

# 1. Kanonische Ableitung und Reduktion

---

a)

Rechtskanonisch:

$$*E* \Rightarrow$$

$$- *T* \Rightarrow$$

$$- T * *F* \Rightarrow$$

$$- T * ( *E* ) \Rightarrow$$

$$- T * ( E + *T* ) \Rightarrow$$

$$- T * ( E + T / *F* ) \Rightarrow$$

$$- T * ( E + *T* / v ) \Rightarrow$$

$$- T * ( E + *F* / v ) \Rightarrow$$

$$- T * ( *E* + v / v ) \Rightarrow$$

$$- T * ( *T* + v / v ) \Rightarrow$$

$$- T * ( *F* + v / v ) \Rightarrow$$

$$- *T* * ( v + v / v ) \Rightarrow$$

$$- *F* * ( v + v / v ) \Rightarrow$$

$$- v * ( v + v / v )$$

Linkskanonisch:

$$*E* \Rightarrow$$

$$- *T* \Rightarrow$$

$$- *T* * F \Rightarrow$$

$$- *F* * F \Rightarrow$$

$$- v * *F* \Rightarrow$$

$$- v * ( *E* ) \Rightarrow$$

$$- v * ( *E* + T ) \Rightarrow$$

$$- v * ( *T* + T ) \Rightarrow$$

$$- v * ( *F* + T ) \Rightarrow$$

$$- v * ( v + *T* ) =>$$

$$- v * ( v + *T* / F ) =>$$

$$- v * ( v + *F* / F ) =>$$

$$- v * ( v + v / *F* ) =>$$

$$- v * ( v + v / v )$$

b)

Rechtskanonisch:

$$( ( v + v ) * v / v ) - ( v / *v* ) =>$$

$$\cancel{( ( v + v ) * v / v ) - ( v / *F* )} \Rightarrow$$

$$\cancel{( ( v + v ) * v / v ) - ( F / T )} \Rightarrow$$

$$( ( v + v ) * v / v ) - ( *v* / F ) =>$$

$$( ( v + v ) * v / v ) - ( *F* / F ) =>$$

$$( ( v + v ) * v / v ) - ( *T / F* ) =>$$

$$( ( v + v ) * v / v ) - ( *T* ) =>$$

$$( ( v + v ) * v / v ) - *( E )* =>$$

$$( ( v + v ) * v / v ) - *F* =>$$

$$( ( v + v ) * v / *v* ) - T =>$$

$$( ( v + v ) * *v* / F ) - T =>$$

$$( ( v + *v* ) * F / F ) - T =>$$

$$( ( v + *F* ) * F / F ) - T =>$$

$$( ( *v* + T ) * F / F ) - T =>$$

$$( ( *F* + T ) * F / F ) - T =>$$

$$( ( *T* + T ) * F / F ) - T =>$$

$$( ( *E + T* ) * F / F ) - T =>$$

$$( *( E )* * F / F ) - T =>$$

$$( *F* * F / F ) - T =>$$

$$( *T * F* / F ) - T =>$$

$$( *T / F* ) - T =>$$

E

3 / 11

$$( ( *F* + v ) * v / v ) - ( v / v ) =>$$

$$( ( *T* + v ) * v / v ) - ( v / v ) =>$$

$$( ( E + *v* ) * v / v ) - ( v / v ) =>$$

$$( ( E + *F* ) * v / v ) - ( v / v ) =>$$

$$( ( *E + T* ) * v / v ) - ( v / v ) =>$$

$$( *( E ) * * v / v ) - ( v / v ) =>$$

$$( *F* * v / v ) - ( v / v ) =>$$

$$( T * *v* / v ) - ( v / v ) =>$$

$$( *T * F* / v ) - ( v / v ) =>$$

$$( T / *v* ) - ( v / v ) =>$$

$$( *T / F* ) - ( v / v ) =>$$

$$( *T* ) - ( v / v ) =>$$

$$*( E ) * - ( v / v ) =>$$

$$*F* - ( v / v ) =>$$

$$*T* - ( v / v ) =>$$

$$E - ( *v* / v ) =>$$

$$E - ( *F* / v ) =>$$

$$E - ( T / *v* ) =>$$

$$E - ( *T / F* ) =>$$

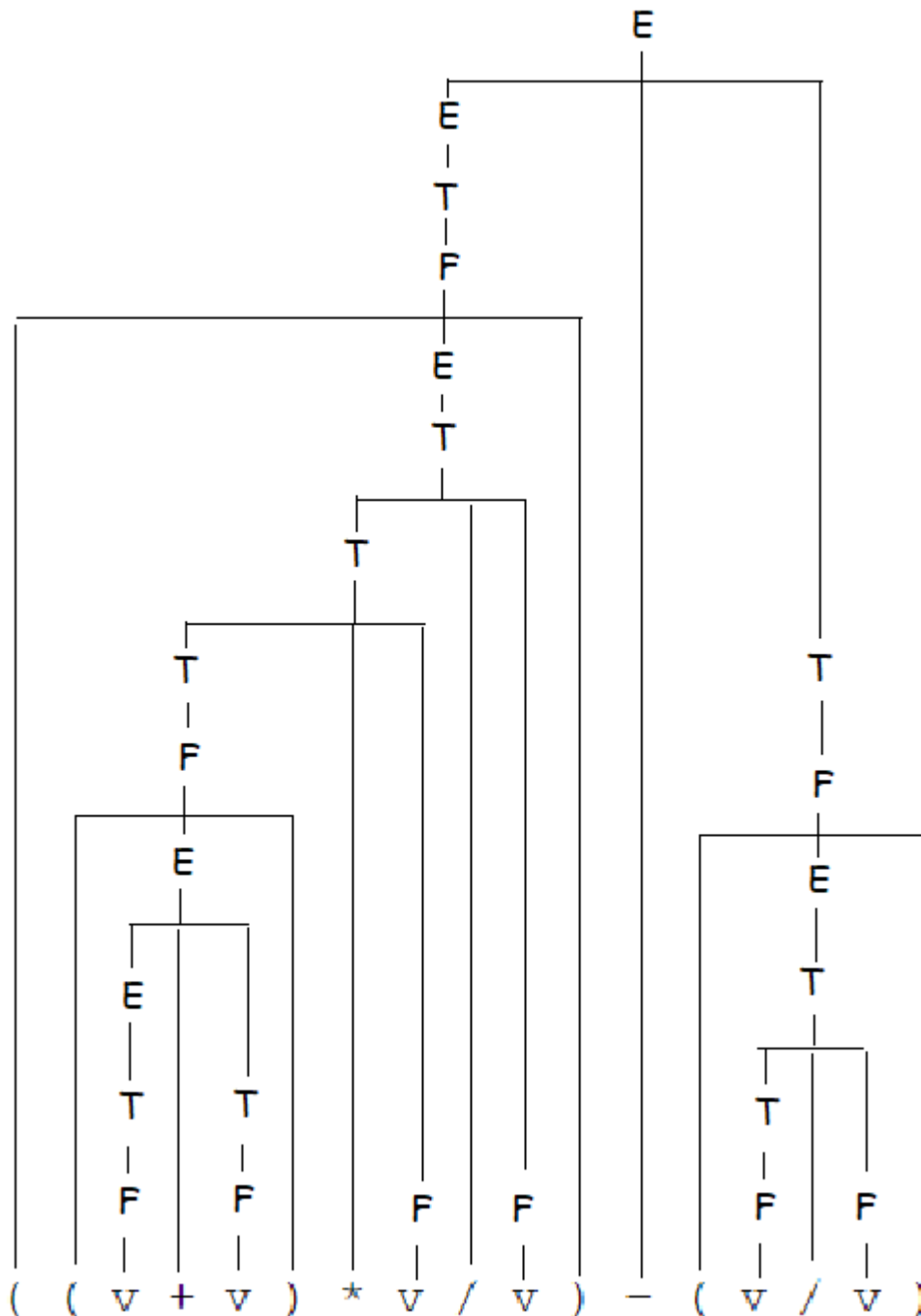
$$E - ( *T* ) =>$$

$$E - *( E ) * =>$$

$$E - *F* =>$$

$$*E - T* =>$$

$$E$$



Die Anzahl der Reduktionen ist bei beiden Varianten gleich (24 Reduktionen) und die Syntaxbäume sind ident.

