„Vor lauter Bäumen...“
XML und formale Sprachen

Marburg
März 2005

Thomas Schwentick

Bevor es losgeht...

Meine Forschungsgebiete

- Grundlagen von Datenbanken
- Logik in der Informatik
- Komplexitätstheorie
- Automaten und Formale Sprachen

Aktueller Forschungsschwerpunkt

Grundlagen von Anfragesprachen für semistrukturierte Daten

Ziel des Vortrags

- XML und Datenbanken
- Formalsprachliche Konzepte im Zusammenhang mit XML
- Verwendbarkeit für die Schule?
- Keine (!) Einführung in XML

XML: Ein erster Blick

XML ...

⟨Composer⟩
⟨Name⟩ Claude Debussy ⟨/Name⟩
⟨Vita⟩
⟨Born⟩ ⟨When⟩ August 22, 1862 ⟨/When⟩⟨Where⟩ Paris ⟨/Where⟩⟨/Born⟩
⟨Married⟩⟨When⟩ October 1899 ⟨/When⟩⟨Whom⟩ Rosalie⟨/Whom⟩⟨/Married⟩
⟨Married⟩⟨When⟩ January 1908 ⟨/When⟩⟨Whom⟩ Emma ⟨/Whom⟩⟨/Married⟩
⟨Died⟩⟨When⟩ March 25, 1918 ⟨/When⟩⟨Where⟩ Paris ⟨/Where⟩ ⟨/Died⟩
⟨/Vita⟩
⟨Piece⟩
⟨PTitle⟩ La Mer ⟨/PTitle⟩
⟨PYear⟩ 1905 ⟨/PYear⟩
⟨Instruments⟩ Large orchestra ⟨/Instruments⟩
⟨Movements⟩ 3 ⟨/Movements⟩
...
⟨/Piece⟩
...
⟨/Composer⟩
...

XML: Ein erster Blick

XML ...

```
⟨Composer⟩
   ⟨Name⟩ Claude Debussy ⟨/Name⟩
   ⟨Vita⟩
      ⟨Born⟩ ⟨When⟩ August 22, 1862 ⟨/When⟩⟨Where⟩ Paris ⟨/Where⟩⟨/Born⟩
      ⟨Married⟩⟨When⟩ October 1899 ⟨/When⟩⟨Whom⟩ Rosalie⟨/Whom⟩⟨/Married⟩
      ⟨Married⟩⟨When⟩ January 1908 ⟨/When⟩⟨Whom⟩ Emma ⟨/Whom⟩⟨/Married⟩
      ⟨Died⟩⟨When⟩ March 25, 1918 ⟨/When⟩⟨Where⟩ Paris ⟨/Where⟩ ⟨/Died⟩
   ⟨/Vita⟩
   ⟨Piece⟩
      ⟨PTitle⟩ La Mer ⟨/PTitle⟩
      ⟨PYear⟩ 1905 ⟨/PYear⟩
      ⟨Instruments⟩ Large orchestra ⟨/Instruments⟩
      ⟨Movements⟩ 3 ⟨/Movements⟩
      ...
   ⟨/Piece⟩
   ...
⟨/Composer⟩
...
```

Beobachtungen

- XML ist einfach
-
-
-
-

XML: Ein erster Blick

XML ...

```
⟨Composer⟩
   ⟨Name⟩ Claude Debussy ⟨/Name⟩
   ⟨Vita⟩
      ⟨Born⟩ ⟨When⟩ August 22, 1862 ⟨/When⟩⟨Where⟩ Paris ⟨/Where⟩⟨/Born⟩
      ⟨Married⟩⟨When⟩ October 1899 ⟨/When⟩⟨Whom⟩ Rosalie⟨/Whom⟩⟨/Married⟩
      ⟨Married⟩⟨When⟩ January 1908 ⟨/When⟩⟨Whom⟩ Emma ⟨/Whom⟩⟨/Married⟩
      ⟨Died⟩⟨When⟩ March 25, 1918 ⟨/When⟩⟨Where⟩ Paris ⟨/Where⟩ ⟨/Died⟩
   ⟨/Vita⟩
   ⟨Piece⟩
      ⟨PTitle⟩ La Mer ⟨/PTitle⟩
      ⟨PYear⟩ 1905 ⟨/PYear⟩
      ⟨Instruments⟩ Large orchestra ⟨/Instruments⟩
      ⟨Movements⟩ 3 ⟨/Movements⟩
      ...
   ⟨/Piece⟩
   ...
⟨/Composer⟩
...
```

Beobachtungen

- XML ist einfach
- XML ist selbstbeschreibend
-
-
-

XML: Ein erster Blick

XML ...

```
⟨Composer⟩
   ⟨Name⟩ Claude Debussy ⟨/Name⟩
   ⟨Vita⟩
      ⟨Born⟩ ⟨When⟩ August 22, 1862 ⟨/When⟩⟨Where⟩ Paris ⟨/Where⟩⟨/Born⟩
      ⟨Married⟩⟨When⟩ October 1899 ⟨/When⟩⟨Whom⟩ Rosalie⟨/Whom⟩⟨/Married⟩
      ⟨Married⟩⟨When⟩ January 1908 ⟨/When⟩⟨Whom⟩ Emma ⟨/Whom⟩⟨/Married⟩
      ⟨Died⟩⟨When⟩ March 25, 1918 ⟨/When⟩⟨Where⟩ Paris ⟨/Where⟩ ⟨/Died⟩
   ⟨/Vita⟩
   ⟨Piece⟩
      ⟨PTitle⟩ La Mer ⟨/PTitle⟩
      ⟨PYear⟩ 1905 ⟨/PYear⟩
      ⟨Instruments⟩ Large orchestra ⟨/Instruments⟩
      ⟨Movements⟩ 3 ⟨/Movements⟩
      ...
   ⟨/Piece⟩
   ...
⟨/Composer⟩
...
```

Beobachtungen

- XML ist einfach
- XML ist selbstbeschreibend
- XML ist flexibel
-
-

XML: Ein erster Blick

XML ...

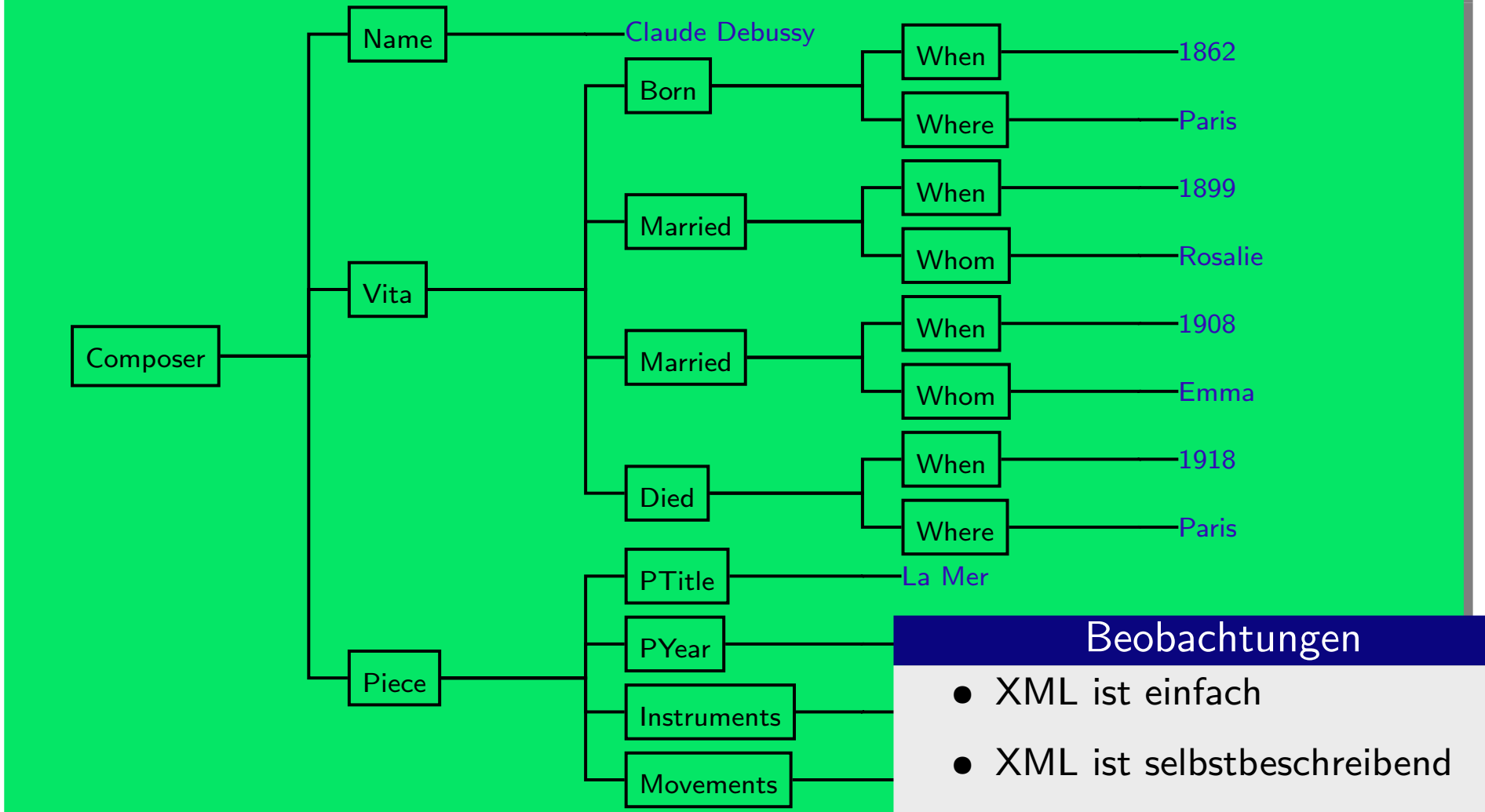
```
⟨Composer⟩
   ⟨Name⟩ Claude Debussy ⟨/Name⟩
   ⟨Vita⟩
      ⟨Born⟩ ⟨When⟩ August 22, 1862 ⟨/When⟩⟨Where⟩ Paris ⟨/Where⟩⟨/Born⟩
      ⟨Married⟩⟨When⟩ October 1899 ⟨/When⟩⟨Whom⟩ Rosalie⟨/Whom⟩⟨/Married⟩
      ⟨Married⟩⟨When⟩ January 1908 ⟨/When⟩⟨Whom⟩ Emma ⟨/Whom⟩⟨/Married⟩
      ⟨Died⟩⟨When⟩ March 25, 1918 ⟨/When⟩⟨Where⟩ Paris ⟨/Where⟩ ⟨/Died⟩
   ⟨/Vita⟩
   ⟨Piece⟩
      ⟨PTitle⟩ La Mer ⟨/PTitle⟩
      ⟨PYear⟩ 1905 ⟨/PYear⟩
      ⟨Instruments⟩ Large orchestra ⟨/Instruments⟩
      ⟨Movements⟩ 3 ⟨/Movements⟩
      ...
   ⟨/Piece⟩
   ...
⟨/Composer⟩
...
```

Beobachtungen

- XML ist einfach
- XML ist selbstbeschreibend
- XML ist flexibel
- XML ist nichts Besonderes
-

XML: Ein erster Blick

... als Baum



Beobachtungen

- XML ist einfach
- XML ist selbstbeschreibend
- XML ist flexibel
- XML ist nichts Besonderes
- XML ist baumartig

Inhalt

XML: Wozu?

