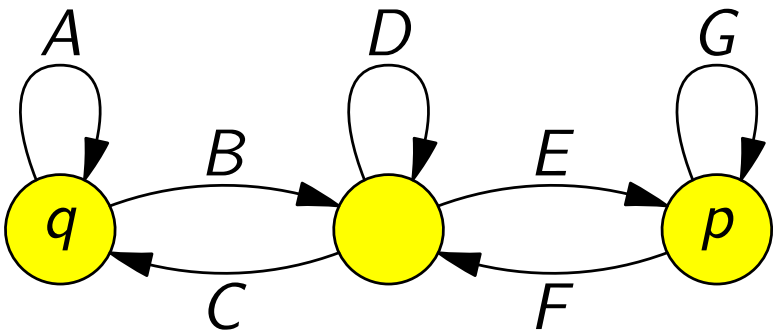
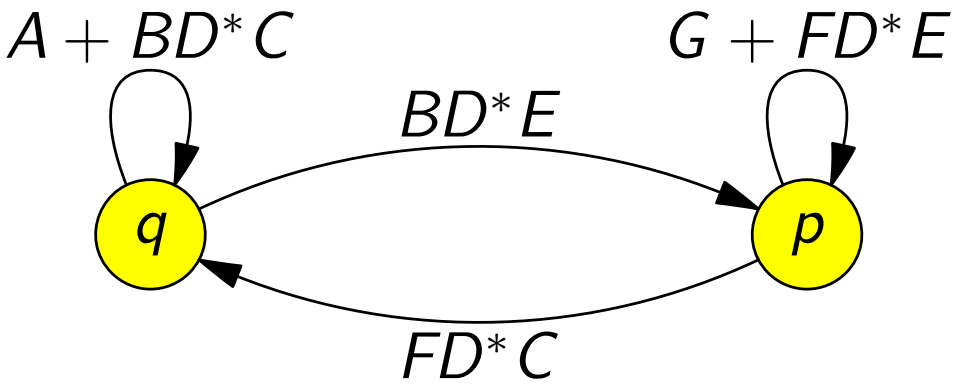
Eliminierung eines Zustandes



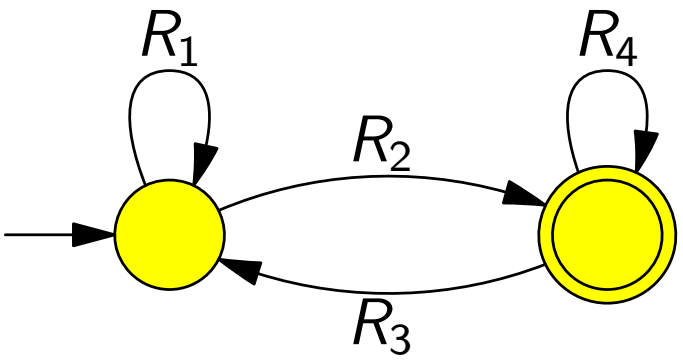


Für *alle* Paare (p, q)!

Eliminierung von Zuständen

Allgemeines Vorgehen:

1. Starte mit einem NFA mit nur einem Endzustand
2. Eliminiere Zustände bis auf den Start- und Endzustand
3. Lies den regulären Ausdruck ab



Gesuchter Ausdruck ist:

$$R_1^* R_2 (R_4 + R_3 R_1^* R_2)^*$$

Theorem (Pumping-Lemma)

Sei L eine reguläre Sprache. Dann gibt $n \in \mathbf{N}$, so daß jedes Wort $w \in L$ mit $|w| \geq n$ in Wörter $w = xyz$ zerlegt werden kann mit

1. $|xy| \leq n$
2. $|y| > 0$
3. $xy^i z \in L$ *für alle* $i \geq 0$

Beweis.

