Reguläre Ausdrücke

Bemerkung

1. *Reguläre Ausdrücke können geklammert werden, um die Reihenfolge der Operationen festzulegen. Um mit wenig Klammern auszukommen, vereinbart man Prioritäten, und zwar haben * und $^+$ die höchste Priorität, dann kommt das Produkt und schließlich die Vereinigung. Alle Operationen sind linksassoziativ.*
2. *Häufig werden zur Abkürzung regulären Ausdrücken Namen zugeordnet, die in anderen Ausdrücken benutzt werden können.*

Reguläre Ausdrücke

Satz

Die durch reguläre Ausdrücke bezeichneten Mengen sind reguläre Sprachen, und jede reguläre Sprache lässt sich durch einen regulären Ausdruck bezeichnen.

Reguläre Ausdrücke

Beispiel

$(a \mid b)^* abb$ — Wörter über $\{a, b\}$, die auf *abb* enden

$(0 \mid 1)^*\, 0\, (0 \mid 1)(0 \mid 1)$ — Binär-Zahlen mit Achter-Resten 0,1,2,3

$0\, (0 \mid 1)^*\, 0$ — Binär-Zahlen, die mit 0 beginnen und enden

$(A \mid B \mid \ldots \mid Z)(A \mid B \mid \ldots \mid Z \mid 0 \mid \ldots \mid 9)^*$ — Bezeichner

Der reguläre Ausdruck $(+ \mid - \mid \varepsilon)\ \text{<digit>}^+\ (\varepsilon \mid .\ \text{<digit>}^+)$ beschreibt Festkommazahlen (ggf. mit Vorzeichen).

Dabei ist $\text{<digit>} = 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$ ein Name für einen regulären Ausdruck.

Abkürzende Schreibweisen in regulären Ausdrücken

Definition

1. Null- oder einmaliges Vorkommen:
   Ein Ausdruck $r \mid \varepsilon$ wird zu $r?$ abgekürzt.
2. Terminalzeichen c_i:
   Für den Ausdruck $c_1 \mid c_2 \mid \dots \mid c_k$ schreibt man $[\, c_1 c_2 \dots c_k \,]$,
   bei im Alphabet *aufeinanderfolgenden* Zeichen: $[\, c_1 - c_k \,]$.

Reguläre Ausdrücke

Beispiel

Der reguläre Ausdruck für Festkommazahlen wird jetzt kürzer formuliert:

Statt

$$(+ \mid - \mid \varepsilon)\ <\text{digit}>^{+}\ (\varepsilon \mid .\ <\text{digit}>^{+})$$
$$<\text{digit}> = 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

nun

$$(+ \mid -)?\ <\text{digit}>^{+}\ (.\ <\text{digit}>^{+})?$$
$$<\text{digit}> = [0-9]$$

oder

$$(+ \mid -)?\, [0-9]^{+}\, (.\, [0-9]^{+})?$$

Von Zeichen zu Token

Für gängige Programmiersprachen nutzt man Scanner, um Zeichen zu Token zusammenzufassen, also zur Erkennung von

- reservierten Wörtern (**begin**, **for**, **if**, ...),
- Identifikatoren (Namen für z.B. Variable, Konstante, Typen),
- Konstanten (ganze Zahlen, Gleitpunktzahlen, Zeichenketten...),
- (zusammengesetzten) Sonderzeichen (:=, <=, ...) und
- Zwischenraum (*white space*)
  (als Folge von Leerzeichen, Tabulatoren, Zeilenwechsel)

Tokenklassen für eine einfache Programmiersprache

Beispiel (Tokenklassen für eine einfache Programmiersprache mit arithmetischen Ausdrücken, Wertzuweisungen, und if-Anweisungen)

Mit den Abkürzungen

```
<letter> = [A - Z] | [a - z]
<digit>  = [0 - 9]
```

können folgende Tokenklassen definiert werden:

ident (Identifier, Namen)	`<letter> (<letter>|<digit>)`*
number (vorzeichenlose Zahlen)	`<digit>`$^+$ `( .<digit>`$^+$ `)?`
relop (Vergleichsoperatoren)	`= | <= | <> | < | > | >=`
arithop (arithmetische Op.)	`+ | - | * | /`

Die Tokenklassen **if**, **then** und **else** enthalten jeweils das gleichnamige reservierte Wort.

Scanner-Generatoren

Verwendet man Scanner-Generatoren wie **LEX**, **FLEX** oder **JFLEX**, so definiert man reguläre Ausdrücke für die Tokenklassen und geeignete Aktionen, die beim Erkennen eines Tokens ausgeführt werden sollen.

Wichtig ist, dass *alle* Eingabezeichen zu Token verarbeitet werden.

Daher sollte man eine weitere Tokenklasse **ws** (*white space*) schaffen, die das Überlesen von Zwischenraum steuert.

`<delimiter> = <blank> | <tab> | <newline>`
und die Tokenklasse **ws** wird definiert durch
`<delimiter>`$^{+}$

Die zugeordnete Aktion ist in vielen Sprachen die Null-Operation!

Hat eine **Einrückung** aber eine Bedeutung in der Programmiersprache (z.B. Haskell, Python) (*off-side rule*), so erzeugt der Scanner
bei verstärkter Einrückung ein **`INDENT`**-Token und
bei verringerter Einrückung ein **`DEDENT`**-Token.

Scanner-Generatoren

Scanner-Generatoren erzeugen Scanner automatisch;
typisch ist folgendes Vorgehen:

- Erzeugung eines (nicht-determinstischen) Zustandübergangsgraphen eines endlichen Automaten für jeden regulären Ausdruck einer Tokenklasse.
- Entfernung der ε-Übergänge im nicht-deterministischen Zustandübergangsgraphen.
- Umwandlung des nicht-deterministischen Zustandübergangsgraphen ohne ε-Übergänge in einen deterministischen Zustandübergangsgraphen.
- Ableitung eines Scanners aus dem deterministischen Zustandübergangsgraphen.

Oft sind aber **direkt erzeugte Scanner** effizienter.

Oder der Parser erledigt diese Aufgabe gleich mit.

Das Parsing-Problem

Gegeben ist
eine kontextfreie Grammatik $G = (N, T, P, S)$ und
ein Wort $w \in T^*$.

Frage: Ist $w \in L(G)$? „Wortproblem für kontextfreie Sprachen“

- Das Problem ist entscheidbar,
  d.h. es gibt einen Algorithmus, der diese Frage beantwortet.
- Der Cocke-Younger-Kasami-Algorithmus läuft in einer Zeit $O(|w|^3)$ (für kontextfreie Grammatiken in Chomsky-Normalform).
- Wir suchen ein lineares Verfahren!