Bemerkungen

- Ursprünglich: Datenaustausch im Internet
- Zusätzlich: Lingua Franca der Datenintegration
- Zwei Welten repräsentierbar: Relationale Daten und Dokumente
- In der Forschung: Neues Datenmodell für Datenbanken
- Allgemeinerer Ansatz: Semistrukturierte Daten (Graph statt Baum)

XML vs. Relationale Datenbanken

Relationale Datenbanken

- Deklarative Anfragen
- Mengenorientierte Auswertungsalgebra
- Effiziente Auswertung
- Starres Schema
- Ordnungsprinzip: Tabellen

XML

- Anfragesprachen: fortgeschrittene Standards
- Navigation
- Auswertungsstrategien: Baustelle
- Ordnungsprinzip: Bäume
- XML-Datenbanken???

XML-Verarbeitung

Vier Arten der Verarbeitung von XML-Daten

Validierung

Test, ob ein Dokument einem gegebenen Typ entspricht

Navigation

Auswahl von Positionen in einem Dokument

Anfragen

Extraktion relevanter Daten aus Dokumenten

Transformation

Konstruktion eines neuen Dokumentes aus einem gegebenen Dokument

XML-Verarbeitung

Vier Arten der Verarbeitung von XML-Daten und ihre Sprachen

Validierung `DTD, XML Schema`

Test, ob ein Dokument einem gegebenen Typ entspricht

Navigation `XPath`

Auswahl von Positionen in einem Dokument

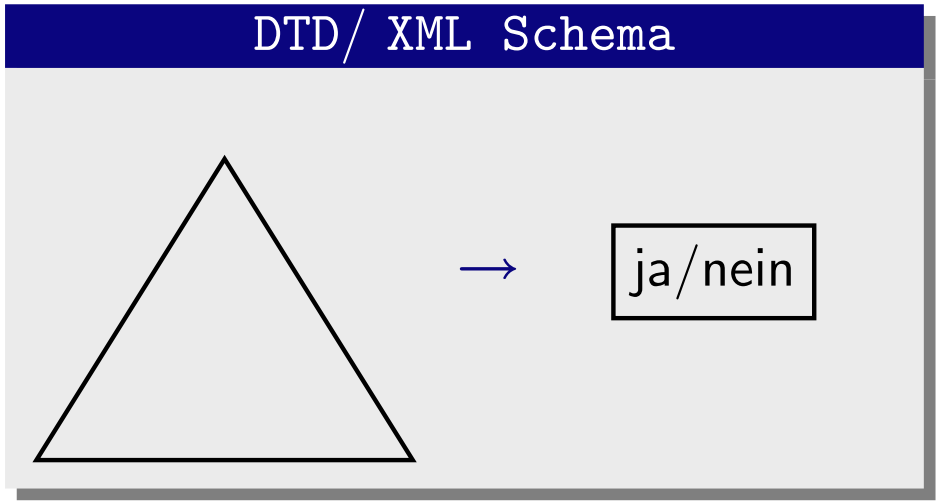
Anfragen `XQuery`

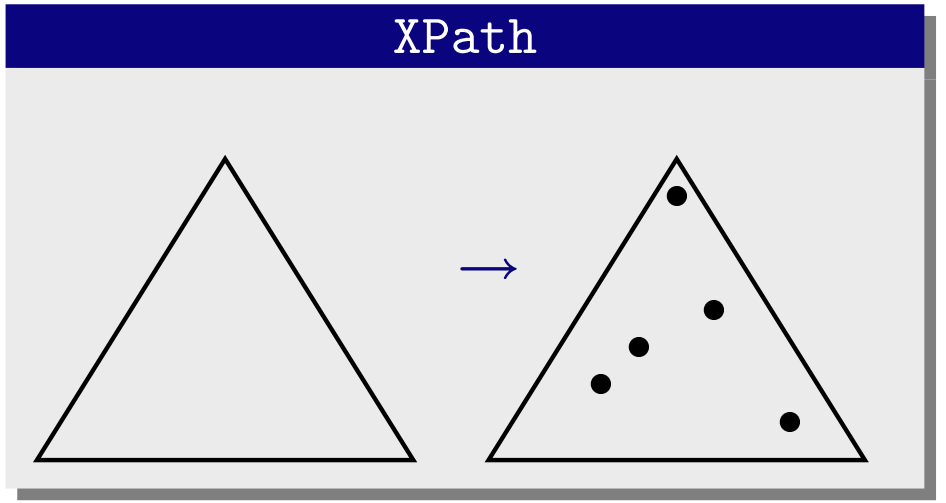
Extraktion relevanter Daten aus Dokumenten

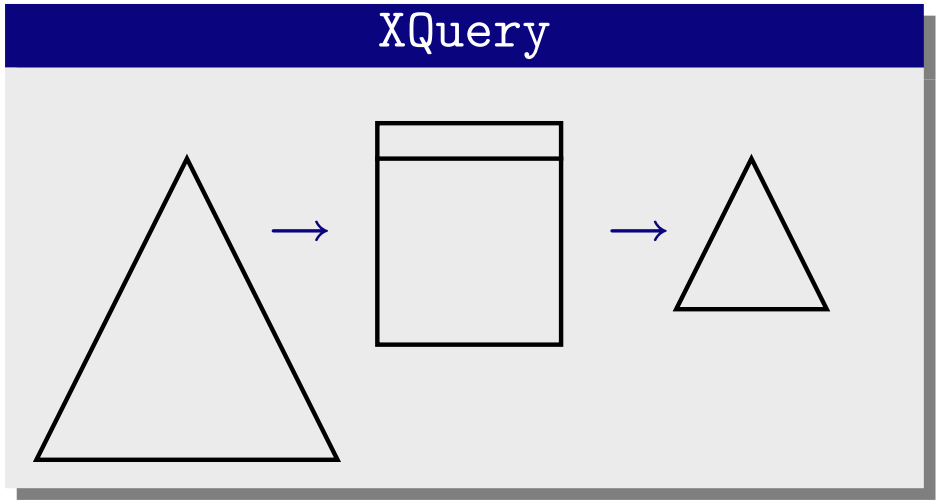
Transformation `XSLT`

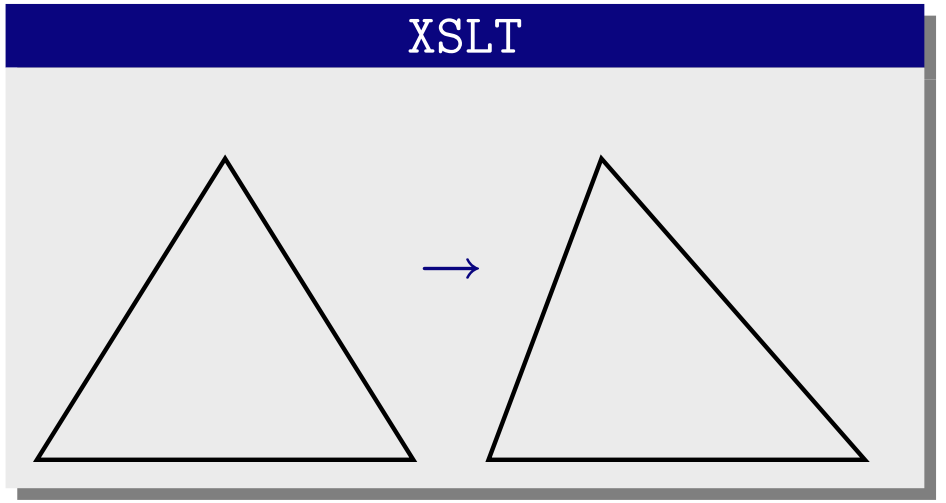
Konstruktion eines neuen Dokumentes aus einem gegebenen Dokument

Schematisch betrachtet...