Es sei $L = L(M)$ für einen DFA M mit n Zuständen.
Ist $|w| \geq n$ dann durchläuft M einen Zustand doppelt.
p sei der erste solche Zustand.
Es gibt also $xyz = w$, mit

- $\hat{\delta}(q_0, x) = p$,
- $\hat{\delta}(p, y) = p$, $|xy| \leq n$, $|y| > 0$,
- $xy^i z \in L$

Das Pumping-Lemma als Spiel

Gegeben sei eine Sprache $L \subseteq \Sigma^*$.

1. Alice wählt eine Zahl n.
2. Bob wählt ein Wort $w \in L$ mit $|w| \geq n$.
3. Alice wählt $x, y, z \in \Sigma^*$ mit $xyz = w$, $|xy| \leq n$, $|y| > 0$.
4. Bob wählt eine Zahl i.

Alice gewinnt, wenn $xy^iz \in L$.
Bob gewinnt, wenn $xy^iz \notin L$.

Falls L regulär ist, kann Alice immer gewinnen.

Gewinnstrategie für Bob $\Longrightarrow$ L nicht regulär!

Beispiel

Es sei $L = \{ a^n b^n \mid n \geq 0 \}$.

1. Alice wählt eine Zahl n.
2. Bob wählt $w = a^n b^n$.
3. Alice wählt $x, y, z \in \{a, b\}^*$ mit $xyz = w$, $|xy| \leq n$, $|y| > 0$.
4. Bob wählt $i = 2$.

Bob hat gewonnen:

$xyyz$ enthält eine ungleiche Anzahl von a und b.

Also ist L (leider) nicht regulär.

Beispiel

Es sei $L = \{ a^{n^3} \mid n \geq 0 \}$.

1. Alice wählt eine Zahl n.
2. Bob wählt $w = a^{m^3}$ mit $m = n + 30$.
3. Alice wählt $x, y, z \in a^*$ mit $xyz = w$, $|xy| \leq n$, $|y| > 0$.
4. Bob denkt: $x = a^r$, $y = a^s$, $z = a^t$ mit $r + s + t = m^3$.
   Bob wählt $i = 0$ und denkt weiter:

$$(m-1)^3 = m^3 - 3m^2 + 3m - 1 < m^3 - m \leq |xy^0z| = r + t < m^3$$

Das Leerheitsproblem für reguläre Sprachen

Theorem

Das Leerheitsproblem für reguläre Sprachen ist in polynomieller Zeit lösbar:

Eingabe: Ein DFA, NFA oder ein regulärer Ausdruck für Sprache L

Frage: Ist $L = \emptyset$?

Beweis.

DFA, NFA: Ist ein Endzustand vom Anfangszustand erreichbar? Lineare Laufzeit mit Tiefensuche.

Regulärer Ausdruck: In NFA verwandeln. □

Das Wortproblem für reguläre Sprachen

Theorem

Das Wortproblem für reguläre Sprachen ist in quadratischer (DFA: linearer) Zeit lösbar:

Eingabe: Ein DFA, NFA oder ein regulärer Ausdruck für Sprache L

Frage: Ist $w \in L$?

Beweis.

DFA: Algorithmus aus Vorlesung
Laufzeit: $O(|M| + |w|)$
NFA: Algorithmus aus Vorlesung
Laufzeit: $O(|M| + |Q|^2 \cdot |\Sigma| + |Q|^2 \cdot |w|)$

Regulärer Ausdruck: In NFA verwandeln. □

Das Universalitätsproblem für reguläre Sprachen

Theorem

Das Universalitätsproblem für reguläre Sprachen ist algorithmisch lösbar:

Eingabe: DFA, NFA oder regulärer Ausdruck für Sprache $L \subseteq \Sigma^*$

Frage: Ist $L = \Sigma^*$?

Beweis.

DFA: Sind alle erreichbaren Zustände Endzustände?
Lineare Laufzeit mit Tiefensuche.

NFA, regulärer Ausdruck: In DFA verwandeln.
Exponentielle Laufzeit. □

Das Endlichkeitsproblem für reguläre Sprachen

Theorem

Das Endlichkeitsproblem für reguläre Sprachen ist in linearer Zeit lösbar:

Eingabe: DFA, NFA oder regulärer Ausdruck für Sprache $L \subseteq \Sigma^*$

Frage: Ist L endlich?

