Formale Grundlagen der Informatik

10

Backus-Naur-Form
Kontextfreie Grammatiken
Compiler • Parsing

Recap: Hierarchiebeziehungen

$$\mathcal{L}(\text{REG}) \subset \mathcal{L}(\text{REC}) \subset \mathcal{L}(\text{RE})$$

echt wegen $\{\, a^n b^n \mid n \geq 0 \,\}$

echt wegen L_u

Die Sprache $\{\, a^n b^n \mid n \geq 0 \,\}$

- ist relevant für die Beschreibung syntaktischer korrekter Eigenschaften, wie sie vom Compiler geprüft werden
- ist nicht regulär $\longrightarrow$ kann nicht mit regulären Ausdrücken spezifiziert werden ☹
- kann von DTM akzeptiert werden, aber das Wortproblem und das Äquivalenzproblem sind für DTMs unentscheidbar ☹

➢ Benötigen Mechanismen, die stärker sind als reguläre Ausdrücke und schwächer als DTMs!

Syntax-Definition von HTML5

```
tag-open := '<' tag-name ws* attr-list? ws* '>'
tag-empty := '<' tag-name ws* attr-list? ws* '/>'
tag-close := '</' tag-name ws* '>'

attr-list := (ws+ attr)*
attr := attr-empty | attr-unquoted | attr-single-quoted | attr-double-quoted

attr-empty := attr-name
attr-unquoted := attr-name ws* = ws* attr-unquoted-value
attr-single-quoted := attr-name ws* = ws* ' attr-single-quoted-value '
attr-double-quoted := attr-name ws* = ws* " attr-double-quoted-value "

tag-name := (alphabets | digits)+
attr-name := /[^\s"'>/=\p{Control}]+/
attr-unquoted-value := /[^\s"'=<>`]+/
attr-single-quoted-value := /[^']*/
attr-double-quoted-value := /[^"]*/
```

```
alphabets := /[a-zA-Z]/
digits := /[0-9]/
ws := /\s/
```

Definition von syntaktischen Kategorien in (erweiterter) Backus-Naur- Form

Backus-Naur-Form (BNF)

- entwickelt von John Backus und Peter Naur
- Definition der Syntax durch einen Satz von Ersetzungsregeln der Form

  Kategorie := Ausdruck

  - wobei Kategorie durch einen Namen angegeben wird und
  - Ausdruck eine durch | getrennte Folge von Wörtern ist,
    wobei jedes Wort aus Buchstaben, wie sie im Programm vorkommen,
    und Kategorien besteht

    attr := attr-empty | attr-unquoted | attr-single-quoted | attr-double-quoted

    tag-open := '<' tag-name ws* attr-list? ws* '>'

  - In <u>erweiterter</u> BNF dürfen die Wörter reguläre Ausdrücke sein.
    (*betrachten wir hier nicht weiter*)

Backus-Naur-Form (BNF)

- Beispiel aus der Definition von Programmiersprachen:

  statement := assignment | loop | branch

  assignment := identifier '=' arithmetic-expression

  arithmetic-expression := ...

- Die Ersetzungsregel für statement kann in drei Regeln aufgeteilt werden.
- Dann hat jede Regel eine linke Seite (Kategorie) und eine rechte Seite, durch die die Kategorie ersetzt („verfeinert“) werden kann.
  - Sie sind also Paare der Form (linke-Seite, rechte-Seite).

Kontextfreie Grammatiken

- sind eine Abstraktion von BNF
- Komponenten:
  - Alphabet der nichtterminalen Symbole („Kategorien“)
    - müssen weiter ersetzt werden, damit ein syntaktisch korrektes Programm entsteht
  - Alphabet der terminalen Symbole (wie sie im Programmen vorkommen)
    - müssen also nicht weiter ersetzt werden
  - endliche Menge von Ersetzungsregeln
    - beschreiben, wie ein Nichtterminal ersetzt werden kann
  - ein Startsymbol aus der Menge der Nichtterminale
    - die („gröbste“/allgemeinste) syntaktische Kategorie, die als erstes zu ersetzen ist
    - *z.B.* Programm *bzw.* HTML-Dokument

Kontextfreie Grammatiken - Definition

Definition: Eine **kontextfreie Grammatik (kfG)** ist ein Quadrupel $G = (N, T, P, S)$, wobei

- N und T Alphabete sind, $N \cap T = \emptyset$,
  deren Symbole **Nichtterminale** bzw. **Terminale** heißen;
- $P \subseteq N \times (N \cup T)^*$ eine endliche Menge von **Ersetzungsregeln** (Regeln/Produktionen) ist,
  deren Elemente in der Form $A \rightarrow \alpha$ (statt (A, α)) geschrieben werden;
- $S \in N$ das **Startsymbol** (oder Axiom) ist.