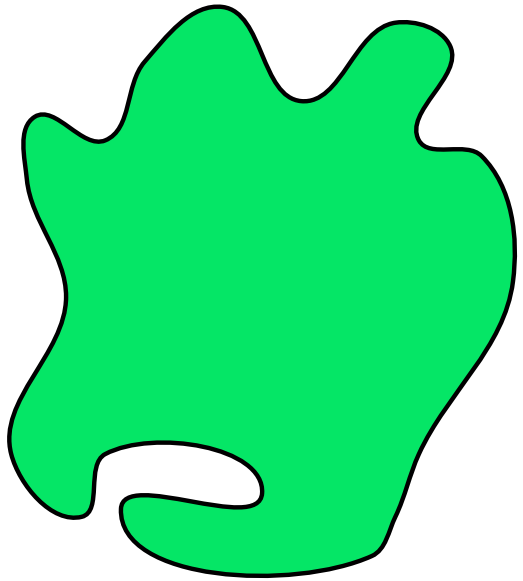






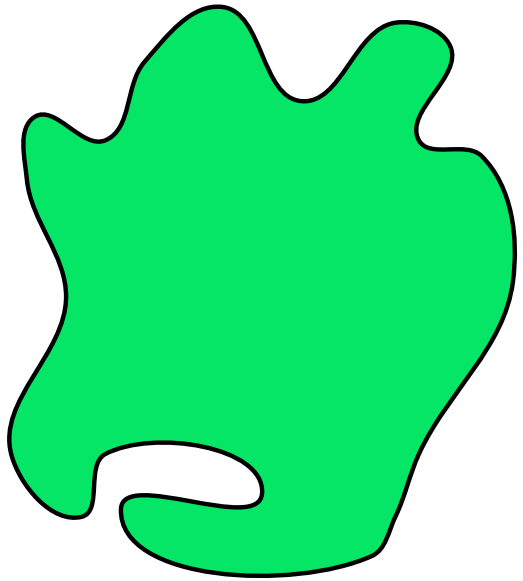
Forschungsziele für Theoretiker

XML-Sprachen

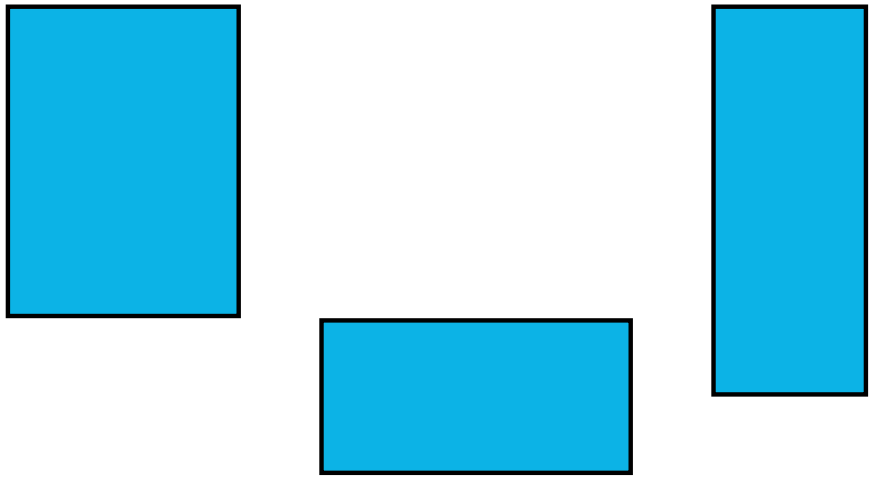


Forschungsziele für Theoretiker

XML-Sprachen



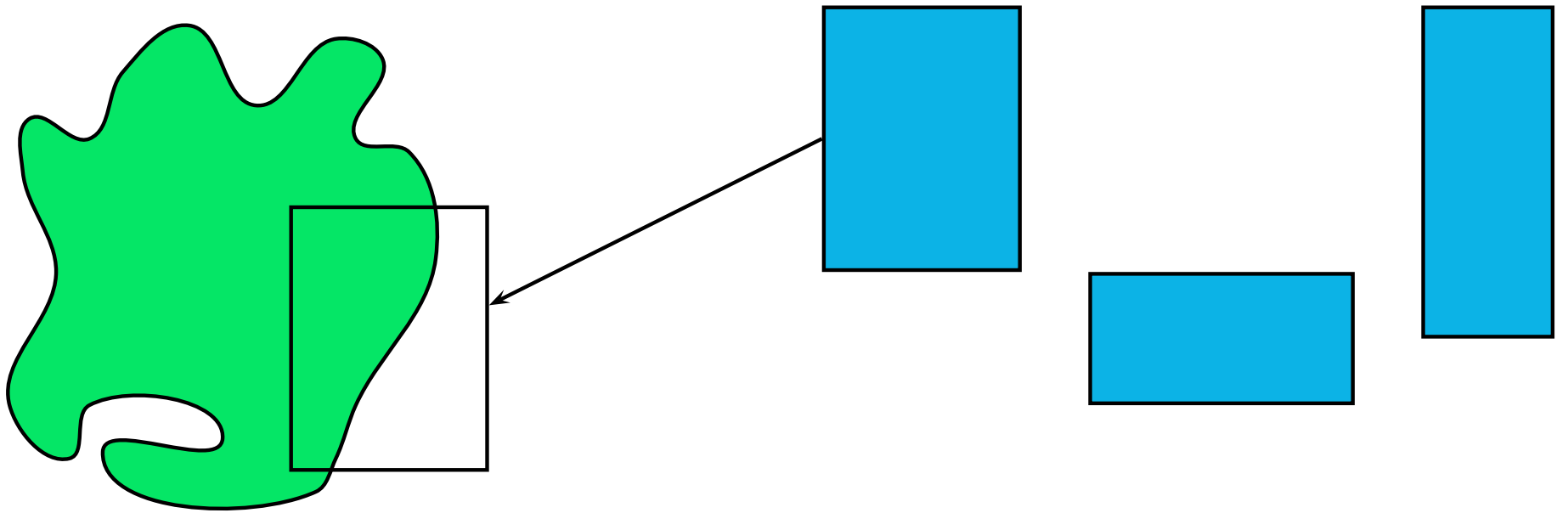
Bekannte formale Modelle



Forschungsziele für Theoretiker

XML-Sprachen

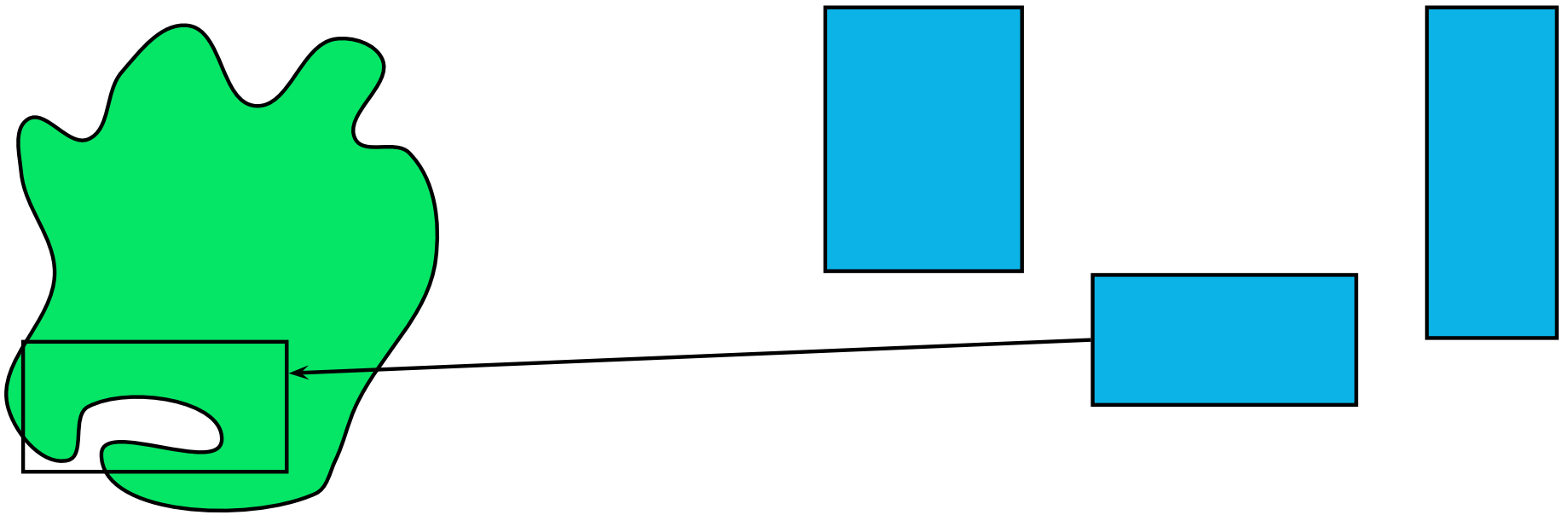
Bekannte formale Modelle



Forschungsziele für Theoretiker

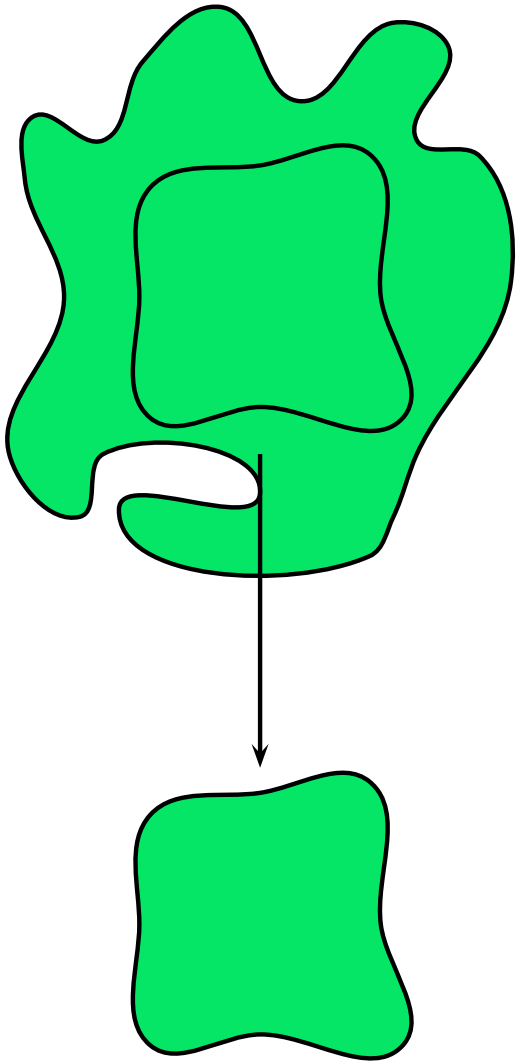
XML-Sprachen

Bekannte formale Modelle

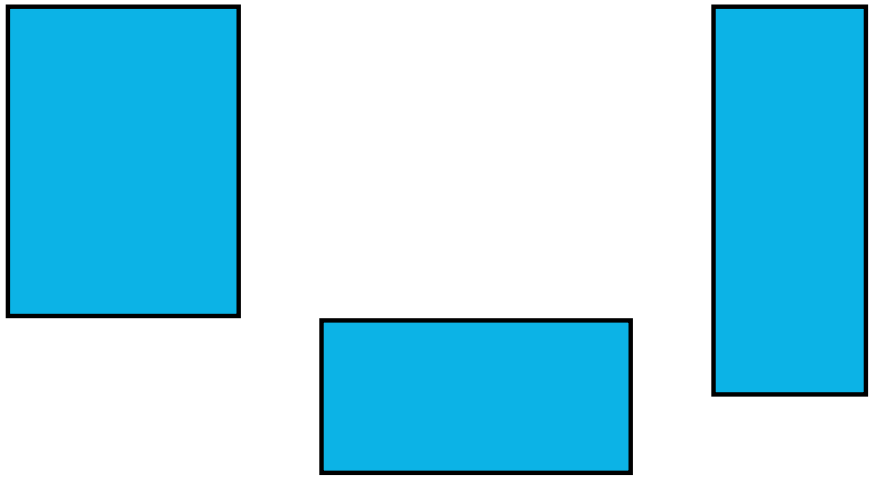


Forschungsziele für Theoretiker

XML-Sprachen



Bekannte formale Modelle

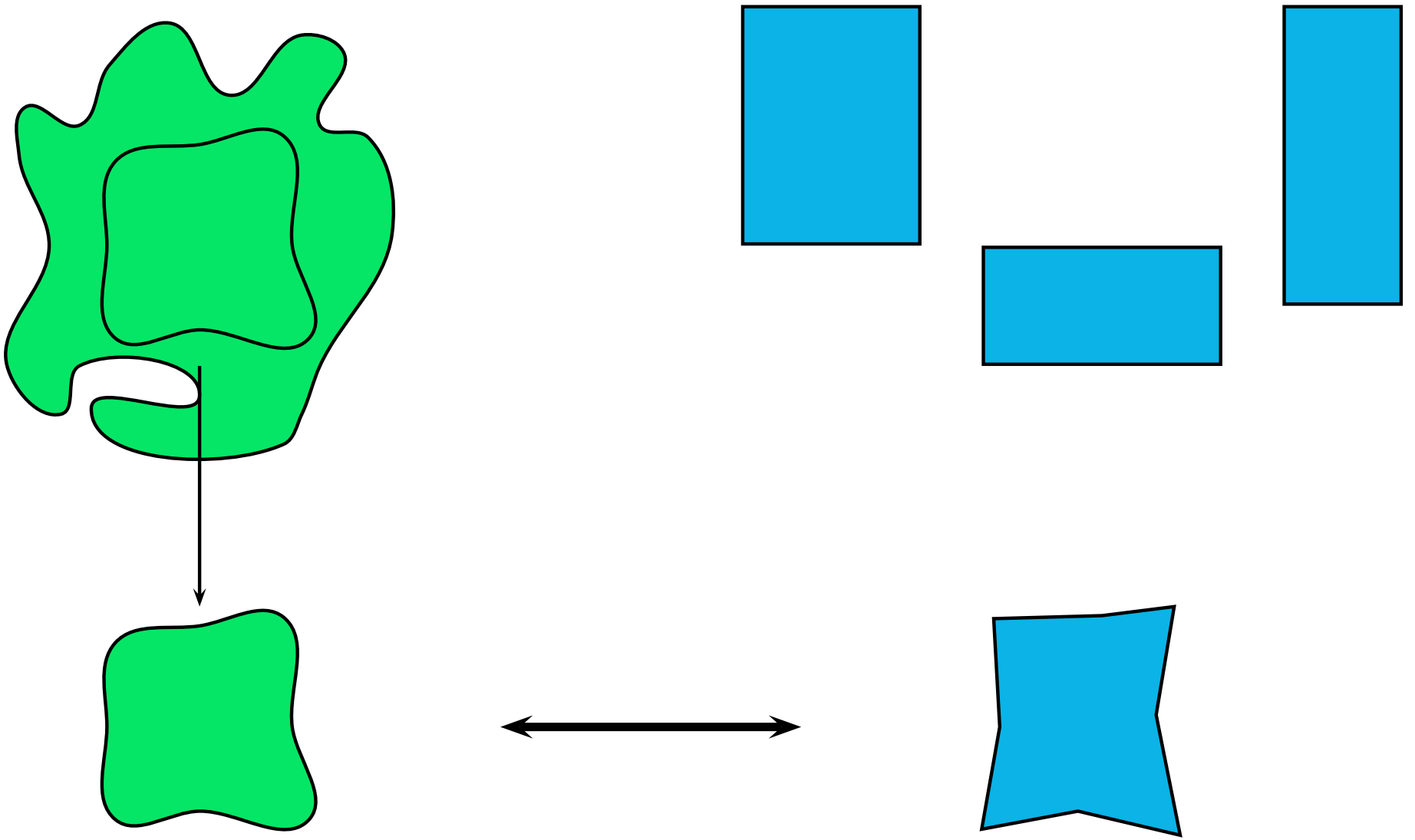


Geeignete Fragmente

Forschungsziele für Theoretiker

XML-Sprachen

Bekannte formale Modelle



Geeignete Fragmente

Angepasste formale Modelle

Inhalt

DTDs und reguläre Ausdrücke

Beispiel: DTD

```
<!DOCTYPE Composers [
    <!ELEMENT Composers (Composer*)>
    <!ELEMENT Composer (Name, Vita, Piece*)>
    <!ELEMENT Vita (Born, Married*, Died?)>
    <!ELEMENT Born (When, Where)>
    <!ELEMENT Married (When, Whom)>
    <!ELEMENT Died (When, Where)>
    <!ELEMENT Piece (PTitle, PYear,
         Instruments, Movements)>
    ]>
```

- DTDs verwenden reguläre Ausdrücke zur Beschreibung der Folge der Kinder eines Knoten
- Einschränkung zur effizienten Auswertung:

  „It is an error if an element in the document can match more than one occurrence of an element type in the content model [without looking ahead]“

  [XML 1.0, W3C recommendation Oktober 2000]

Deterministische reguläre Ausdrücke

- „It is an error if an element in the document can match more than one occurrence of an element type in the content model [without looking ahead]“
- $bc + bd$ erfüllt die Bedingung nicht
- $b(c + d)$ erfüllt sie
- Wie lässt sie sich definieren und effizient testen?

Definition: 1-unambiguous [Brüggemann-Klein, Wood 97]

- Nummeriere Symbole von links nach rechts $\longrightarrow r'$