Deterministische linksableitende Syntaxanalyse

Das Verfahren zur **linksableitenden Syntaxanalyse** (*top-down parsing*) soll

1. das gegebene Wort w nur einmal von links nach rechts lesen.
2. eine **Linksableitung** von w erzeugen.
   Das bedeutet, dass der Ableitungsbaum von oben nach unten und von links nach rechts konstruiert wird.

Außerdem soll das Verfahren **deterministisch** arbeiten:

3. Muss in der Linksableitung das nichtterminale Symbol A als nächstes verarbeitet werden, so ist die anzuwendende A-Produktion durch das nächste zu lesende Symbol von w (**Vorschau-Symbol**, *lookahead*) eindeutig bestimmt.

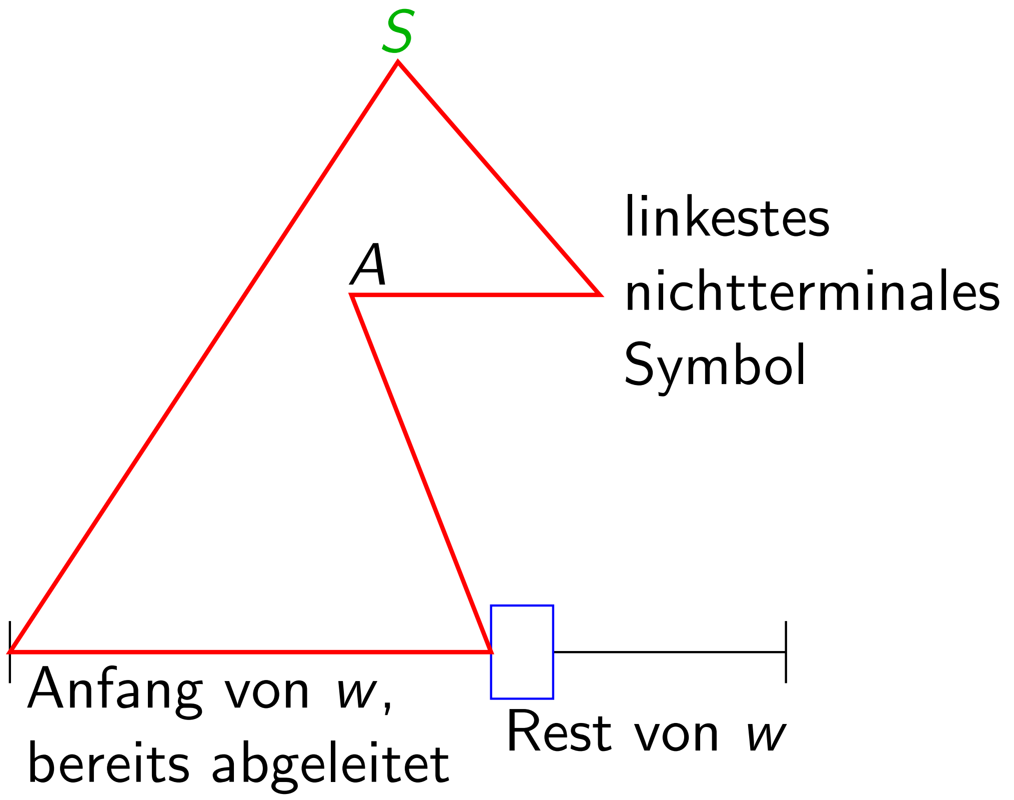
Anfangssituation bei der linksableitenden Syntaxanalyse

S Startsymbol der Grammatik

abzuleitendes Wort w

Lesefenster für Vorschau-Symbol,
zeigt zu Anfang auf das erste Symbol der Eingabe

Situation bei der linksableitenden Syntaxanalyse



Bemerkung

1. *Nicht alle Grammatiken erlauben eine Syntaxanalyse dieser Art.*

Beispiel

$$E \to E + E \mid E * E \mid a$$

$w = a + a$ Startsymbol E Vorschau a Welche Produktion anwenden?

2. *Es gibt sogar kontextfreie Sprachen, für die es keine Grammatik gibt, die eine determinstische linksableitende Syntaxanalyse ermöglicht:*

$$\{a^n 0 b^n \mid n \geq 0\} \cup \{a^n 1 b^{2n} \mid n \geq 0\}$$

Kfr. Produktionen: $\{S \to A \mid B, \quad A \to aAb \mid 0, \quad B \to aBbb \mid 1\}$

Aber eine deterministische linksableitende Syntaxanalyse ist stets möglich, falls für jedes nichtterminale Zeichen A gilt:
Alle rechten Seiten der A-Produktionen beginnen mit *verschiedenen terminalen* Zeichen (**s-Grammatik**).
(Das leere Wort wird nur direkt aus dem Startsymbol abgeleitet, das dann nicht auf der rechten Seite einer Produktion vorkommen darf.)

Hindernisse für deterministisches Top-Down Parsing

Eigenschaften von Grammatiken, die eine deterministische linksableitende Syntaxanalyse offensichtlich verhindern

1. Es gibt verschiedene Produktionen mit der gleichen linken Seite, deren rechte Seiten ein gleiches Anfangsstück (*Präfix*) haben.
2. Es gibt *linksrekursive Produktionen*, d.h. Produktionen der Form $A \to A\alpha,\ \alpha \in (N \cup T)^+$.

Entfernen gemeinsamer Präfixe

$$A \to \alpha\beta_1 \mid \ldots \mid \alpha\beta_r \mid \gamma_1 \mid \ldots \mid \gamma_s$$

seien alle A-Produktionen der Grammatik,
wobei $\alpha \neq \varepsilon$ und kein γ_i das Präfix α hat.

Ersetze diese Produktionen durch

$$\begin{array}{lcl} A & \to & \alpha A' \mid \gamma_1 \mid \ldots \mid \gamma_s \quad \text{und} \\ A' & \to & \beta_1 \mid \ldots \mid \beta_r \ , \end{array}$$

wobei A' ein neues nichtterminales Symbol ist.

Man sieht leicht: Die umgeformte Grammatik erzeugt die gleiche Sprache.

Entfernen gemeinsamer Präfixe

Beispiel

$S \to a + S \mid a * S \mid (S) * S \mid a$

$$\begin{aligned} S &\to aS' \mid (S) * S \quad \text{und} \\ S' &\to +S \mid *S \mid \varepsilon \end{aligned}$$

Entfernen linksrekursiver Produktionen

$$A \to A\alpha_1 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \ldots \mid \beta_n$$

seien alle A-Produktionen,
wobei kein β_i mit A beginnt und alle $\alpha_i \neq \varepsilon$ sind.