Beweis.

NFA, DFA: Gibt es

1. eine starke Zusammenhangskomponente, die vom Startzustand erreichbar ist, und
2. von der aus ein Endzustand erreichbar ist?

Lineare Laufzeit (Konstruktion der starken Zusammenhangskomponenten: Datenstrukturen und Algorithmen)

□

Das Äquivalenzproblem für reguläre Sprachen

Theorem

Das Äquivalenzproblem für reguläre Sprachen ist algorithmisch lösbar:

Eingabe: Zwei DFAs, NFAs oder reguläre Ausdrücke für Sprachen L_1 und L_2

Frage: Ist $L_1 = L_2$?

Beweis.

DFA: Minimiere beide und teste ob sie dann isomorph sind.

NFA, Regulärer Ausdruck: In DFA verwandeln. □

Kontextfreie Grammatik

Definition

Eine *kontextfreie Grammatik* ist ein 4-Tupel (N, T, P, S)

- N ein Alphabet der *Nonterminale* oder *Variablen*
- T ein Alphabet der *Terminalsymbole*
- P Menge von *Produktionen* der Form $A \to \alpha$ mit $A \in N$ und $\alpha \in (N \cup T)^*$
- $S \in N$ das *Startsymbol*

Wir verlangen $N \cap T = \emptyset$.

Satzformen sind die Wörter aus $(N \cup T)^*$.

Notation:

Wir verwenden oft

- $a, b, c, \ldots$ für Terminalsymbole
- $A, B, C, \ldots$ für Nonterminale
- $u, v, w, \ldots$ für Terminalwörter aus T^*
- $\alpha, \beta, \gamma, \ldots$ für Satzformen

Beispiel

Eine kontextfreie Grammatik:

$$G = (\{E, A\}, \{0, 1, +, -, (,)\}, P, E)$$

mit

$$P = \{E \to (E+E), E \to (E-E), E \to A, A \to 0, A \to 1, A \to AA\}$$

Ableitungen

Wir definieren die Relation

$$\Rightarrow_G : (N \cup T)^* \to (N \cup T)^*$$

vermöge

$$\alpha A \beta \Rightarrow \alpha \gamma \beta \text{ falls } A \to \gamma \in P.$$

$\Rightarrow_G$ heißt Ableitungsrelation.

$\overset{*}{\Rightarrow}_G$ ist die reflexiv-transitive Hülle von $\Rightarrow_G$

Beispiel

$$G = (\{E, A\}, \{0, 1, +, -, (,)\}, P, E)$$

mit
$$P = \{E \to (E+E), E \to (E-E), E \to A, A \to 0, A \to 1, A \to AA\}$$

$$E \Rightarrow (E + E) \Rightarrow (E + A) \Rightarrow (A + A) \Rightarrow (AA + A) \Rightarrow (A0 + A) \Rightarrow (A0 + 1) \Rightarrow (10 + 1) \in T^*$$

Wir geben in Zukunft oft nur die Produktionen an.

(Wenn klar ist was N, T und das Startsymbol sind.)

Sprache einer Grammatik

Definition

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

Wir definieren $L(G) = \{ w \in T^* \mid S \overset{*}{\Rightarrow} w \}$.

$L(G)$ ist die von G *erzeugte Sprache*.

Wir nennen $\alpha \in \beta$ einen *Ableitungsschritt* und
$S \Rightarrow \alpha_1 \Rightarrow \cdots \Rightarrow \alpha_{n-1} \Rightarrow w$ eine *Ableitung* von w.

Eine Ableitung ist eine Linksableitung, falls in jedem Schritt das am weitesten links stehende Nonterminal ersetzt wird. (Rechtsableitung analog).

Beispiel

CFG G mit $S \to aSSb$, $S \to \epsilon$.