- r ist **1-unambiguous**, falls gilt:

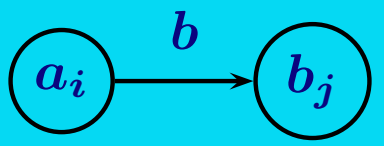
  $ux_iv \in L(r')$ und $uy_jw \in L(r')$, $x_i \neq y_j \Longrightarrow x \neq y$

Beispiel

- $r = bc + bd$ $\qquad r' = b_1c_2 + b_3d_4$
- $b_1c_2 \in L(r')$, $b_3d_4 \in L(r')$, $b_1 \neq b_3$ $\qquad$ aber: $b = b$

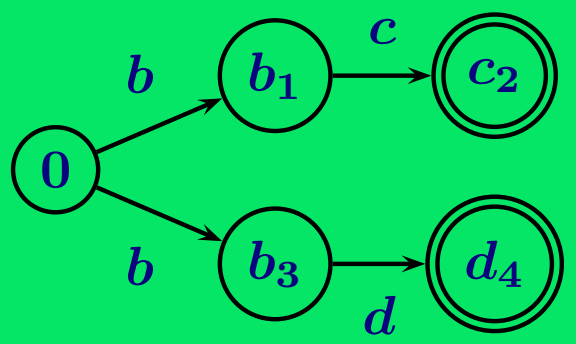
1-unambiguous: Charakterisierung

Definition: Glushkov-Automat

- Sei r' „nummerierter“ regulärer Ausdruck
- Zuständen von $\mathcal{A}_{r'}$: $\{0\} \cup \{\text{alle Zeichen von } r'\}$
- 

falls $ua_ib_jv \in L(r')$

Beispiel

- $r = bc + bd$ $\qquad r' = b_1c_2 + b_3d_4$
- 

Satz [Brüggemann-Klein, Wood 97]

Ein regulär Ausdruck r ist genau dann 1-unambiguous, wenn $\mathcal{A}_{r'}$ deterministisch ist

Inhalt

XML-Dokumente als Ableitungsbäume

Beispiel

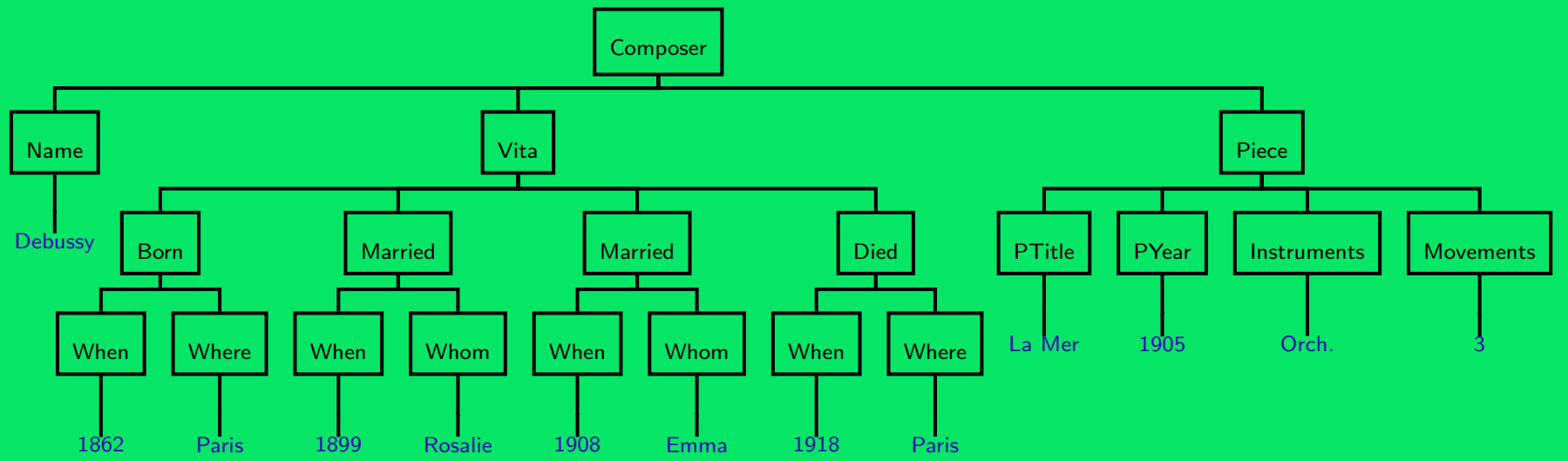
```
<!DOCTYPE Composers [
    <!ELEMENT Composers (Composer*)>
    <!ELEMENT Composer (Name, Vita, Piece*)>
    <!ELEMENT Vita (Born, Married*, Died?)>
    <!ELEMENT Born (When, Where)>
    <!ELEMENT Married (When, Whom)>
    <!ELEMENT Died (When, Where)>
    <!ELEMENT Piece (PTitle, PYear,
         Instruments, Movements)>
    ]>
```

Beobachtung

- DTDs sind im Grunde **erweiterte kontextfreie Grammatiken**:
  - Produktionen haben reguläre Ausdrücke auf der rechten Seite
  - Rest wie kontextfreie Grammatiken
- Gültige XML-Dokumente sind dann Ableitungsbäume zu dieser Grammatik

DTDs und Ableitungsbäume (Forts.)

Beispiel-Teil-Baum



Beispiel-DTD...

```
<!DOCTYPE Composers [
    <!ELEMENT Composers (Composer*)>
    <!ELEMENT Composer (Name, Vita, Piece*)>
    <!ELEMENT Vita (Born, Married*, Died?)>
    <!ELEMENT Born (When, Where)>
    <!ELEMENT Married (When, Whom)>
    <!ELEMENT Died (When, Where)>
    <!ELEMENT Piece (PTitle, PYear,
        Instruments, Movements)>
    ]>
```

...als Grammatik

Composers $\longrightarrow$ Composer*

Composer $\longrightarrow$ Name Vita Piece*

Vita $\longrightarrow$ Born Married* Died?