## 2. Mehrdeutigkeit, Beschreibung und Schreibweisen

a)

Bei der Regel **frac** liegt die Mehrdeutigkeit, da man auf **n** kommen kann, indem man entweder die 1. Alternative verwendet, oder die 2. Alternative und dann das dazukommende **frac** ein  $\epsilon$  ableitet.

Beispiel: 6.9

**real** => **mant** => **sign int . frac** =>  $\epsilon$  **int . frac** =>  $\epsilon$  **n . frac**

**option 1**

**option 2**

option 1	option 2
$\Rightarrow \varepsilon n . n$	$\Rightarrow \varepsilon n . \text{frac } n$
$\Rightarrow \varepsilon 6 . n$	$\Rightarrow \varepsilon 6 . \text{frac } n$
$\Rightarrow \varepsilon 6 . 9$	$\Rightarrow \varepsilon 6 . \varepsilon n$
$\Rightarrow \varepsilon 6 . 9$	

Änderung:  $\text{frac} \rightarrow n \mid \text{frac } n \mid \varepsilon .$  auf  $\text{frac} \rightarrow \text{frac } n \mid \varepsilon .$ , der Rest bleibt gleich.

## b) Äquivalente, eindeutige Grammatik

Möglichst wenige Regeln:

$G(\text{real})$ :

$\text{real} = ['+' | '-'] (0 | \dots | 9) \{0 | \dots | 9\} ['.' \{0 | \dots | 9\}] ['E' ['+' | '-'] (0 | \dots | 9) \{0 | \dots | 9\}] .$

"Kürzer":

$\text{real} = \text{optSign } n \{n\} ['.' \{n\}] ['E' \text{optSign } n \{n\}] .$

$\text{optSign} = ['+' | '-'] .$

$n = 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 .$

## 3. Reguläre Grammatiken

---

a)

Sätze:

- $\varepsilon$
- $ab$
- $ab(ab)^*$
- $bb$
- $bb(b)^*$

(oder man verwendet die Impl. aus Übung 1):

```

1  G(S):
2  S -> b A | a B | eps
3  A -> b A | b
4  B -> b C | b
5  C -> a B

```

Microsoft Visual Studio Debug Console

```

language(g2):
L(G(S)): maxLength=6 {
eps
a b
a b a b
a b a b a b
b b
b b b
b b b b
b b b b b
b b b b b b
}

```

$S \rightarrow A b \mid B b \mid \epsilon .$

$A \rightarrow a \mid C a .$

$C \rightarrow A b .$

$B \rightarrow b \mid B b .$

b)

$ab(ab)^* + bb(b)^* + \epsilon$  oder besser  $(ab)^* + (b)^*$  (erspaart das  $\epsilon$ )

## 4. Bezeichner in der Programmiersprache Ada

---

a)

$B \rightarrow l \mid l R$

$R \rightarrow d \mid l \mid \text{'\_'} U \mid l R \mid d R$

$U \rightarrow l \mid d \mid l R \mid d R$

b)

$B \rightarrow l \mid R l \mid R d$

$R \rightarrow U \text{'\_'} \mid R l \mid R d \mid l$

$U \rightarrow R d \mid R l \mid l$

c)