Kontextfreie Regeln

- *Beispiel*: assignment := identifier '=' arithmetic-expression

$$\begin{array}{ccccc} \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ A & \rightarrow & a & b & B \end{array}$$

mit $A, B \in N$ und $a, b \in T$

- *Konvention*:
  - Nichtterminale durch Großbuchstaben angeben,
  - Terminale durch Kleinbuchstaben,
  - Wörter, in denen Terminale und Nichtterminale vorkommen, durch griechische Buchstaben

Kontextfreie Grammatik - Beispiel

- einfache arithmetische Ausdrücke mit den Operatoren + und *:
  - Nichtterminal E für „expression“
  - Terminale $+$ und $*$
  - Terminal **id** für „identifier“ als elementarer Ausdruck

- $G = (\{E\}, \{+, *, \textbf{id}\}, P, E)$, wobei P aus folgenden Regeln besteht:

$$E \to E + E \qquad\qquad E \to E * E$$

$$E \to (E) \qquad\qquad E \to \textbf{id}$$

oder: $E \to E + E \mid E * E \mid (E) \mid \textbf{id}$

Ableitungen

- Sei $G = (N, T, P, S)$ eine kontextfreie Grammatik.
- Ein direkter Ableitungsschritt in G ist wie folgt definiert:
  Für Wörter $\alpha, \beta \in (N \cup T)^*$ gilt $\boldsymbol{\alpha \underset{G}{\Longrightarrow} \beta}$ falls
  1. $\alpha = \gamma_1 A \gamma_2$
  2. $\beta = \gamma_1 \nu \gamma_2$
  3. $A \to \nu \in P$

Der Index G kann weggelassen werden, wenn aus dem Kontext eindeutig klar.

- Es gilt $\alpha \overset{m}{\underset{G}{\Longrightarrow}} \beta$ falls Wörter $\alpha_0, \alpha_1, \dots, \alpha_m$ existieren, so dass

$$\alpha = \alpha_0 \underset{G}{\Longrightarrow} \alpha_1 \underset{G}{\Longrightarrow} \cdots \underset{G}{\Longrightarrow} \alpha_m = \beta.$$

- Es gilt $\alpha \overset{*}{\underset{G}{\Longrightarrow}} \beta$ falls $\alpha \overset{m}{\underset{G}{\Longrightarrow}} \beta$ für ein $m \geq 0$.
  Wir sagen dann, dass β aus α abgeleitet wird.

$\overset{*}{\Longrightarrow}$ ist die reflexive und transitive Hülle von $\Longrightarrow$.

Ableitungen – Beispiele

$E \to E + E \mid E * E \mid (E) \mid \textbf{id}$

- $E \Longrightarrow E + E$
- $E \Longrightarrow E + E \Longrightarrow E + E * E$
- $E \Longrightarrow \textbf{id}$
- $E \Longrightarrow E * E \Longrightarrow (E) * E \Longrightarrow (E + E) * E$
  $\Longrightarrow (\textbf{id} + E) * E \Longrightarrow (\textbf{id} + \textbf{id}) * E \Longrightarrow (\textbf{id} + \textbf{id}) * \textbf{id}$

also

$E \overset{*}{\Longrightarrow} E + E$

$E \overset{*}{\Longrightarrow} E + E * E$

$E \overset{*}{\Longrightarrow} \textbf{id}$

$E \overset{*}{\Longrightarrow} (\textbf{id} + \textbf{id}) * \textbf{id}$

terminale Wörter

Sprache einer kontextfreien Grammatik

- Falls $S \xRightarrow{*} \alpha$ für ein $\alpha \in (N \cup T)^*$, dann heißt α **Satzform** von G.
- Terminale Satzformen
  - erlauben keine weiteren Ableitungsschritte
  - enthalten nur noch Symbole, die in Programmen (als Token) so vorkommen
  - ➢ sind Elemente der durch die Grammatik definierten Sprache

- **Definition:** Sei $G = (N, T, P, S)$ eine kontextfreie Grammatik.