Ersetze diese Produktionen durch:

$$\begin{array}{rcl} A & \to & \beta_1 A' \mid \ldots \mid \beta_n A' \quad \text{und} \\ A' & \to & \alpha_1 A' \mid \ldots \mid \alpha_m A' \mid \varepsilon \ , \end{array}$$

wobei A' ein neues nichtterminales Symbol ist.

Bemerkung

Diese Umformungen müssen nicht unbedingt zu einer Grammatik führen, die deterministisches Parsing erlaubt.

Entfernen linksrekursiver Produktionen

Beispiel (durch Komma getrennte Liste L von Werten v)

$L \to v \mid L, v$

$$\begin{aligned} L &\to vR \quad \text{und} \\ R &\to \, ,vR \mid \varepsilon \end{aligned}$$

Beispiel (gültige Klammerungen)

$S \to SS \mid (S) \mid [S] \mid \varepsilon$

$$\begin{aligned} S &\to (S)S' \mid [S]S' \mid \varepsilon S' \quad \text{und} \\ S' &\to SS' \mid \varepsilon \end{aligned}$$

Definition der Funktionen **First** und **Follow**

Wann erlaubt eine kontextfreie Grammatik eine deterministische linksableitende Syntaxanalyse?

Definition

Sei $\alpha \in (N \cup T)^*$. Dann ist

$$\textbf{First}(\alpha) = \{a \in T \mid \alpha \stackrel{*}{\Longrightarrow} a\beta,\ \beta \in (N \cup T)^*\} \cup \{\varepsilon \mid \alpha \stackrel{*}{\Longrightarrow} \varepsilon\}$$

Bemerkung

First(α) *enthält diejenigen terminalen Zeichen, die bei Linksableitungen von* α *als erste auftreten.*
Kann man aus α *das leere Wort* ε *herleiten, so ist* ε *ebenfalls in* **First**(α).

Beispiel

$S \to aS' \mid (S) * S \qquad S' \to +S \mid *S \mid \varepsilon$

$\textbf{First}(S) = \{a, (\} \qquad \textbf{First}(S') = \{+, *, \varepsilon\}$

Definition der Funktionen **First** und **Follow**

Definition

Sei $A \in N$ und \$ ein neues Symbol, das das Ende der Eingabe markiert.

$$\begin{aligned}\textbf{Follow}(A) &= \{a \in T \mid S \overset{*}{\Longrightarrow} \alpha A a \beta,\ \alpha, \beta \in (N \cup T)^*\} \\ &\quad \cup \{\$ \mid S \overset{*}{\Longrightarrow} \alpha A,\ \alpha \in (N \cup T)^*\}\end{aligned}$$

Bemerkung

Follow(A) *enthält die terminalen Zeichen, die bei einer Ableitung vom Startsymbol S direkt auf A folgen können.*
Ist A das letzte Zeichen, so ist die Endmarke \$ in **Follow**(A).

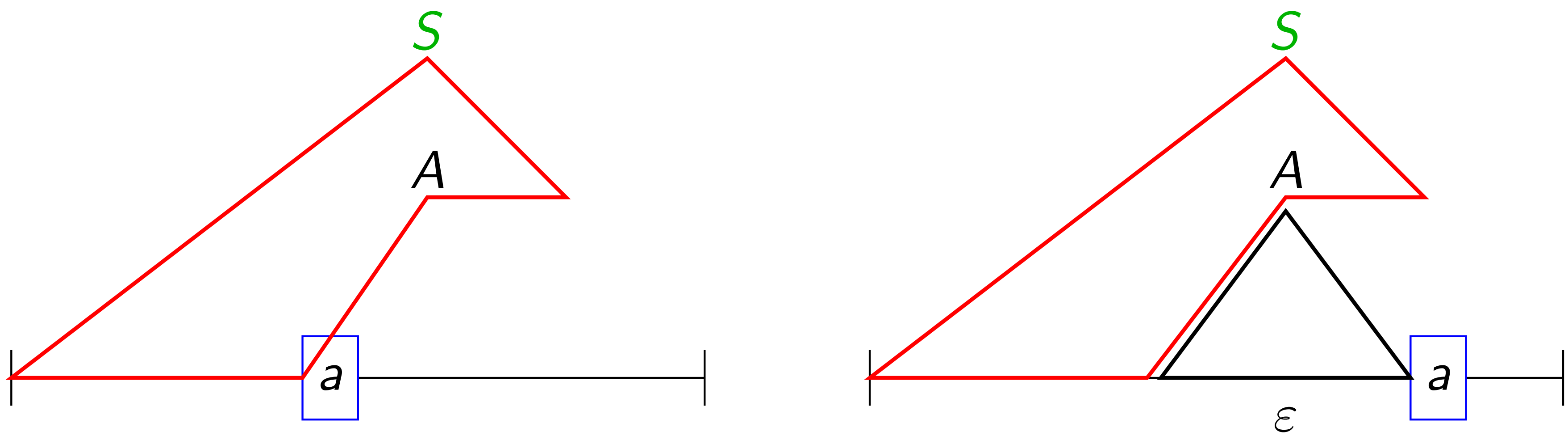
Beispiel

$S \to aS' \mid (S) * S \qquad S' \to +S \mid *S \mid \varepsilon$

Follow$(S) = \{\$,)\}$ **Follow**$(S') = \{\$,)\}$

Linksableitung eines Wortes

Betrachten wir eine Linksableitung eines Wortes der Sprache und sei A das linkeste nichtterminale Zeichen, es ist also als nächstes eine A-Produktion anzuwenden:



erwartete Vorschau-Symbole einer Produktion

Wir bestimmen zunächst mit Hilfe der Grammatik die Menge der erwarteten Symbole, die in einer Linksableitung bei Anwendung einer Produktion $A \rightarrow \alpha$ als Vorschau-Symbol vorkommen können:

$$\textbf{Erwartet}(A \rightarrow \alpha) = \begin{cases} \textbf{First}(\alpha) & \text{, falls } \varepsilon \notin \textbf{First}(\alpha) \\ (\textbf{First}(\alpha) \backslash \{\varepsilon\}) \cup \textbf{Follow}(A) & \text{, falls } \varepsilon \in \textbf{First}(\alpha) \end{cases}$$

Das Vorschau-Symbol des zu testenden Wortes soll die anzuwendende A-Produktion *eindeutig* bestimmen, dazu müssen die Vorschau-Mengen aller A-Produktionen paarweise disjunkt sein.

LL(1) Grammatik

Definition

Eine kontextfreie Grammatik $G = (N, T, P, S)$ heißt **LL(1)-Grammatik**
(Lesen der Eingabe von **L***inks nach rechts,*
Erzeugen einer **L***inksableitung und*
nur **1** *Zeichen als Vorschau),*
falls für alle $A \in N$ gilt:

Seien $A \to \alpha_1 \mid \ldots \mid \alpha_n$ alle A-Produktionen in P.