$l ( l + d + \text{'\_'} l + \text{'\_'} d )^*$

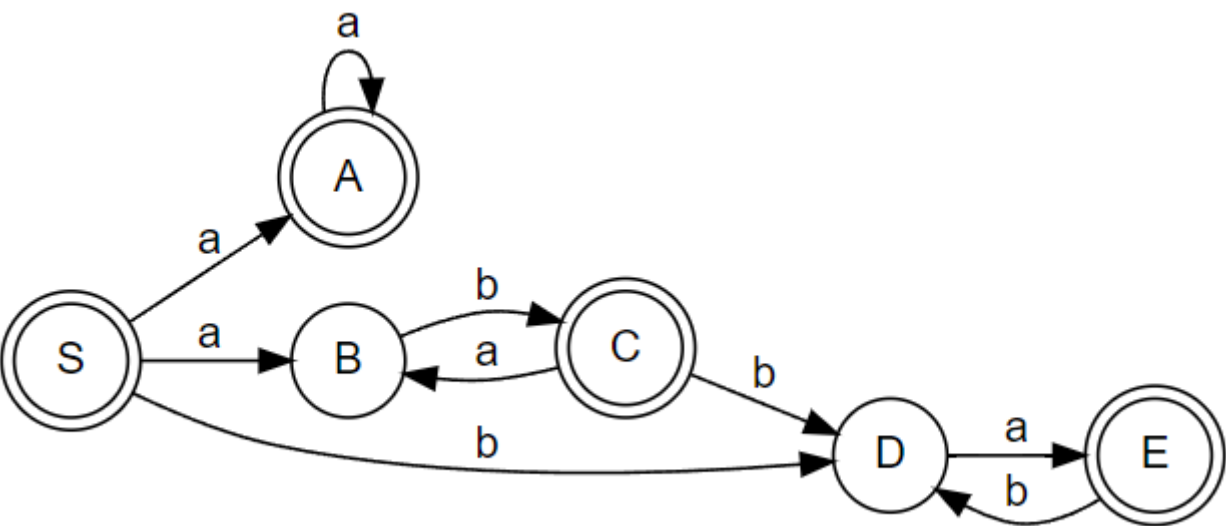
Unix:

$1(1|d|_1|_d)^* = ^1 = 1(_?(1|d))^* = ^1 = 1(_?[1d])^*$

# 5. Transformation zwischen Darstellungsformen regulärer Sprachen

a)

```
digraph non_deterministic_finite_state_machine {
  fontname="Helvetica,Arial,sans-serif"
  node [fontname="Helvetica,Arial,sans-serif"]
  edge [fontname="Helvetica,Arial,sans-serif"]
  rankdir=LR;
  node [shape = doublecircle]; S A C E;
  node [shape = circle];
  S -> A [label = "a"]
  A -> A [label = "a"]
  S -> B [label = "a"]
  B -> C [label = "b"]
  S -> D [label = "b"]
  C -> D [label = "b"]
  C -> B [label = "a"]
  D -> E [label = "a"]
  E -> D [label = "b"]
}
```



	0	1
-> S	{A, B}	{D}
o A	{A}	-



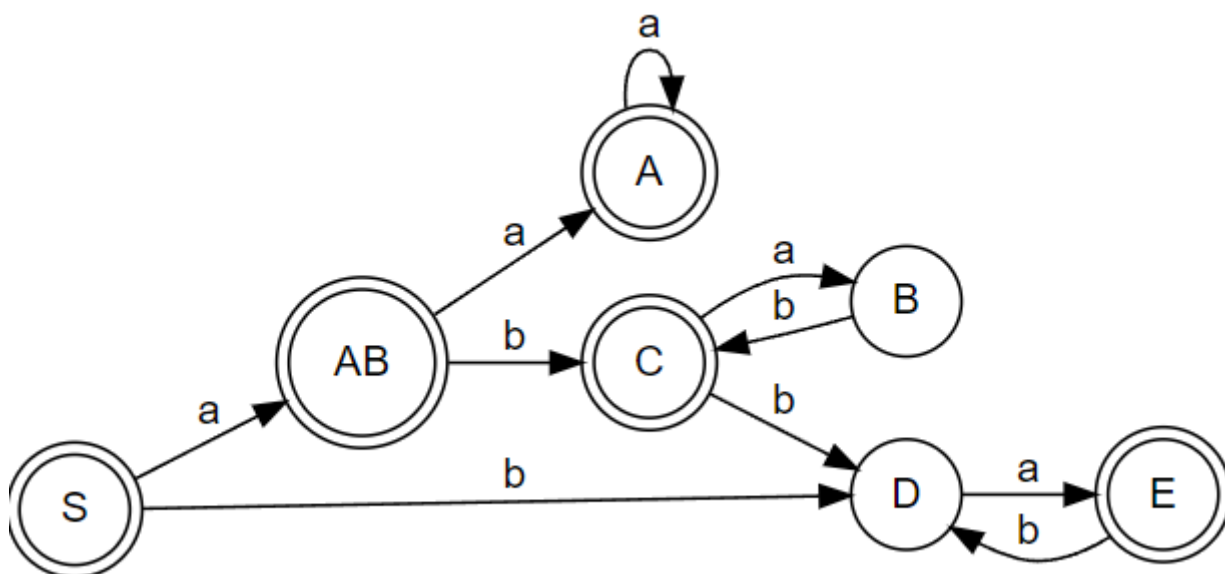
	0	1
B	-	{C}
o C	{B}	{D}
D	{E}	-
o E	-	{D}
o {A, B}	{A}	{C}

```

digraph deterministic_finite_state_machine {
    fontname="Helvetica,Arial,sans-serif"
    node [fontname="Helvetica,Arial,sans-serif"]
    edge [fontname="Helvetica,Arial,sans-serif"]
    rankdir=LR;
    node [shape = doublecircle]; S A C E AB;
    node [shape = circle];

    S -> AB [label = "a"]
    AB -> A [label = "a"]
    AB -> C [label = "b"]
    A -> A [label = "a"]
    B -> C [label = "b"]
    S -> D [label = "b"]
    C -> D [label = "b"]
    C -> B [label = "a"]
    D -> E [label = "a"]
    E -> D [label = "b"]
}