  Die von G **erzeugte Sprache** ist definiert als

$$L(G) = \{ \, w \in T^* \mid S \xRightarrow{*} w \, \}.$$

- Eine Sprache heißt kontextfrei, wenn sie von einer kfG erzeugt wird.

Kontextfreie Sprache - Beispiel

- Konstruieren eine kfG für $L = \{\, a^n b^n \mid n \geq 0 \,\}$.
- *Idee:* Erzeugen von einem Nichtterminal gleichzeitig ein a nach links und ein b nach rechts: $A \to aAb$.
- Dies kann wiederholt werden, bis durch Anwendung von $A \to \varepsilon$ ein Terminalwort entsteht.
- Da ε in der Sprache L ist, muss A nicht vom Axiom unterschieden werden.

➢ $G = (\{A\}, \{a, b\}, \{A \to aAb, A \to \varepsilon\}, A)$

$$
\begin{array}{ccccccccccc}
A & \Longrightarrow & aAb & \Longrightarrow & aaAbb & \Longrightarrow & \cdots & \Longrightarrow & a^n A b^n & \Longrightarrow & \cdots \\
\Downarrow & & \Downarrow & & \Downarrow & & & & \Downarrow & & \\
\varepsilon & & ab & & aabb & & & & a^n b^n & & \cdots
\end{array}
$$

Korrektheitsbeweis

- $L = \{ a^n b^n \mid n \geq 0 \}$, $G = (\{A\}, \{a, b\}, \{A \to aAb, A \to \varepsilon\}, A)$; *Beh.:* $L(G) = L$
- Zeigen durch Induktion: $A \overset{n}{\Longrightarrow} w$ **gdw.** $w \in \{a^n A b^n, a^{n-1} b^{n-1}\}$ für alle $n \geq 1$.

  - **I.A.:** $n = 1$. $A \Longrightarrow w$ durch Anwendung einer der Regeln, die das Axiom A ersetzen. Somit $A \Longrightarrow aAb$, $A \Longrightarrow \varepsilon$ und keine weiteren direkten Ableitungen.
    Also $A \overset{1}{\Longrightarrow} w$ **gdw.** $w \in \{aAb, \varepsilon\} = \{a^1 A b^1, a^0 b^0\}$.
  - **I.V.:** $A \overset{n}{\Longrightarrow} w$ **gdw.** $w \in \{a^n A b^n, a^{n-1} b^{n-1}\}$
  - **I.S.:** Keine Ableitung von $a^{n-1} b^{n-1}$, also genau zwei Ableitungen der Länge $n + 1$:
    - $A \overset{n}{\Longrightarrow} a^n A b^n \Longrightarrow a^{n+1} A b^{n+1}$ durch Anwendung von $A \to aAb$
    - $A \overset{n}{\Longrightarrow} a^n A b^n \Longrightarrow a^n b^n$ durch Anwendung von $A \to \varepsilon$

    Also gilt $A \overset{n+1}{\Longrightarrow} w$ **gdw.** $w \in \{a^{n+1} A b^{n+1}, a^n b^n\}$.

➢ $A \overset{*}{\Longrightarrow} w$ und $w \in T^*$ (also $w \in L(G)$) **gdw.** $w \in \{ a^n b^n \mid n \geq 0 \} = L$. ■

Korrektheitsbeweise

Für den Nachweis von $L(G) = L$ ist es auch hier wieder wichtig, beide „Richtungen“ zu beachten:

1. Es können alle Wörter von L erzeugt werden,
   also $L \subseteq L(G)$.
2. Es kann kein Wort erzeugt werden, dass nicht aus L ist,
   also $L(G) \subseteq L$.