Born $\longrightarrow$ When Where

Married $\longrightarrow$ When Whom

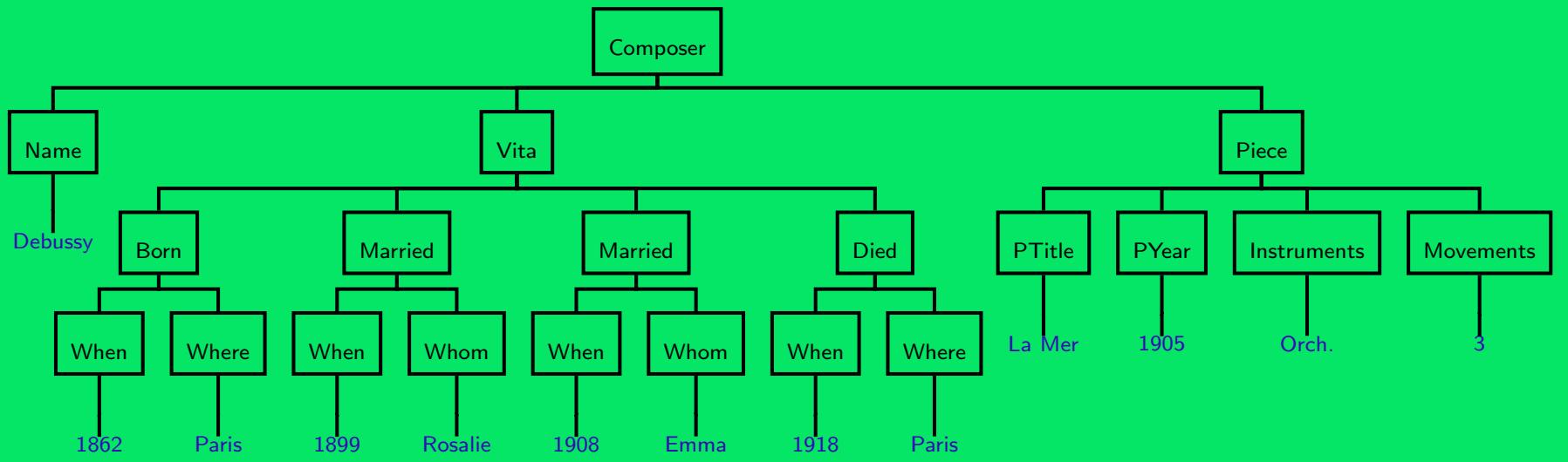
Died $\longrightarrow$ When Where

Piece $\longrightarrow$ PTitle PYear Instr Move

...

Validierung bezüglich DTDs

Beispiel-Baum



Beispiel-DTD

Composers $\longrightarrow$ Composer*

Composer $\longrightarrow$ Name Vita Piece*

Vita $\longrightarrow$ Born Married* Died?

Born $\longrightarrow$ When Where

Married $\longrightarrow$ When Whom

Died $\longrightarrow$ When Where

Piece $\longrightarrow$ PTitle PYear Instr Move

...

Bemerkung

Validierung bzgl. DTD ist einfach:
Für jeden Knoten teste, ob die Folge der Kinder dem zugehörigen regulären Ausdruck entspricht

Inhalt

Ausdrucksschwäche von DTDs

Bemerkung

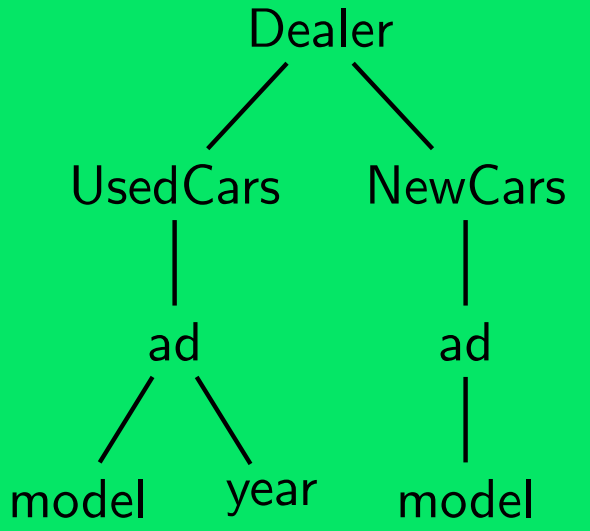
- DTDs haben nur eine sehr eingeschränkte Ausdrucksfähigkeit:
- Elemente mit dem selben Namen können nicht in verschiedenen Zusammenhängen auftreten
- Zuordnung von Elementen zu **Typen** wäre wünschenswert

→ **DTDs mit Typen** (Abstraktion von XML-Schema)

Ein klassisches Beispiel

```
<!DOCTYPE Dealer [
    <!ELEMENT Dealer (UsedCars NewCars)>
    <!ELEMENT UsedCars (ad*)>
    <!ELEMENT NewCars (ad*)>
    <!ELEMENT ad ((model, year) | model)> ]>
```

Intention



DTDs mit Typen

Definition: DTD mit Typen

- Σ: Elementnamen
- Σ': Typen für Elemente
- $\mu : \Sigma' \longrightarrow \Sigma$: Zuordnung von Typen zu Elementnamen
- d: (einfache) DTD über Typen

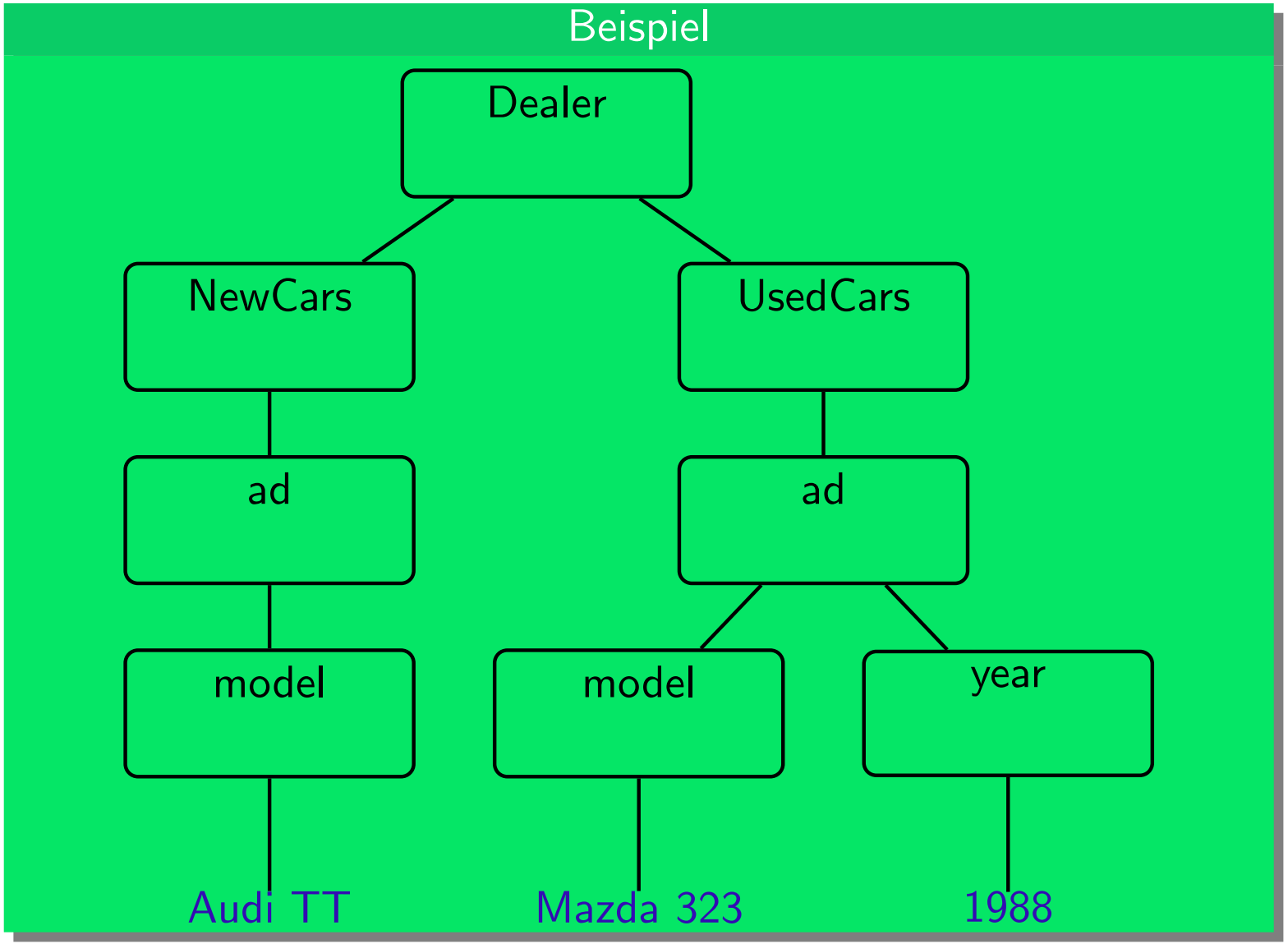
Beispiel

Dealer $\longrightarrow$ UsedCars NewCars	μ(Dealer) = Dealer
UsedCars $\longrightarrow$ adUsed*	μ(UsedCars) = UsedCars
NewCars $\longrightarrow$ adNew*	μ(NewCars) = NewCars
adUsed $\longrightarrow$ model year	μ(adUsed) = ad
adNew $\longrightarrow$ model	μ(adNew) = ad

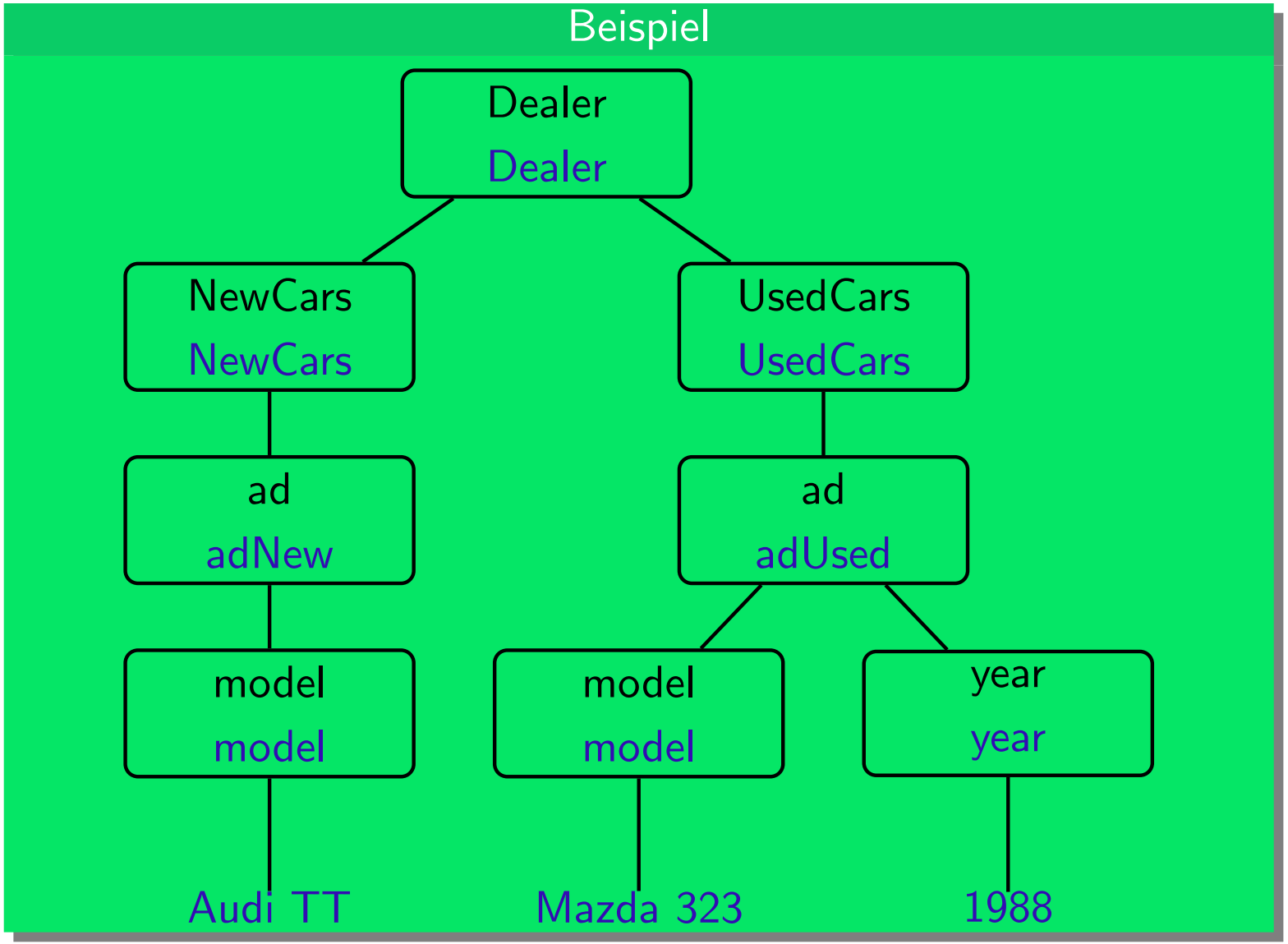
Definition

Ein Baum **erfüllt** eine DTD mit Typen, wenn es eine konsistente Zuordnung seiner Knoten zu Typen gibt