```



b)

Erläuterung:

Ich habe mir angesehen was mit der Teilbarkeit von Binärzahlen passiert, wenn man eine 1 oder 0 anhängt bzw. wie Binärzahlen dividiert durch 3 in andere Restklassen wandern.

Wenn man eine "0" anhängt, wird eine Binärzahl verdoppelt. Der Rest bei der Division durch 3 bleibt gleich. Wenn man eine "1" anhängt, wird eine Binärzahl verdoppelt und dann inkrementiert. Hat die Binärzahl dividiert durch 3 die Restklasse 2, dann bleibt sie nach dieser Operation in dieser Restklasse.

Wenn die Zahl bereits durch 3 Teilbar ist und man hängt

- 0 an, dann bleibt die Zahl durch 3 Teilbar => Restklasse 0. Man kann also beliebig viele 0er anhängen.
- 1 an, ergibt sich ein Rest von 1 => Restklasse 1

Wenn die Zahl geteilt durch 3 einen Rest von 1 hat und man hängt

- 0 an, dann ergibt sich ein Rest von 2 => Restklasse 2
- 1 an, dann ergibt sich ein Rest von  $2r + 1 = 3 = 0$  => Zahl ist durch 3 Teilbar => Restklasse 0

Wenn die Zahl geteilt durch 3 einen Rest von 2 hat und man hängt

- 0 an, dann ergibt sich ein Rest von 4 => Restklasse 1
- 1 an, dann ergibt sich ein Rest von  $2r + 1 = 8$  => Restklasse 2. Man kann also beliebig viele 1er anhängen.

Nach diesem Schema wurde der Automat gebaut. Die Namen der Zustände entsprechen den Restklassen.

```
digraph deterministic_finite_state_machine {
    fontname="Helvetica,Arial,sans-serif"
    node [fontname="Helvetica,Arial,sans-serif"]
    edge [fontname="Helvetica,Arial,sans-serif"]
    rankdir=LR;
    node [shape = doublecircle]; S R0;
    node [shape = circle];

    S -> R0 [label = "0"]
    S -> R1 [label = "1"]
    R0 -> R0 [label = "0"]
    R0 -> R1 [label = "1"]
    R1 -> R2 [label = "0"]
    R2 -> R2 [label = "1"]
    R2 -> R1 [label = "0"]
    R1 -> R0 [label = "1"]
}
```

	0	1
-> S	{R0}	{R1}
o R0	{R0}	{R1}

	0	1
R1	{R2}	{R0}
R2	{R1}	{R2}

Anmerkung: man braucht das S für den Start, da der Automat sonst eine Binärzahl ohne Zeichen erlauben würde. (glaube ich?)