Sprachfamilie

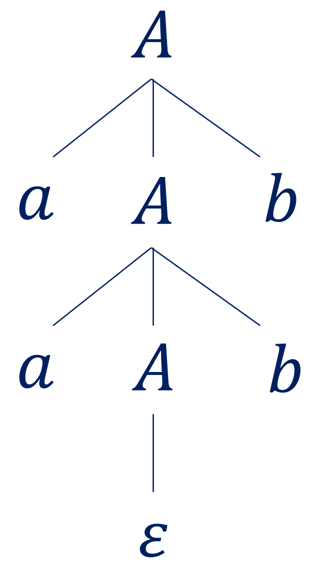
- $\boldsymbol{\mathcal{L}}(\mathbf{CF}) = \{ L \mid L = L(G)$ für eine kfG $G \}$

- $\{ a^n b^n \mid n \geq 0 \} \in \mathcal{L}(\mathrm{CF})$

Ableitungsbäume

- Wurzel markiert mit Axiom
- Innere Knoten mit Nichtterminalen
- Wird Regel $A \to \alpha$ angewendet, dann hat der mit A markierte Knoten $|\alpha| = k$ viele Kindknoten, die (v.l.n.r.) mit den Buchstaben von α markiert sind.
- Bei $A \to \varepsilon$ gibt es einen Kindknoten, der mit ε markiert ist.
- Blätter sind mit Terminalen oder ε markiert.
  V.l.n.r. gelesen ergibt sich das abgeleitete Wort.

Bsp. $A \to aAb \mid \varepsilon$,
$A \Longrightarrow aAb \Longrightarrow aaAbb \Longrightarrow a^2b^2$:

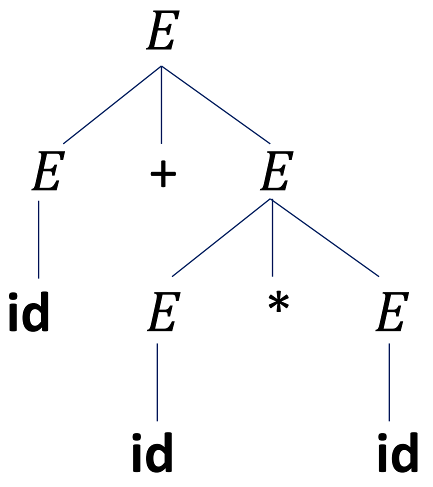


Struktur in den Wörtern

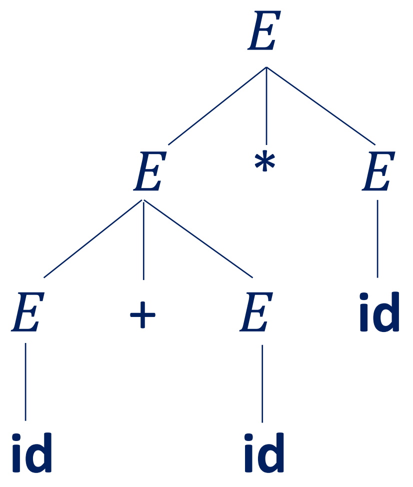
- Ableitungsbäume offenbaren die Struktur in den abgeleiteten Wörtern.
- Bedeutsam für korrekte Übersetzung durch Compiler.
- *am Beispiel* $E \to E + E \mid E * E \mid (E) \mid \mathbf{id}$
  - Es gibt zwei strukturell verschiedene Ableitungen für das terminale Wort $\mathbf{id} + \mathbf{id} * \mathbf{id}$:

$E \Longrightarrow E + E \Longrightarrow E + E * E \overset{3}{\Longrightarrow} \mathbf{id} + \mathbf{id} * \mathbf{id}$

$E \Longrightarrow E * E \Longrightarrow E + E * E \overset{3}{\Longrightarrow} \mathbf{id} + \mathbf{id} * \mathbf{id}$



übliche Vorrang-regel

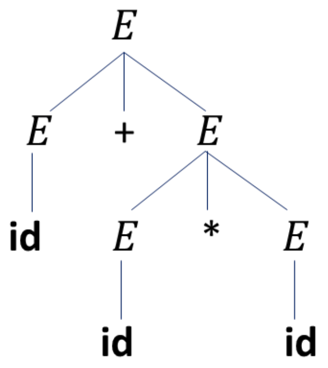


verkehrte Vorrang-regel

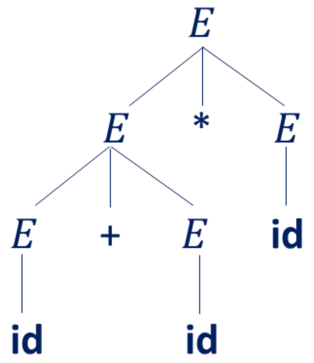
Eindeutigkeit der Zuordnung

Zu jeder Ableitung gehört ein eindeutig bestimmter Ableitungsbaum.

