung nur für Bäume der Höhe n
- aber:
  die Unterbäume eines Baumes der Höhe $n+1$
  können beliebige Höhen $i \leq n$ haben.
- anschaulich: man darf auch für die kleineren Unterbäume
  die Induktionsvoraussetzung benutzen.
- starke Induktion: siehe Kapitel 6

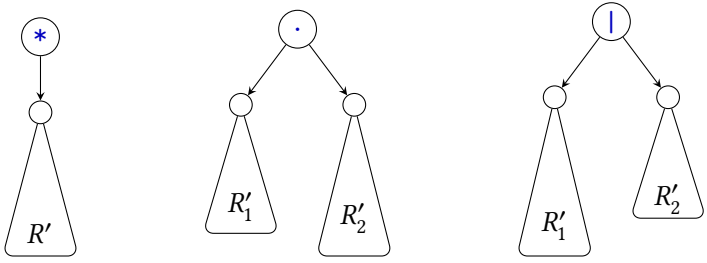
Skizze des Induktionsschritts (1)

- sei R beliebiger Regex-Baum der Höhe $n+1$:



Skizze des Induktionsschritts (1)

- sei R beliebiger Regex-Baum der Höhe $n+1$:



- für alle Unterbäume haben eine Höhe $\leq n$

Strukturelle Induktion = vollst. Induktion über Baumhöhe

Strukturelle Induktion

Induktionsanfang:
$\mathcal{B}$ gilt für alle Basisfälle B.

Induktionsschritt:
Für jeden Konstruktor $k : H^n \to H \in K$
Induktionsannahme: $\mathcal{B}$ gilt für $h_1, ..., h_n \in H$.
zu zeigen: $\mathcal{B}$ gilt für $k(h_1, ..., h_n)$

Vollständige Induktion über Höhe des Baums

Induktionsanfang:
$\mathcal{B}$ gilt für alle Elemente mit Baumhöhe $n = 0$.

Induktionsschritt:
Für einen Baum der Höhe $n > 0$
Induktionsannahme: $\mathcal{B}$ gilt für alle $h_0 \in H$ mit Baumhöhe $\leq n$
zu zeigen: Für jedes Element $h \in H$ mit Baumhöhe n gilt $\mathcal{B}$.
Durch Fallunterscheidung des Konstruktors in h. Für die Unterbäume von h trifft Induktionsannahme zu.

Diese beiden Beweisschemata können sich gegenseitig simulieren!

Was ist wichtig

- Das sollten Sie mitnehmen:
  - Definition rechtslineare Grammatiken
- Das sollten Sie üben:
  - rechtslineare Grammatiken konstruieren (zu gegebenem Akzeptor, regulären Ausdruck, formaler Sprache)
  - strukturelle Induktion

Zusammenfassung

- reguläre Ausdrücke
  - werden von diversen «Werkzeugen» unterstützt
  - in manchen Programmiersprachen zur Textverarbeitung praktisch
- rechtslineare Grammatiken
- strukturelle Induktion