$S \Rightarrow aSSb \Rightarrow aaSSbSb \Rightarrow aaSbSb \Rightarrow aabSb \Rightarrow aabaSSbb \Rightarrow aabaSbb \Rightarrow aababb$

ist eine Linksableitung von $aababb$.

Ableitungsbäume

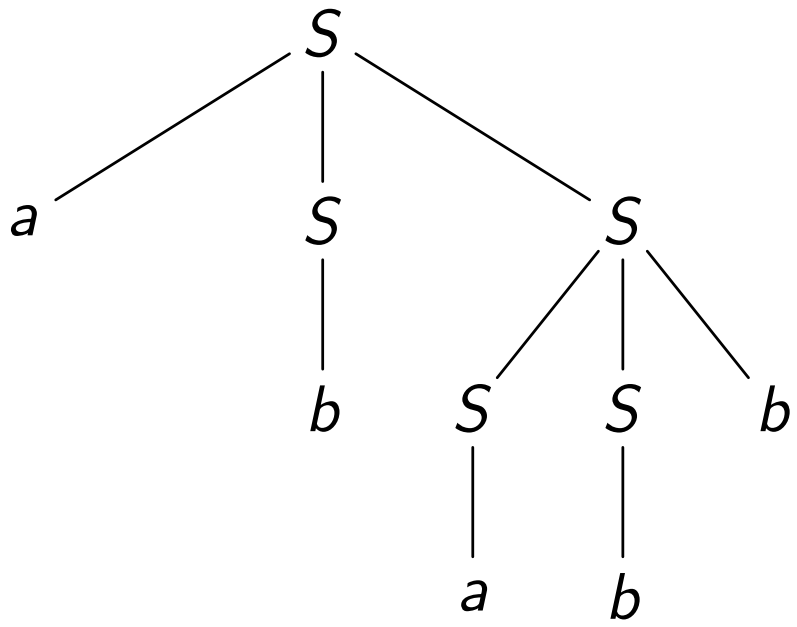
Definition

Ein Ableitungsbaum ist ein Baum:

1. er hat eine Wurzel
2. er ist orientiert
3. innere Knoten sind Nonterminale
4. Blätter sind Terminalsymbole
5. Kinder eines inneren Knotens A sind rechte Seite einer Produktion $A \to \alpha$

Beispiel

$S \to aSS \mid SSb \mid a \mid b$



Zugehörige Linksableitung:

$S \Rightarrow aSS \Rightarrow abS \Rightarrow abSSb \Rightarrow abaSb \Rightarrow ababb$

Theorem

Es sei $G = (N, T, P, S)$ eine CFG und $w \in T^*$.
Dann gilt $w \in L(G)$ genau dann, wenn es einen Ableitungsbaum gibt

- *mit Wurzel S,*
- *mit Blättern w.*

Beweis.

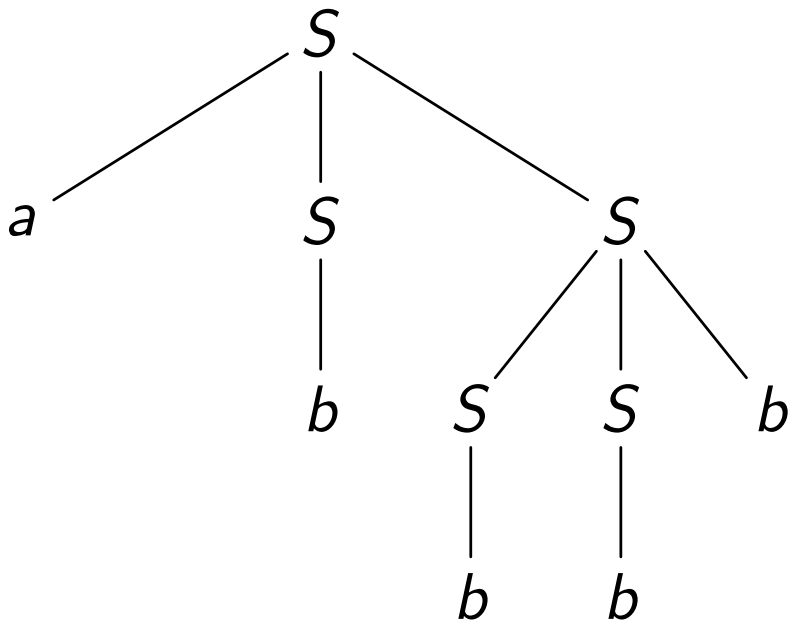
(Idee) Allgemeinere Aussage:
$A \overset{*}{\Rightarrow} \alpha$ genau dann, wenn Ableitungsbaum mit Wurzel A und Blättern α.