1. $\textbf{First}(\alpha_1), \ldots, \textbf{First}(\alpha_n)$ sind paarweise disjunkt, d.h. $\textbf{First}(\alpha_i) \cap \textbf{First}(\alpha_j) = \emptyset$ falls $i \neq j$.
2. Ist $\varepsilon \in \textbf{First}(\alpha_j)$, dann ist $\textbf{Follow}(A) \cap \textbf{First}(\alpha_i) = \emptyset$ für alle $1 \leq i \leq n$, $i \neq j$.

LL(1) Grammatik

Beispiel

$S \to aAaa \mid bAba \quad A \to b \mid \varepsilon$

S: $\textbf{First}(aAaa) = \{a\}$, $\textbf{First}(bAba) = \{b\}$
Bedingung 1 erfüllt, Bedingung 2 trivial!

A: $\textbf{First}(b) = \{b\}$, $\textbf{First}(\varepsilon) = \{\varepsilon\}$
Bedingung 1 erfüllt, Bedingung 2 nicht trivial!

$\textbf{Follow}(A) = \{a, b\}$
Bedingung 2: $\textbf{Follow}(A) \cap \textbf{First}(b) = \{a, b\} \cap \{b\} = \{b\} \neq \emptyset$ nicht erfüllt!

keine LL(1)-Grammatik, nicht zum det. top-down-Parsen geeignet

Berechnung der **First**- und **Follow**-Funktion

Gegeben sei eine kontextfreie Grammatik $G = (N, T, P, S)$.

Wir bestimmen zuerst, aus welchen nichtterminalen Symbolen das leere Wort abgeleitet werden kann, d.h. die Menge $N_\varepsilon = \{A \in N \mid A \stackrel{*}{\Longrightarrow} \varepsilon\}$.

Algorithmus zur Berechnung von N_ε:

1. Bestimme $N_0 = \{A \mid A \to \varepsilon \text{ ist in } P\}$
2. Berechne $N_{i+1} = N_i \cup \{A \mid A \to \alpha \text{ ist in } P \text{ und } \alpha \in N_i^*\}$
3. Ist $N_{i+1} = N_i$, dann setze $N_\varepsilon = N_i$.

Berechnung der **First**- und **Follow**-Funktion

Nun berechnen wir die First-Funktion für alle nichtterminalen Symbole der Grammatik.

Algorithmus zur Berechnung der First-Funktion:
Berechnung von **First**(A) für alle $A \in N$.
Man konstruiert den **Parchmann-Graphen** $\Gamma_{first}(G)$ folgendermaßen:

1. Jedes Symbol aus $N \cup T$ wird durch einen Knoten dargestellt.
2. Für jede Produktion $A \to X_1 \ldots X_n$ mit $n \geq 1$, fügt man eine Kante von A nach X_i hinzu, falls alle davorstehenden Symbole $X_1, \ldots, X_{i-1} \in N_\varepsilon$.
3. Setze **First**$(A) =$
$$\begin{cases} \{a \in T \mid \text{ex. Weg in } \Gamma_{first}(G) \text{ von } A \text{ nach } a\} & \text{, falls } A \notin N_\varepsilon \\ \{\varepsilon\} \cup \{a \in T \mid \text{ex. Weg in } \Gamma_{first}(G) \text{ von } A \text{ nach } a\} & \text{, falls } A \in N_\varepsilon \end{cases}$$

Berechnung der **First**- und **Follow**-Funktion

Damit ist die **First**-Funktion auch für alle Wörter $\alpha \in (N \cup T)^*$ bestimmt.

Wir unterscheiden, ob α das leere Wort ist bzw. mit einem terminalen oder mit einem nichtterminalen Zeichen beginnt:

1. $\textbf{First}(\varepsilon) = \{\varepsilon\}$
2. $\textbf{First}(a\beta) = \{a\}$, falls $a \in T$.
3. $\textbf{First}(A\beta) = \begin{cases} \textbf{First}(A) & \text{, falls } A \in N, A \notin N_\varepsilon \\ (\textbf{First}(A) - \{\varepsilon\}) \cup \textbf{First}(\beta) & \text{, falls } A \in N_\varepsilon \end{cases}$

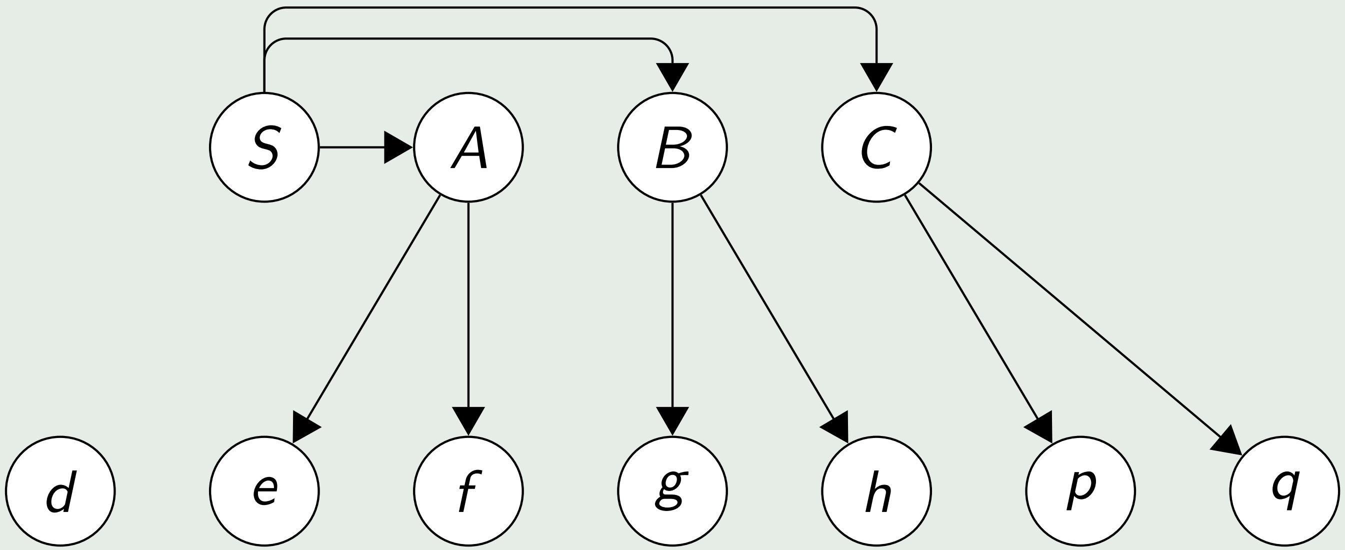
Berechnung der First-Funktion

Beispiel (1)