$E \Longrightarrow E + E \Longrightarrow E + E * E \overset{3}{\Longrightarrow} \textbf{id} + \textbf{id} * \textbf{id}$



$E \Longrightarrow E * E \Longrightarrow E + E * E \overset{3}{\Longrightarrow} \textbf{id} + \textbf{id} * \textbf{id}$



Umgekehrt ist die Reihenfolge der Ableitungsschritte i.A. nicht aus dem Baum ersichtlich!

$E \Longrightarrow E + E \Longrightarrow \textbf{id} + E \Longrightarrow \textbf{id} + E * E \Longrightarrow \textbf{id} + \textbf{id} * E \Longrightarrow \textbf{id} + \textbf{id} * \textbf{id}$ und

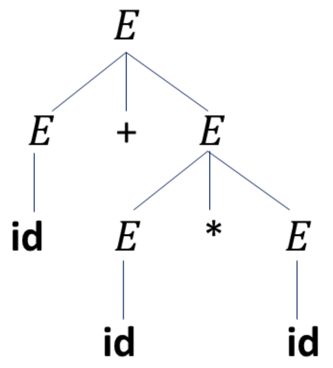
$E \Longrightarrow E + E \Longrightarrow \textbf{id} + E \Longrightarrow \textbf{id} + E * E \Longrightarrow \textbf{id} + E * \textbf{id} \Longrightarrow \textbf{id} + \textbf{id} * \textbf{id}$ und

$E \Longrightarrow E + E \Longrightarrow E + E * E \Longrightarrow E + E * \textbf{id} \Longrightarrow E + \textbf{id} * \textbf{id} \Longrightarrow \textbf{id} + \textbf{id} * \textbf{id}$

werden z.B. durch denselben Ableitungsbaum beschrieben

Links- und Rechtableitungen

- In einer **Linksableitung** wird stets das Nichtterminal ersetzt, das in der Satzform am weitesten links auftritt; für jeden direkten Ableitungsschritt gilt:
  $\boldsymbol{uA\gamma \Longrightarrow_L uv\gamma}$ falls $uA\gamma \Longrightarrow uv\gamma$ und $u \in T^*$.
- In einer **Rechtsableitung** wird stets das Nichtterminal ersetzt, das in der Satzform am weitesten rechts auftritt; für jeden direkten Ableitungsschritt gilt:
  $\boldsymbol{\gamma Au \Longrightarrow_R \gamma vu}$ falls $\gamma Au \Longrightarrow \gamma vu$ und $u \in T^*$.
- **Zu jedem Ableitungsbaum gehört eine eindeutig bestimmte Links- und eine eindeutig bestimmte Rechtsableitung und umgekehrt.**



$$E \Longrightarrow_L E + E \Longrightarrow_L \textbf{id} + E \Longrightarrow_L \textbf{id} + E * E$$
$$\Longrightarrow_L \textbf{id} + \textbf{id} * E \Longrightarrow_L \textbf{id} + \textbf{id} * \textbf{id}$$

$$E \Longrightarrow_R E + E \Longrightarrow_R E + E * E \Longrightarrow_R E + E * \textbf{id}$$
$$\Longrightarrow_R E + \textbf{id} * \textbf{id} \Longrightarrow_R \textbf{id} + \textbf{id} * \textbf{id}$$

Compilerphasen (grob)

Lexikalische Analyse → Syntaktische Analyse → Semantische Analyse → Code-erzeugung

Scanner (Lexer)
Quellcode (Char-Stream) zu Folge von **Token** (z.B. **id**, Schlüsselwörter, Kommentar, …)

Parser
- Syntaxprüfung
- Tokenfolge zu Parsebaum
  (Ableitungsbaum durch Finden einer Links- bzw. Rechtsableitung)