Beispiel

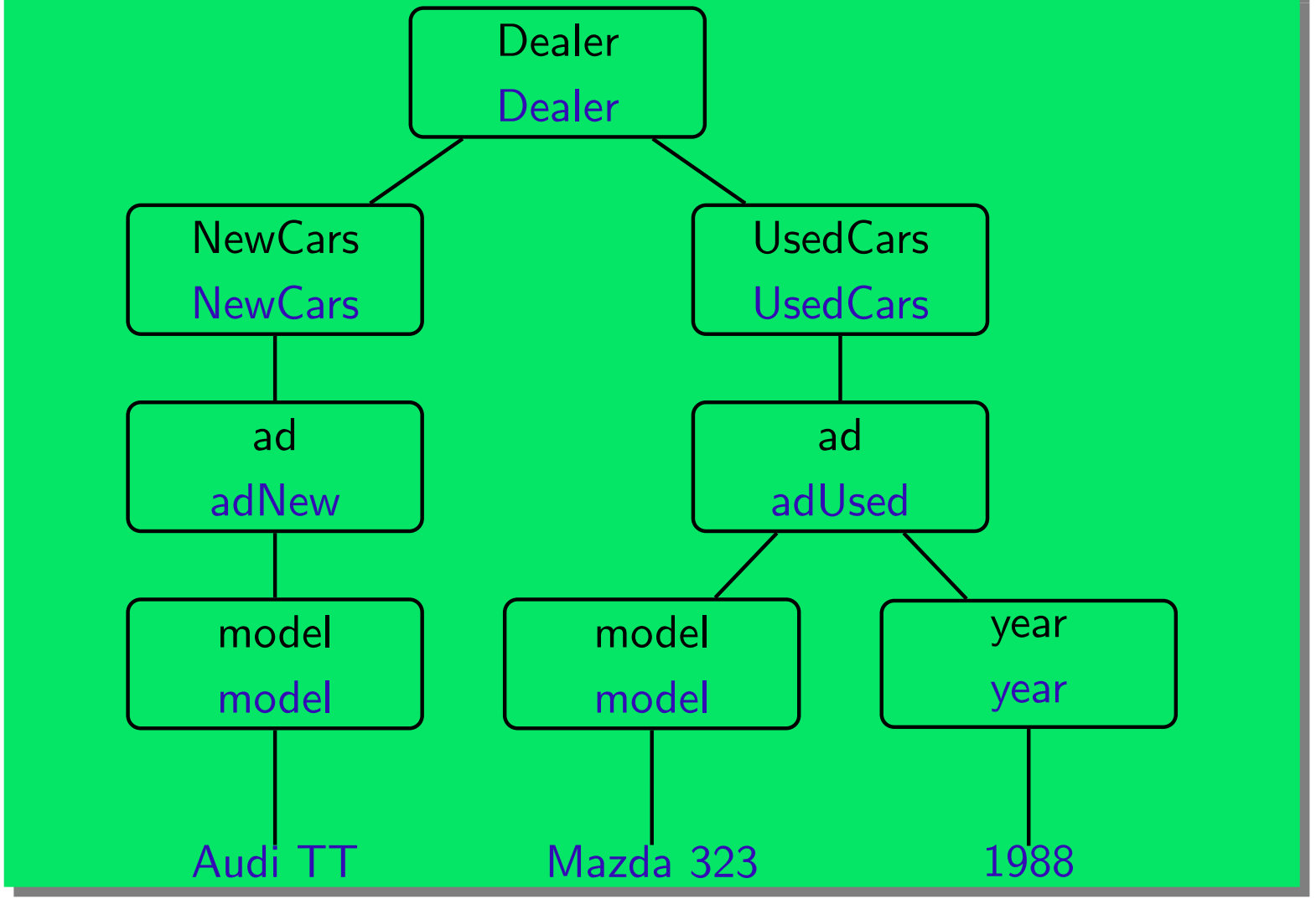

Beispiel
Dealer
NewCars
UsedCars
ad
ad
model
model
year
Audi TT
Mazda 323
1988

Beispiel


Beispiel
Dealer
Dealer
NewCars
NewCars
UsedCars
UsedCars
ad
adNew
ad
adUsed
model
model
model
model
year
year
Audi TT
Mazda 323
1988

Beispiel

Beispiel



Bemerkung

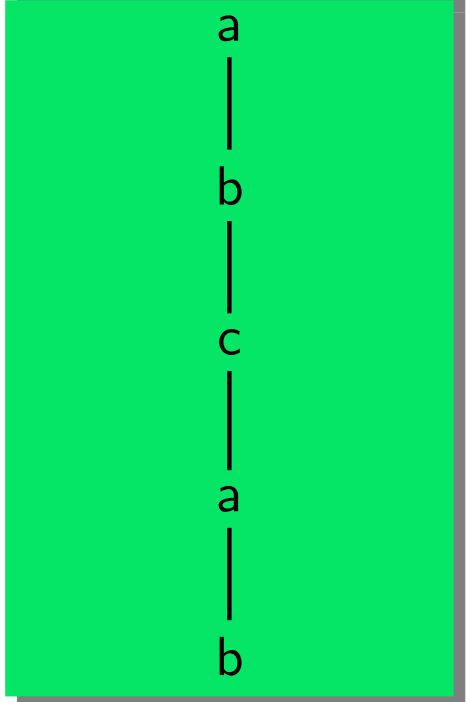
DTDs mit Typen lassen sich als Baumautomaten auffassen