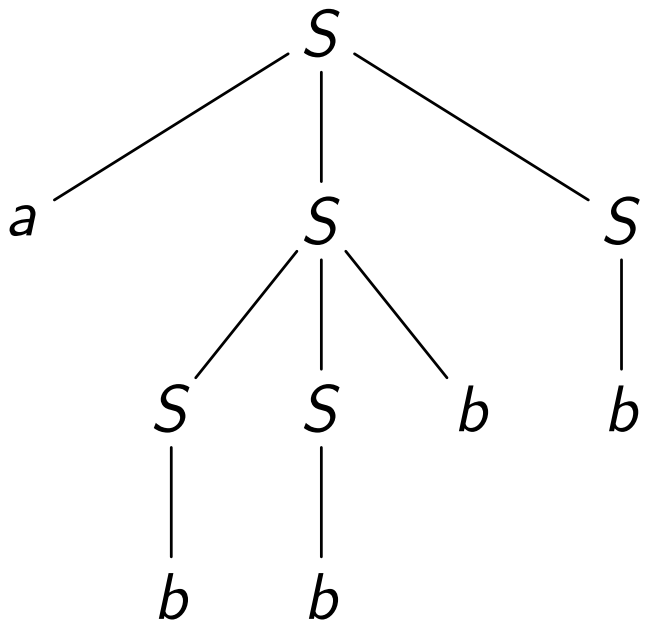
Beweis mit Induktion über Länge einer Ableitung. □

Zu jedem Ableitungsbaum gibt es eine *eindeutige* Linksableitung.