- Prüfung von Constraints
- Symboltabelle/ Scoping
- Typprüfung

- Parsebaum zu Zielcode
- ggf. Code-Optimierung

Scanner und Parser

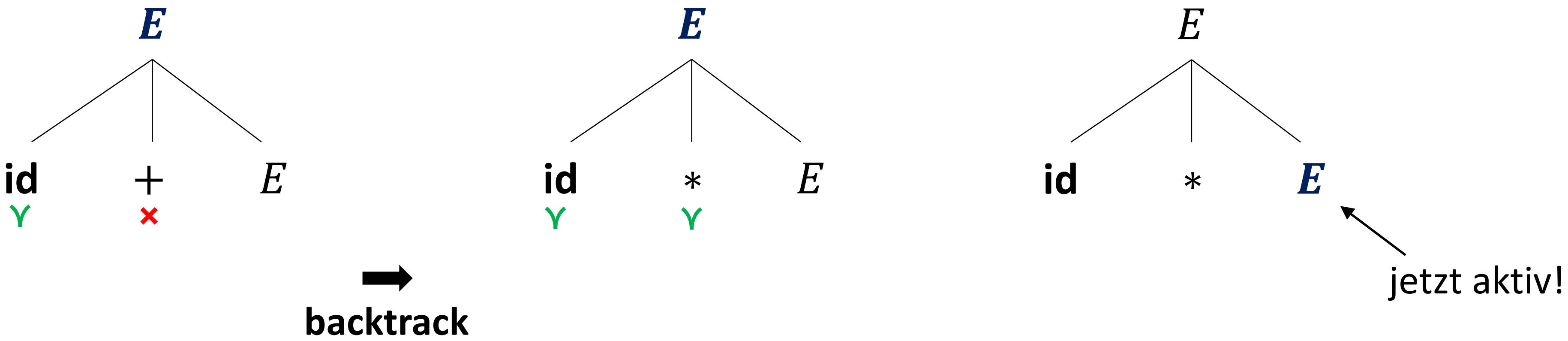
- werden oft automatisch von Compilergeneratoren erzeugt
- **Scanner:**
  - **Token** durch reguläre Ausdrücke spezifiziert
  - aus regulären Ausdrücken werden die DEAs generiert
  - etwas Software „drumherum“
- **Parser:**
  - **Syntax** durch kfG (meist durch erweiterte BNF gegeben) spezifiziert
  - aus der kfG wird der Parser generiert
  - falls Syntaxfehler: Fehlermeldungen erzeugen
  - falls Programm syntaktisch korrekt: Ableitungsbaum ausgeben

Ein einfacher Parser – Prinzip

- **Eingabe:** kfG $G = (N, T, P, S)$, Wort $w \in T^*$
- **Ausgabe:** Ableitungsbaum für w, falls existent (*sonst:* Fehlermeldung)
- Für jedes Nichtterminal A: Festlegen einer Anordnung aller Regeln für A
- Konstruiere den Ableitungsbaum top-down
  - Füge S als Wurzel hinzu, markiere S als **aktiv**
  - Wenn A aktiv, dann wende die nächste A-Regel an
    (führt zu Kindern des Knotens mit aktivem Nichtterminal A)
  - Neu eingeführte Terminale von links nach rechts mit den nächsten noch nicht verglichenen Buchstaben von w vergleichen („Abhaken“)
  - am weitesten links stehendes Nichtterminal als aktiv markieren
  - im Fehlerfall (nicht passende Terminale) zurück zum Elternknoten mit A, nächste Regel

Ein einfacher Parser – Beispiel

$E \rightarrow \mathbf{id} + E \mid \mathbf{id} * E \mid \mathbf{id}$
$w = \mathbf{id} * \mathbf{id}$
✓ ✓