Von Strings zu Bäumen

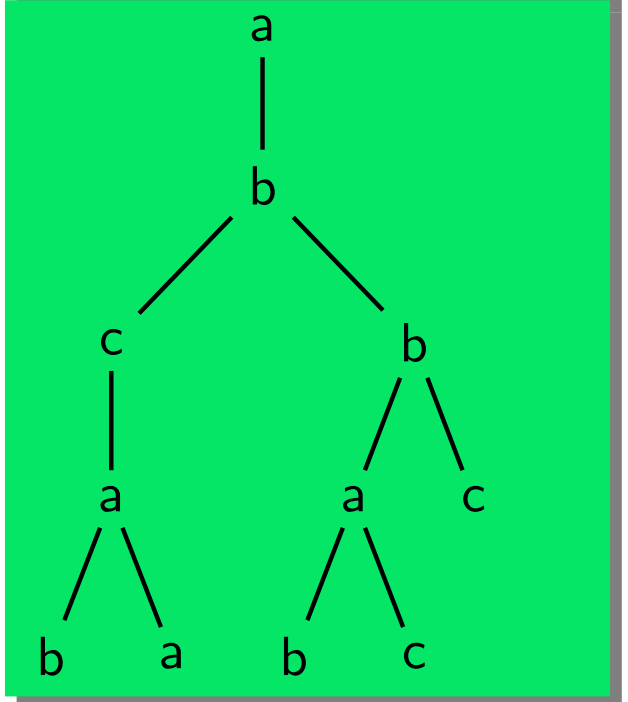
Ein String

abcab

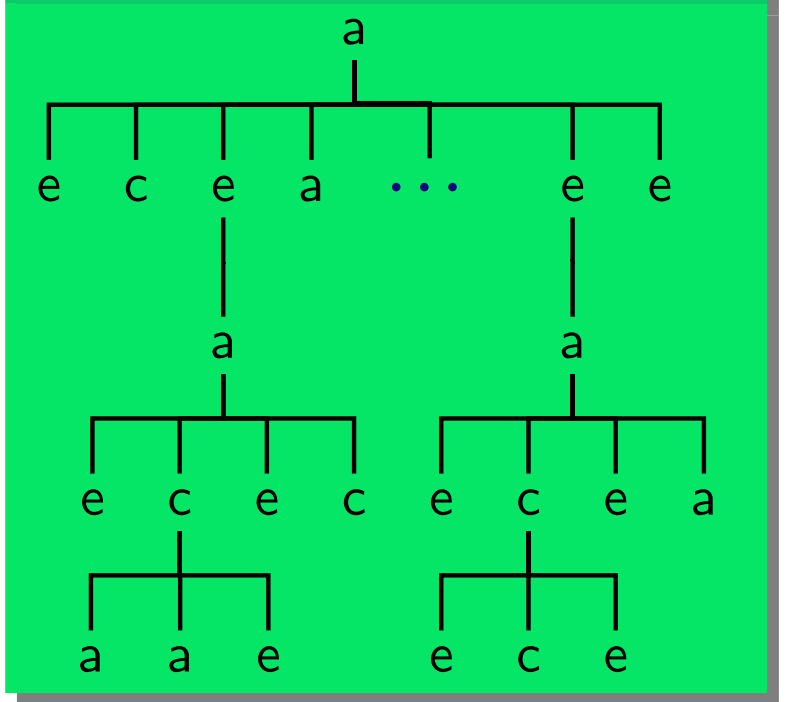
String als Baum



Binärer Baum



Unbeschränkter Baum

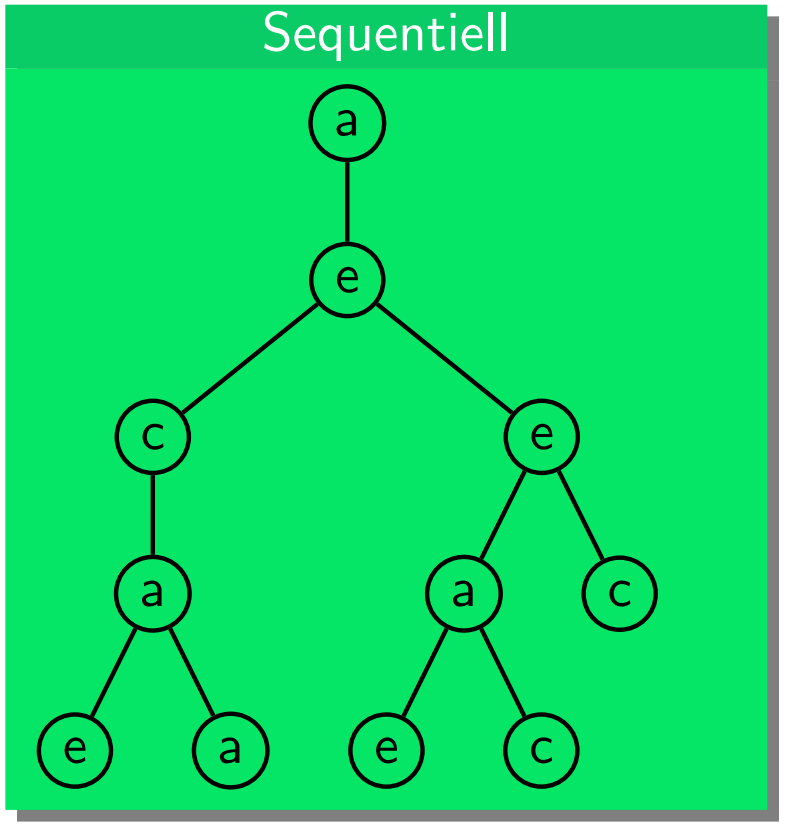


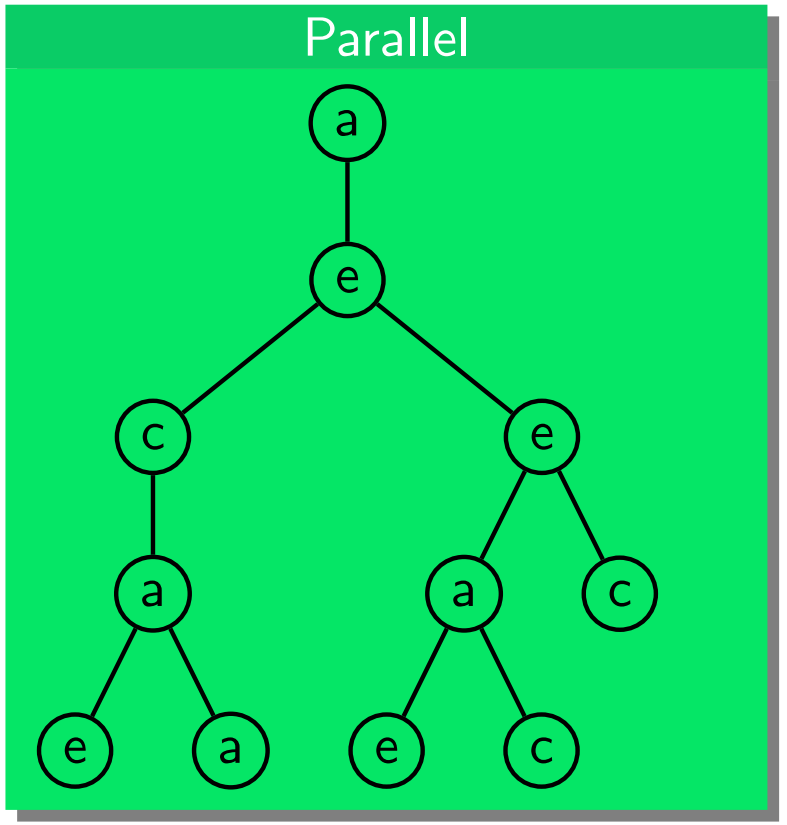
XML und Bäume

- XML-Bäume haben unbeschränkten Knotengrad
- Baumautomaten wurden zuerst für binäre Bäume betrachtet
- Die meisten wichtigen Ideen wurden dabei schon entwickelt

Von String-Automaten zu Baum-Automaten

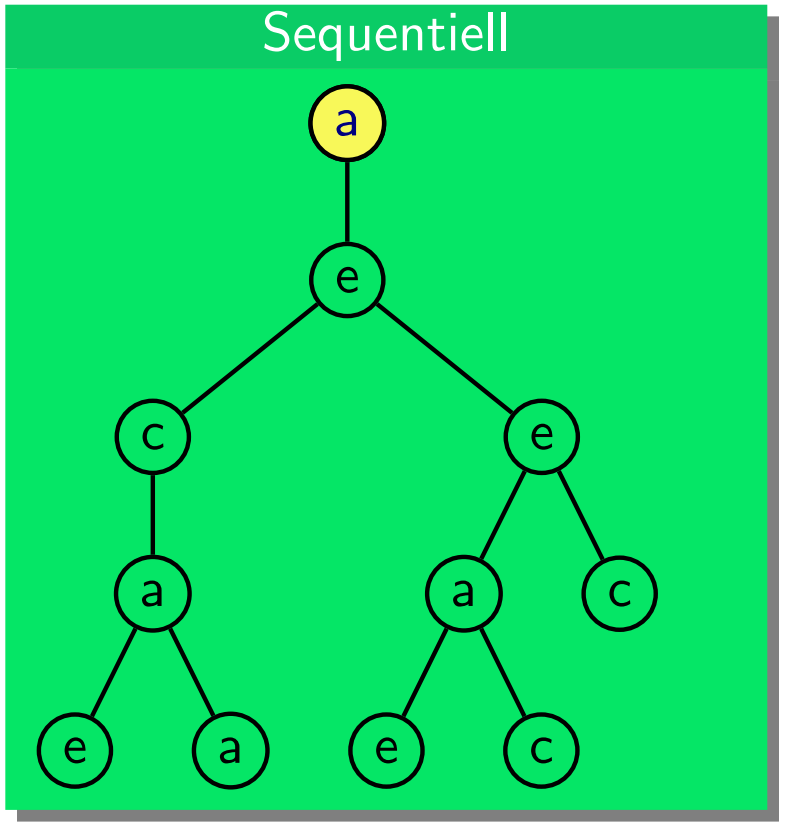
Frage: Wie lassen sich Automaten auf Bäumen definieren?

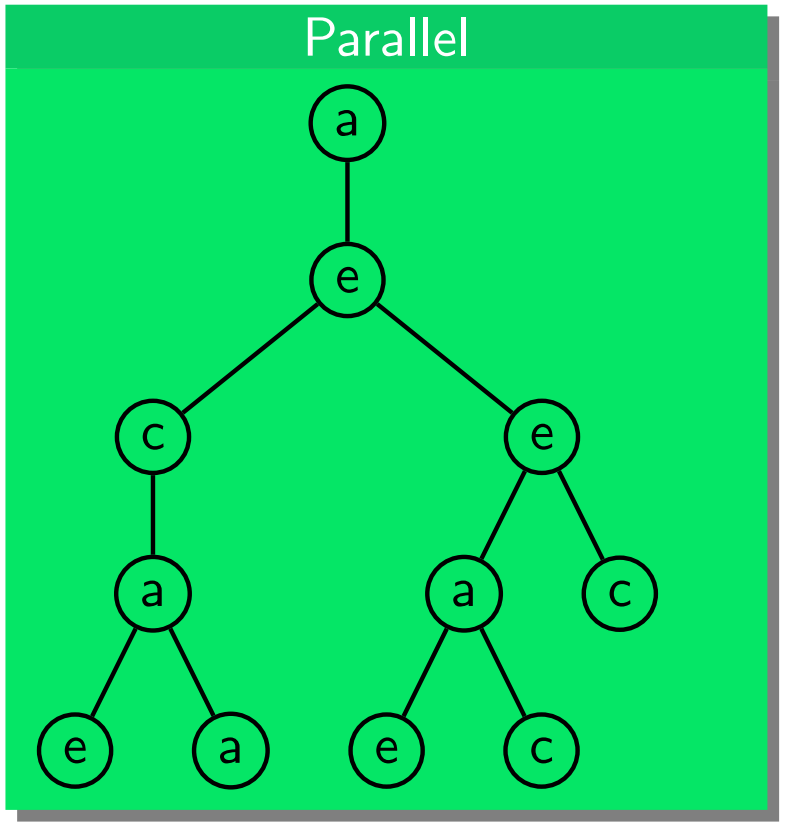




Von String-Automaten zu Baum-Automaten

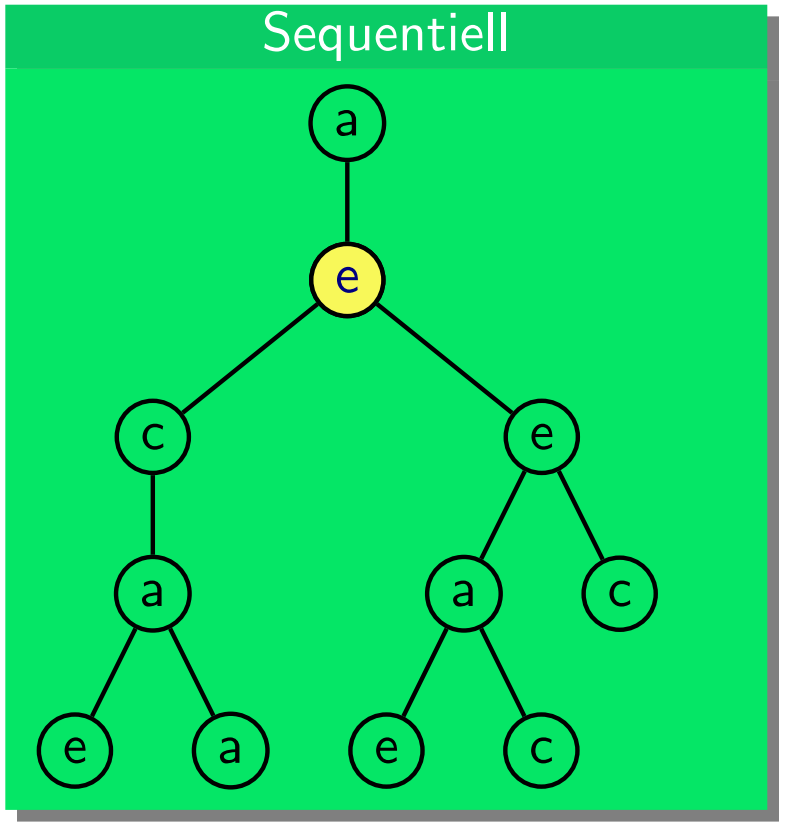
Frage: Wie lassen sich Automaten auf Bäumen definieren?