Eindeutigkeit

$S \to aSS \mid SSb \mid a \mid b$



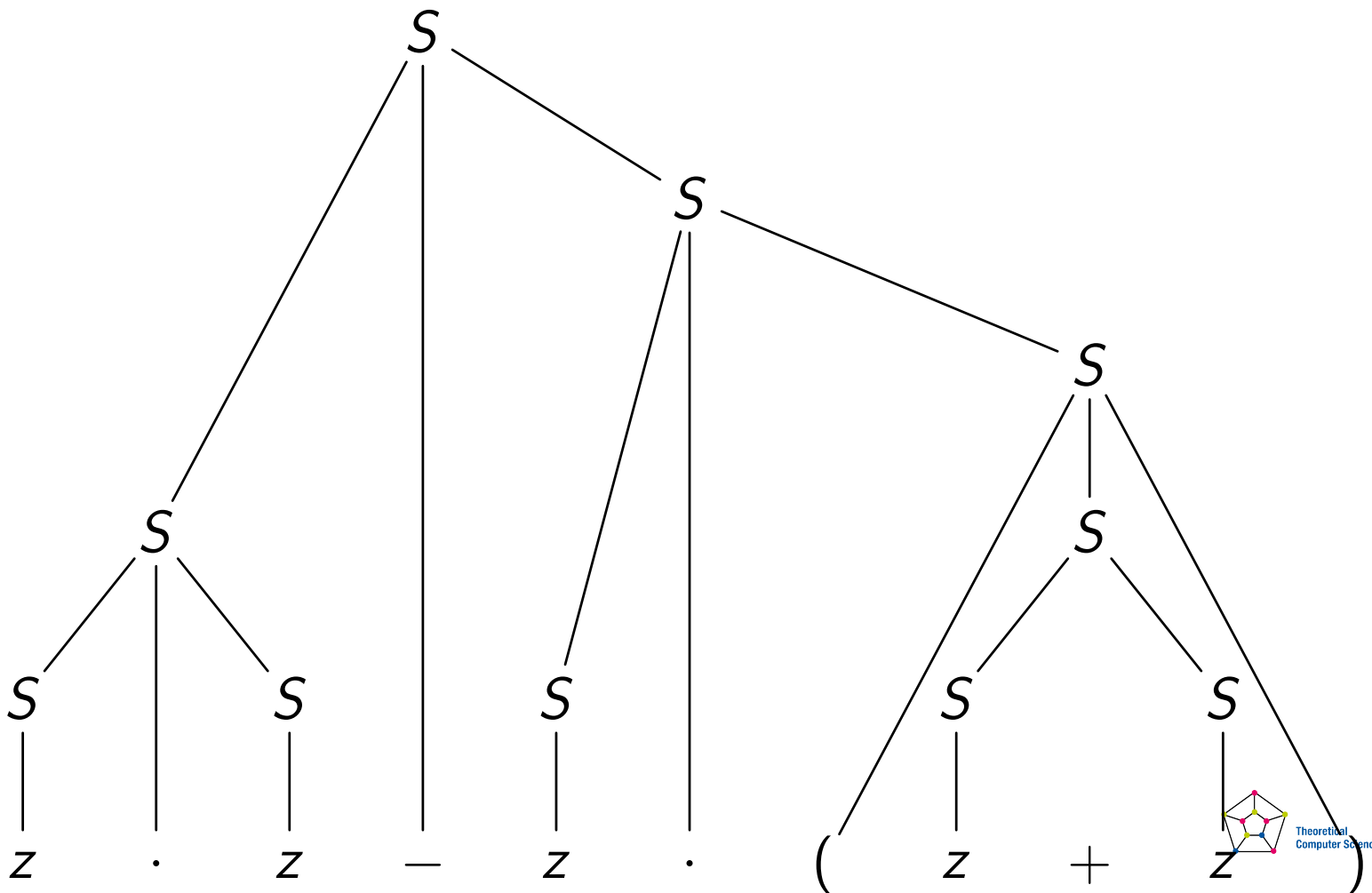


Es gibt zwei verschiedene Linksableitungen.