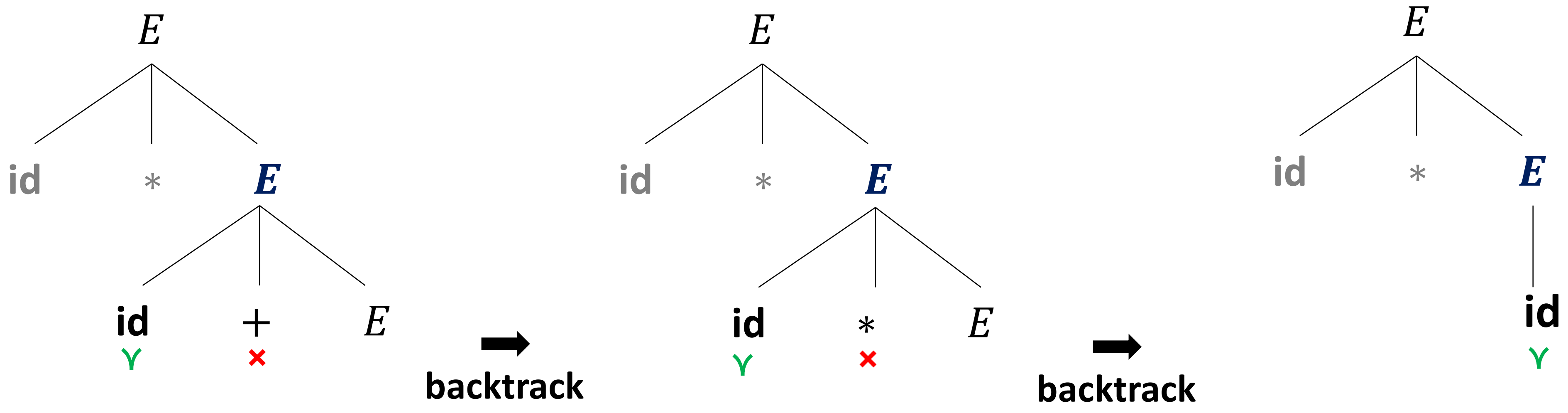


$E \Longrightarrow \mathbf{id} + E$

$E \Longrightarrow \mathbf{id} * E$

Ein einfacher Parser – Beispiel

$E \rightarrow \textbf{id} + E \mid \textbf{id} * E \mid \textbf{id}$
$w = \text{id} * \textbf{id}$



$E \Longrightarrow \text{id} * E \Longrightarrow \text{id} * \textbf{id} + E$ $E \Longrightarrow \text{id} * E \Longrightarrow \text{id} * \textbf{id} * E$ $E \Longrightarrow \text{id} * E \Longrightarrow \text{id} * \textbf{id}$

Ein einfacher Parser – Diskussion

- Aufbau eines Ableitungsbaums von der Wurzel zum Eingabewort („Top-Down-Parsing“)
- Arbeitet stets am linken Ende der bisher gefundenen Satzform
  - Terminal: Abhaken
  - Nichtterminal: Aktivieren (also als nächstes Ersetzen)

➢ Aufbau einer **Linksableitung**

$$E \Longrightarrow_L \mathbf{id} * E \Longrightarrow_L \mathbf{id} * \mathbf{id}$$

- **Backtracking** bei Fehlern

➢ Implementierung mit einem **Stack (linkes Symbol am Top)**

Ein einfacher Parser – Implementierung

$E \to \textbf{id} + E \mid \textbf{id} * E \mid \textbf{id}$

Stack	Eingabe	Aktion
E	$\textbf{id} * \textbf{id}$	$E \to \textbf{id} + E$
$\textbf{id} + E$	$\textbf{id} * \textbf{id}$	Terminal
$+ E$	$* \textbf{id}$	Backtrack
E	$\textbf{id} * \textbf{id}$	$E \to \textbf{id} * E$
$\textbf{id} * E$	$\textbf{id} * \textbf{id}$	Terminal
$* E$	$* \textbf{id}$	Terminal
E	$\textbf{id}$	$E \to \textbf{id} + E$
$\textbf{id} + E$	$\textbf{id}$	Terminal
$+ E$	ε	Backtrack

Ein einfacher Parser – Implementierung

$E \rightarrow \textbf{id} + E \mid \textbf{id} * E \mid \textbf{id}$

Stack	Eingabe	Aktion
E	**id**	$E \rightarrow \textbf{id} * E$
$\textbf{id} * E$	**id**	Terminal
$* E$	ε	Backtrack
E	**id**	$E \rightarrow \textbf{id}$
id	**id**	Terminal
ε	ε	ACCEPT