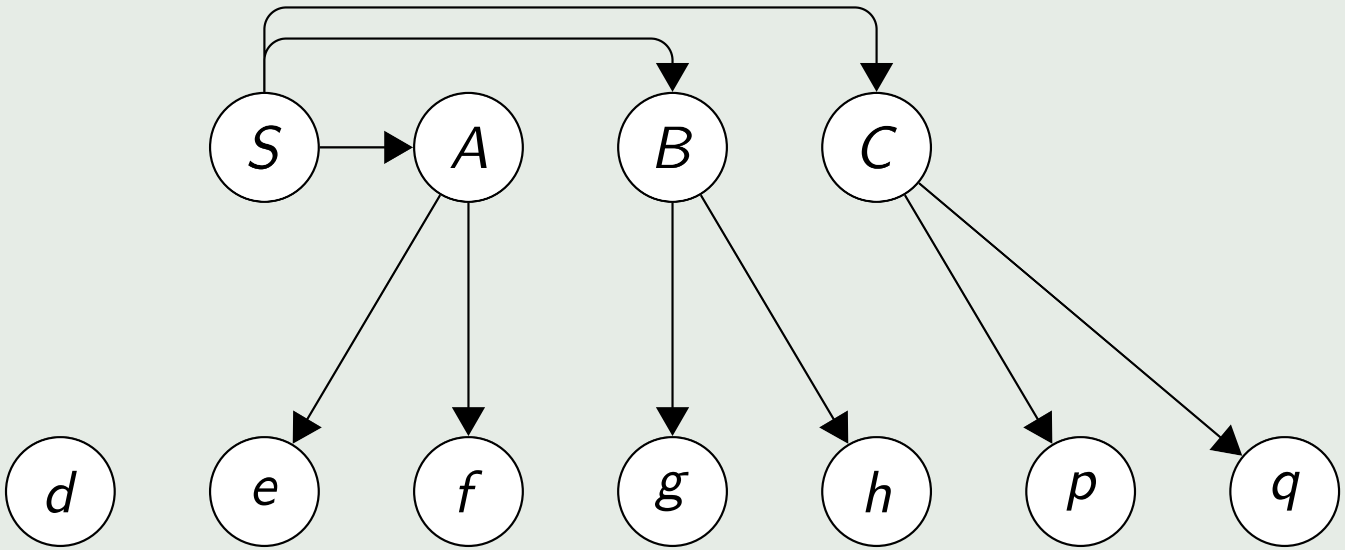
Betrachte die Grammatik $G = (N, T, P, S)$
mit $N = \{S, A, B, C\}$, $T = \{d, e, f, g, h, p, q\}$ und den
Produktionen $S \to ABCd$, $A \to e \mid f \mid \varepsilon$, $B \to g \mid h \mid \varepsilon$ und $C \to p \mid q$.

Es ist $N_\varepsilon = \{A, B\}$ und

der Parchmann-Graph $\Gamma_{first}(G)$ ist:



Beispiel (1)



Also gilt:

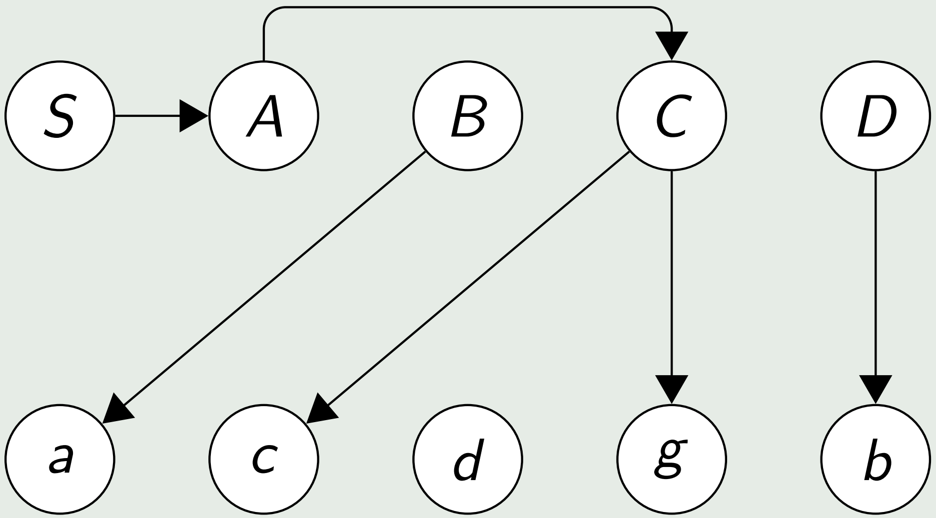
$$\begin{aligned}\textbf{First}(S) &= \{e, f, g, h, p, q\},\\ \textbf{First}(A) &= \{\varepsilon, e, f\},\\ \textbf{First}(B) &= \{\varepsilon, g, h\} \text{ und}\\ \textbf{First}(C) &= \{p, q\}.\end{aligned}$$

Beispiel (2)

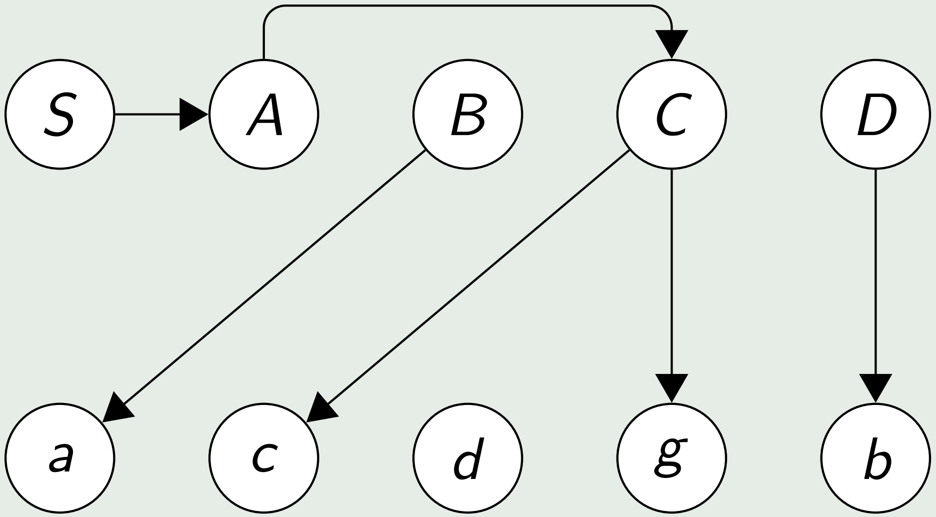
$G = (N, T, P, S)$ mit $N = \{S, A, B, C, D\}$ und $T = \{a, b, c, d, g\}$,
$P = \{S \to AB,\ B \to aAB \mid \varepsilon,\ A \to CD,\ D \to bCD \mid \varepsilon,\ C \to cSd \mid g\}$

Offenbar gilt: $N_\varepsilon = \{B, D\}$

Der Parchmann-Graph $\Gamma_{first}(G)$ ist:



Beispiel (2)



Es gilt also ($N_\varepsilon = \{B, D\}$):

$$\begin{aligned}&\textbf{First}(S) = \textbf{First}(A) = \textbf{First}(C) = \{c, g\},\\ &\textbf{First}(B) = \{\varepsilon, a\}, \text{und}\\ &\textbf{First}(D) = \{\varepsilon, b\}.\end{aligned}$$

und z.B. $\textbf{First}(BAD) = \{a, c, g\}$.