Beispiel 1

$S \to S + S \mid S - S \mid S \cdot S \mid S/S \mid (S) \mid \mathbf{z}$

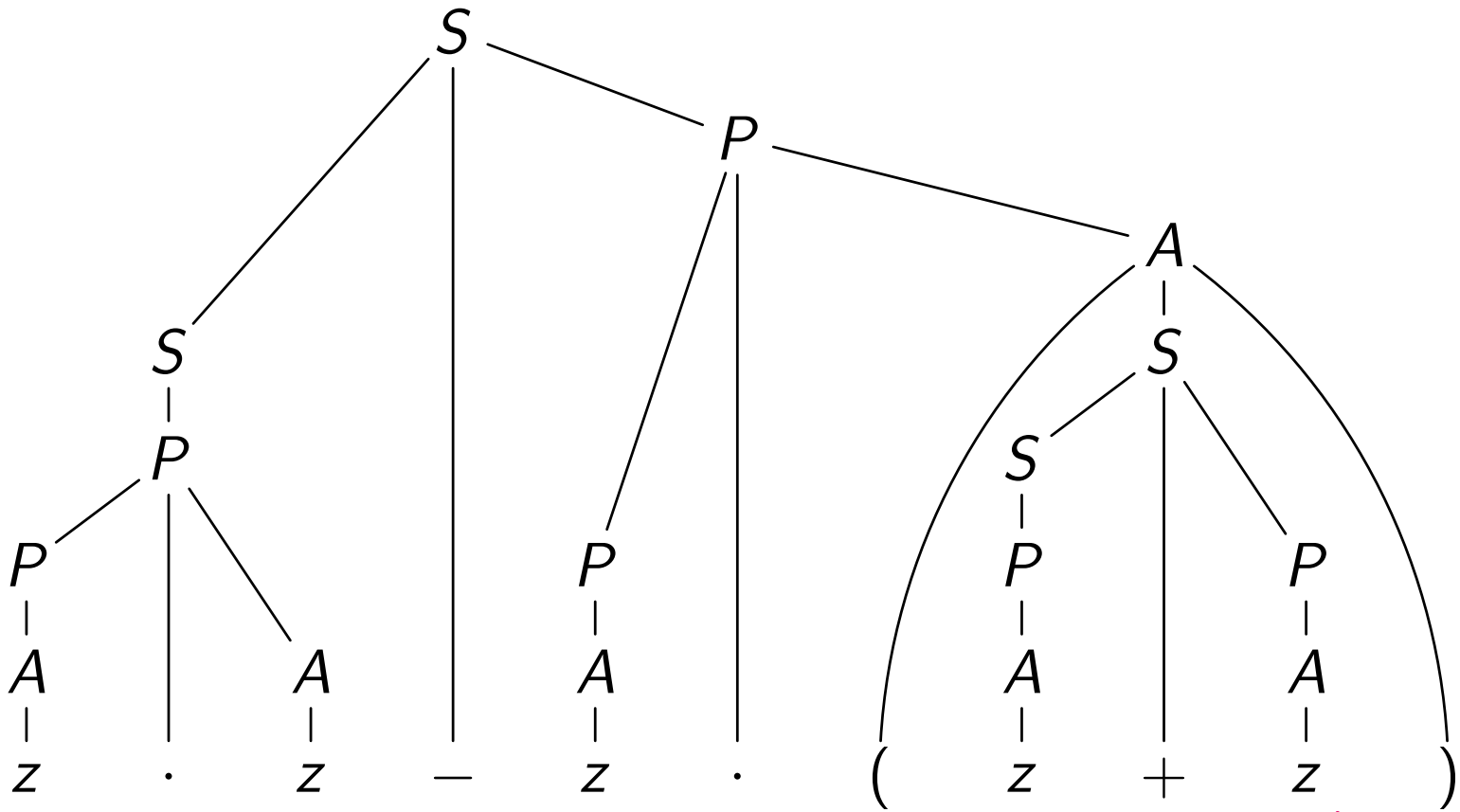
Betrachte: $\mathbf{z} \cdot \mathbf{z} - \mathbf{z} \cdot (\mathbf{z} + \mathbf{z})$



Beispiel 2

$S \to S + P \mid S - P \mid P$
$P \to P \cdot A \mid P/A \mid A$
$A \to (S) \mid \mathbf{z}$

Betrachte: $\mathbf{z} \cdot \mathbf{z} - \mathbf{z} \cdot (\mathbf{z} + \mathbf{z})$



Welchen Vorteil hat diese Grammatik?

Beispiel 3

$S \to P + S \mid P - S \mid P$
$P \to A \cdot P \mid A / P \mid A$
$A \to (S) \mid \mathbf{z}$

Betrachte: $\mathbf{z} \cdot \mathbf{z} - \mathbf{z} \cdot (\mathbf{z} + \mathbf{z})$

Welchen Vorteil hat diese Grammatik?

```c
#include ⟨stdlib.h⟩
#include ⟨stdio.h⟩

char * s;
void A(), P(), Z();

void S() {
  P();
  if(*s == '+') s++, S();
  if(*s == '-') s++, S();
}

void P() {
  A();
  if(*s == '*') s++, P();
  if(*s == '/') s++, P();
}
```

```c
void A() {
  if(*s == '(') {
    ++s;
    S();
    if(*s == ')')++s;
    else exit(1);
  }
  else Z();
}

int main(int argc, char * argv[])
{
  if(argc ≠ 2) exit(1);
  s = argv[1];
  S();
  if(*s ≠ 0) exit(1);
  printf("okay!\n");
  return 0;
}
```