Algorithmus zur Berechnung der Follow-Funktion

Der **Parchmann-Graph** $\Gamma_{follow}(G)$ hat einen Knoten für jedes Symbol in $N \cup T \cup \{\$\}$.

1. Füge eine Kante vom Startsymbol S nach \$ hinzu.
2. Für jedes Vorkommen eines Nonterminals B auf der rechten Seite einer Produktion $A \to \alpha B \beta$ füge Kanten von B nach jedem terminalen Symbol aus **First**(β) hinzu.

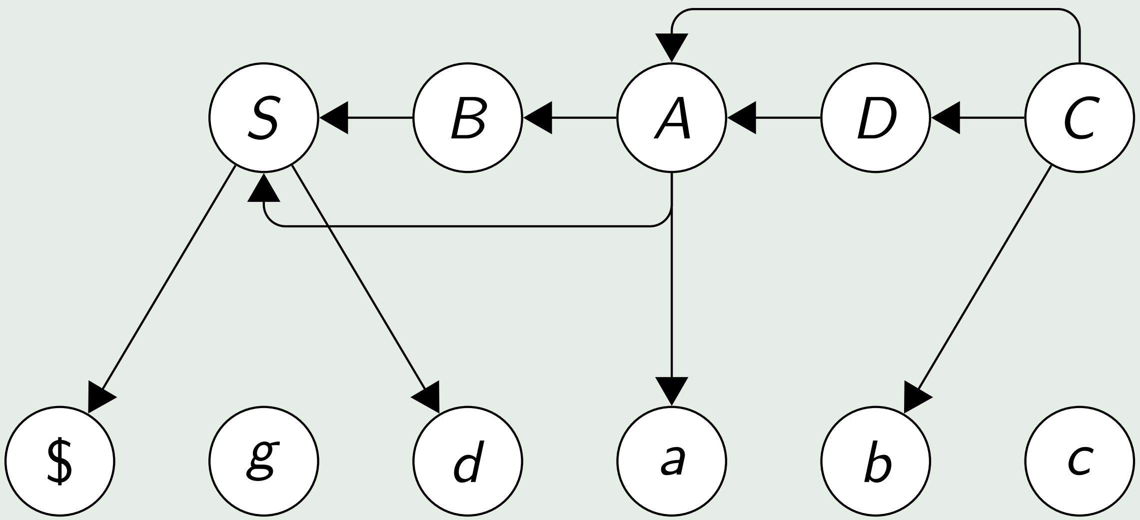
   Ist $\varepsilon \in$ **First**(β) und $A \neq B$, so füge eine Kante von B nach A hinzu.
3. **Follow**$(A) := \{a \in T \cup \{\$\} \mid$ ex. Weg von A nach a in $\Gamma_{follow}(G)\}$.

Berechnung der Follow-Funktion

Beispiel (2)

$P = \{S \to AB,\ B \to aAB \mid \varepsilon,\ A \to CD,\ D \to bCD \mid \varepsilon,\ C \to cSd \mid g\}$

Der Parchmann-Graph $\Gamma_{follow}(G)$ ist:

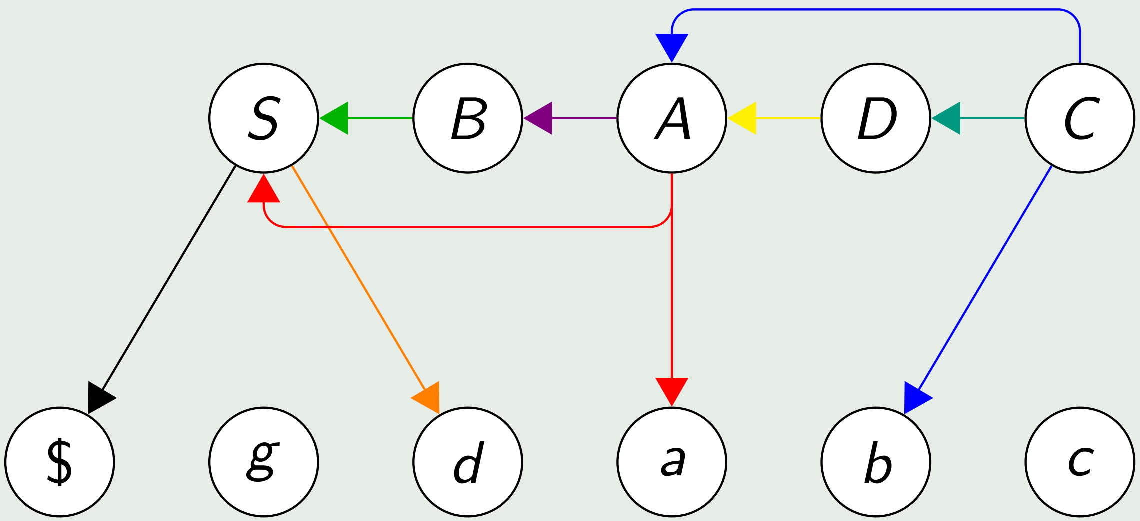


Beispiel (2)

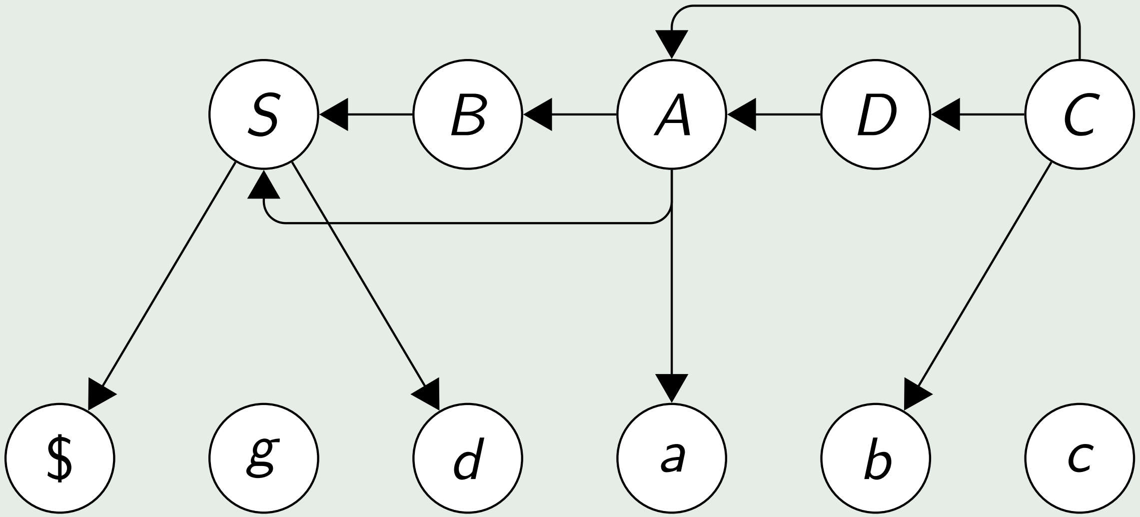
$$P = \{S \to AB,\ B \to aAB \mid \varepsilon,\ A \to CD,\ D \to bCD \mid \varepsilon,\ C \to cSd \mid g\}$$

$$\begin{aligned}&\textbf{First}(S) = \textbf{First}(A) = \textbf{First}(C) = \{c, g\},\\ &\textbf{First}(B) = \{\varepsilon, a\} \text{ und}\\ &\textbf{First}(D) = \{\varepsilon, b\}.\end{aligned}$$

Der Parchmann-Graph $\Gamma_{follow}(G)$ ist:



Beispiel (2)



$\textbf{Follow}(S) = \textbf{Follow}(B) = \{d, \$\}$,
$\textbf{Follow}(A) = \textbf{Follow}(D) = \{a, d, \$\}$ und
$\textbf{Follow}(C) = \{a, b, d, \$\}$.

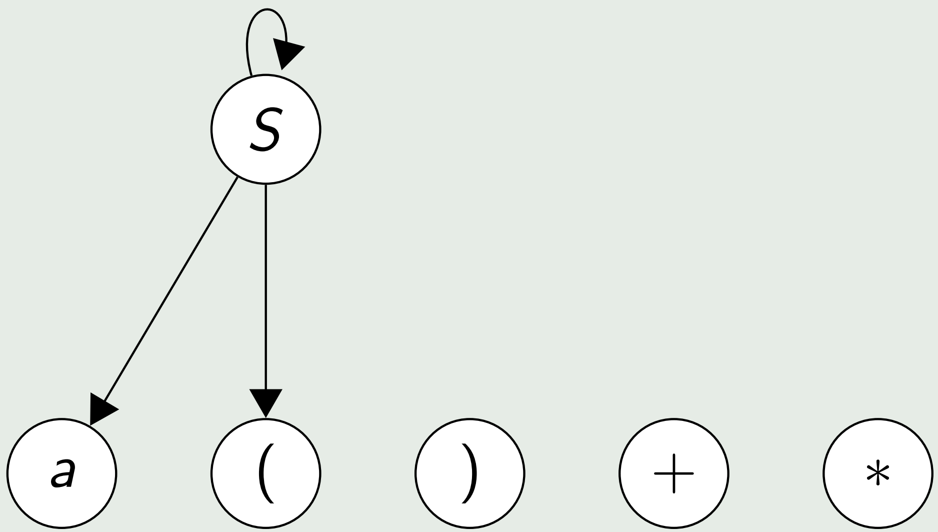
Überprüfung einer Grammatik

Beispiel (Grammatik für arithmetische Ausdrücke)

Produktionen: $S \to S + S \mid S * S \mid (S) \mid a$

Berechnung der **First**-Mengen:

$N_\varepsilon = \emptyset$



First$(S + S) = \{(, a\}$,
First$(S * S) = \{(, a\}$,
First$((S)) = \{(\}$ und
First$(a) = \{a\}$.

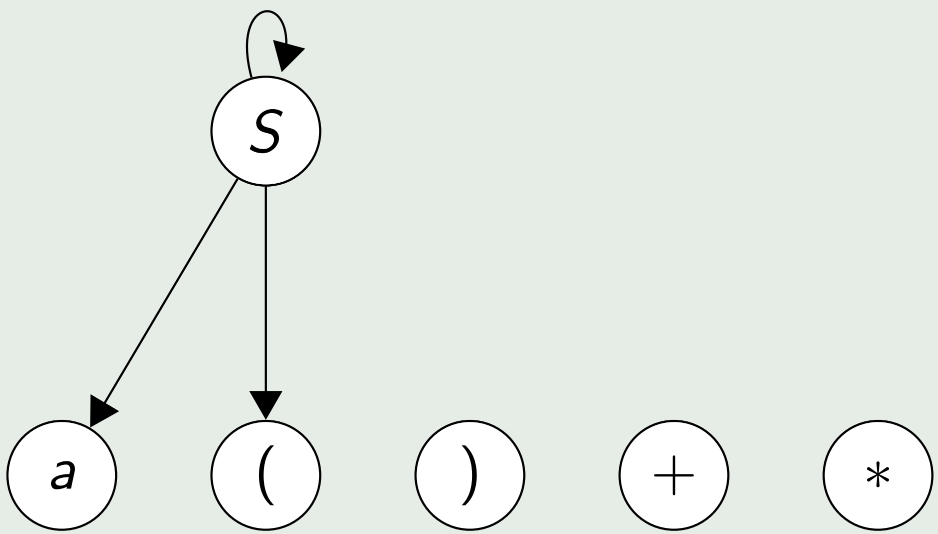
First-Mengen nicht disjunkt!
Keine LL(1)-Grammatik!

Beispiel (Grammatik für arithmetische Ausdrücke)

Produktionen: $S \to S + S \mid S * S \mid (S) \mid a$

Berechnung der **First**-Mengen:

$N_\varepsilon = \emptyset$



First$(S + S) = \{(, a\}$,
First$(S * S) = \{(, a\}$,
First$((S)) = \{(\}$ und
First$(a) = \{a\}$.

First-Mengen nicht disjunkt!
Keine LL(1)-Grammatik!

Nicht überraschend, da Produktionen linksrekursiv!

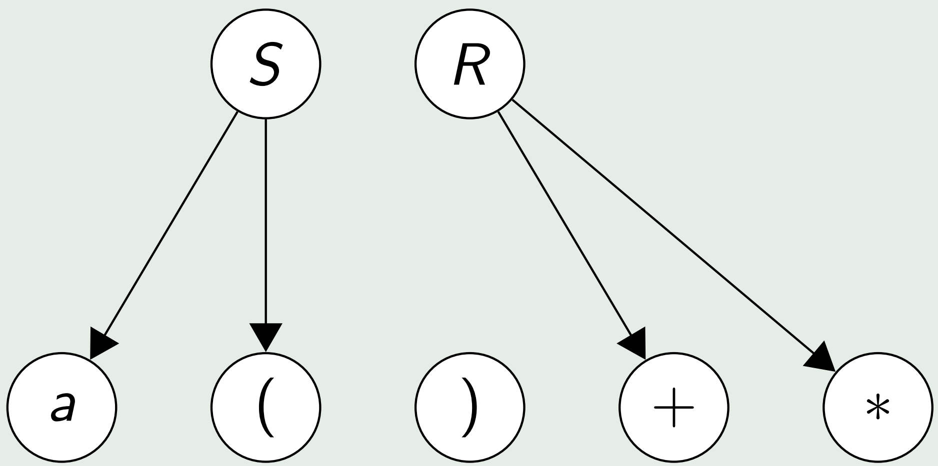
Beispiel (Grammatik nach Entfernung der Linksrekursion)

Produktionen: $S \to (S)R \mid aR$

$R \to +SR \mid *SR \mid \varepsilon$

Berechnung der **First**-Mengen:

$N_\varepsilon = \{R\}$



First$(S) = \{a, (\}$,
First$(R) = \{\varepsilon, +, *\}$,

First$((S)R) = \{(\}$,
First$(aR) = \{a\}$. disjunkt!

First$(+SR) = \{+\}$,
First$(*SR) = \{*\}$ und
First$(\varepsilon) = \{\varepsilon\}$. disjunkt!

Noch zu prüfen: **Follow**$(R) \cap$ **First**$(+SR) = \emptyset$

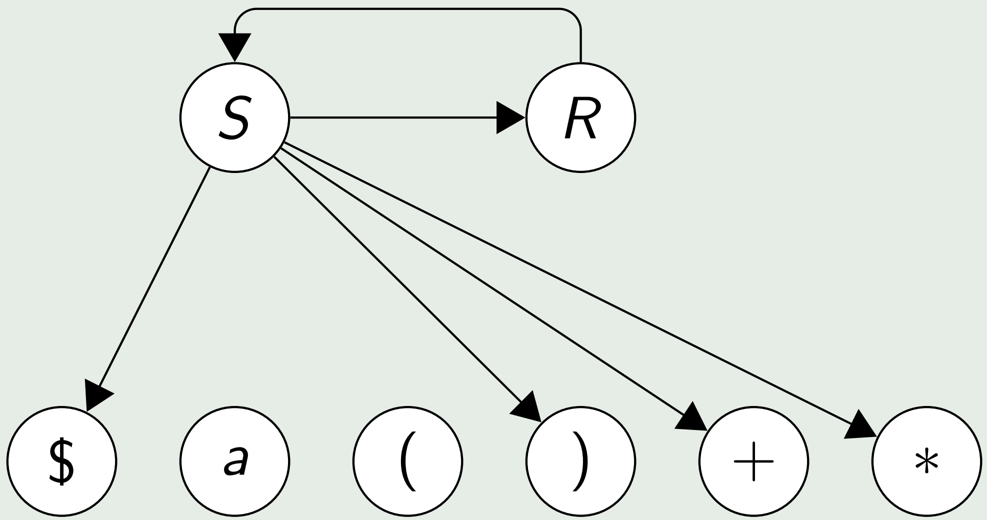
und **Follow**$(R) \cap$ **First**$(*SR) = \emptyset$

Beispiel (Grammatik nach Entfernung der Linksrekursion)

Produktionen: $S \to (S)R \mid aR$

$R \to +SR \mid *SR \mid \varepsilon$

Berechnung der **Follow**-Mengen:



$\textbf{Follow}(S) = \{\$,), +, *\}$,
$\textbf{Follow}(R) = \{\$,), +, *\}$

$\textbf{Follow}(R) \cap \textbf{First}(+SR) =$
$\{\$,), +, *\} \cap \{+\} \neq \emptyset$

$\textbf{Follow}(R) \cap \textbf{First}(*SR) =$
$\{\$,), +, *\} \cap \{*\} \neq \emptyset$

Beispiel für $+ \in \textbf{Follow}(R)$:
$S \Rightarrow aR \Rightarrow a{+}SR \Rightarrow a{+}aRR \Rightarrow$
$a{+}aR{+}SR$

Keine LL(1)-Grammatik!

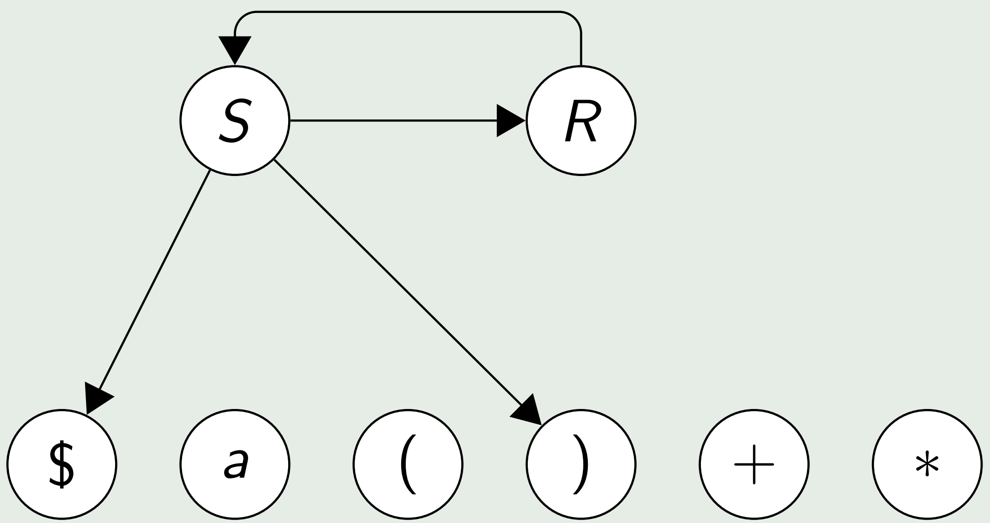
Beispiel (modifizierte Grammatik)

Um $+$ und * aus **Follow**(R) zu entfernen, ändern wir die Grammatik (aber nicht die Sprache!):

Produktionen: $S \to (S)R \mid aR$

$$R \to +S\cancel{R} \mid *S\cancel{R} \mid \varepsilon$$

Berechnung der **Follow**-Mengen:



Follow$(S) = \{\$,)\}$,
Follow$(R) = \{\$,)\}$

Follow$(R) \cap$ **First**$(+S) =$
$\{\$,)\} \cap \{+\} = \emptyset$

Follow$(R) \cap$ **First**$(*S) =$
$\{\$,)\} \cap \{*\} = \emptyset$

LL(1)-Grammatik!