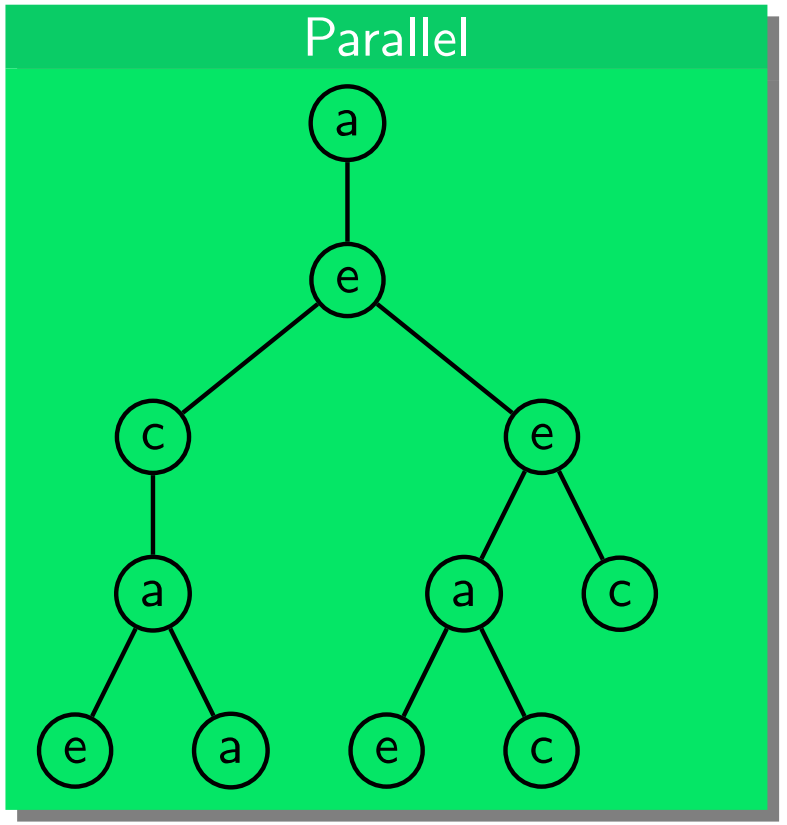




Von String-Automaten zu Baum-Automaten

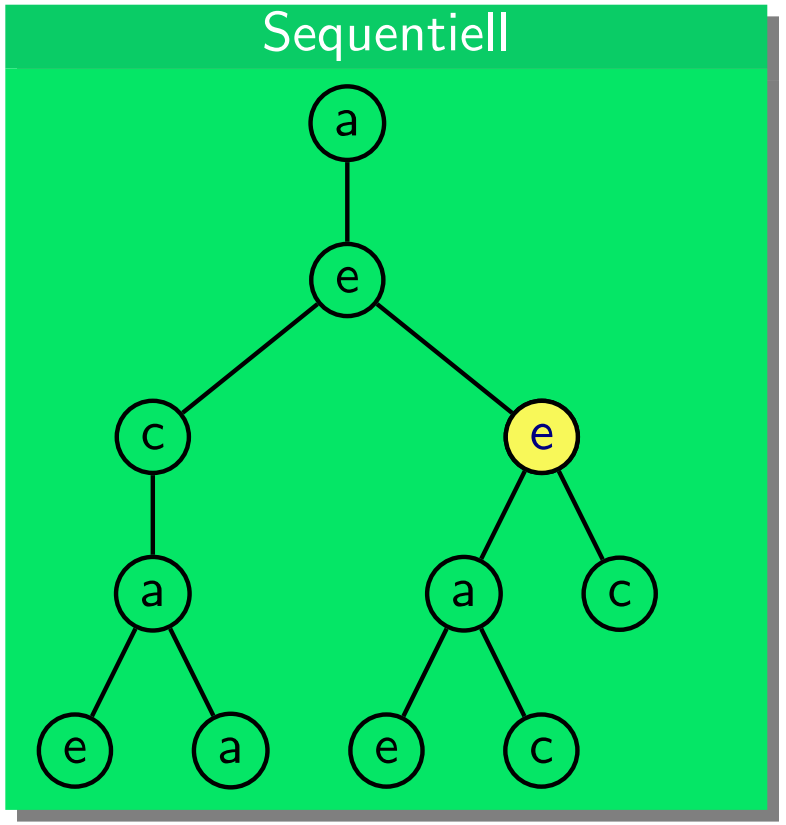
Frage: Wie lassen sich Automaten auf Bäumen definieren?

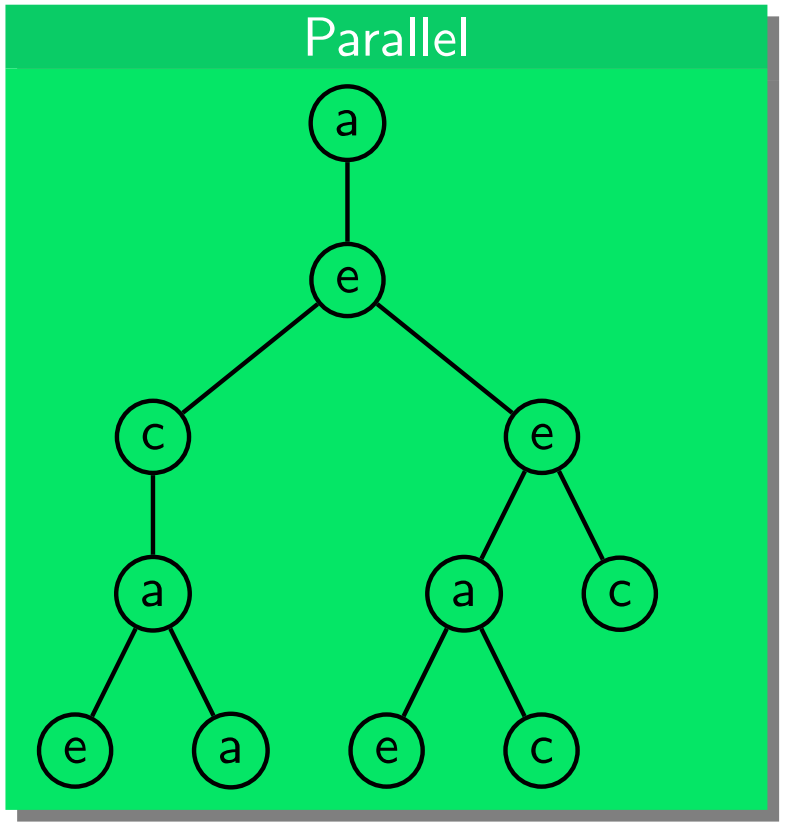




Von String-Automaten zu Baum-Automaten

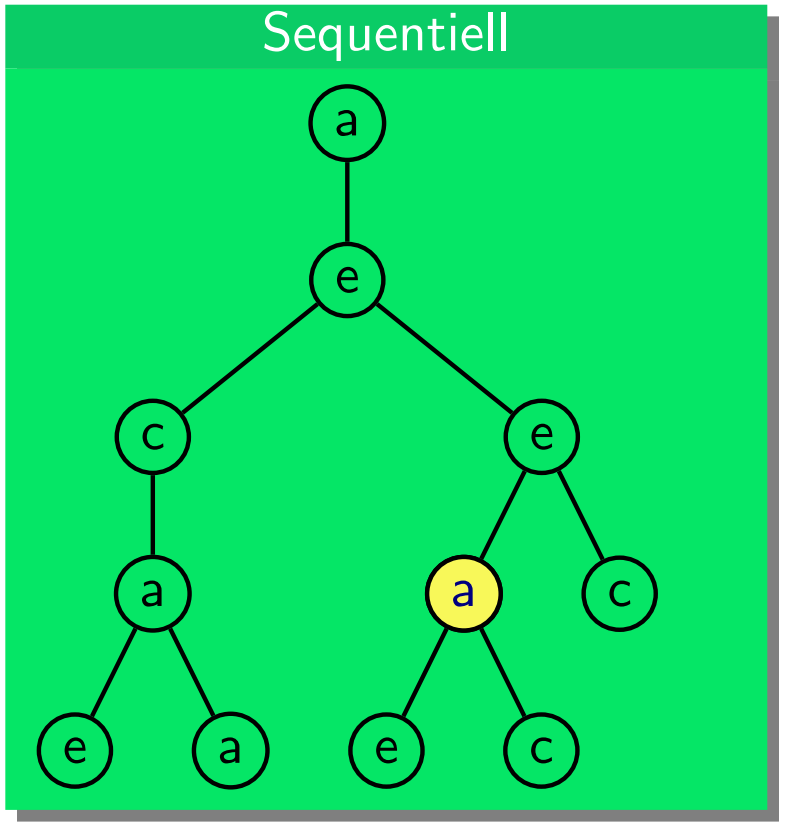
Frage: Wie lassen sich Automaten auf Bäumen definieren?

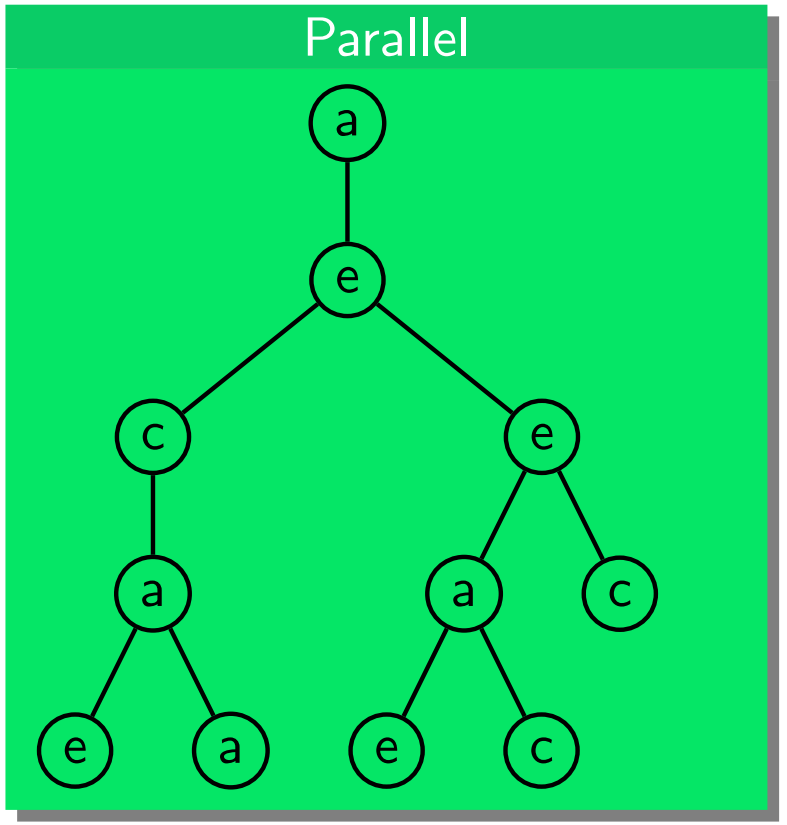




Von String-Automaten zu Baum-Automaten

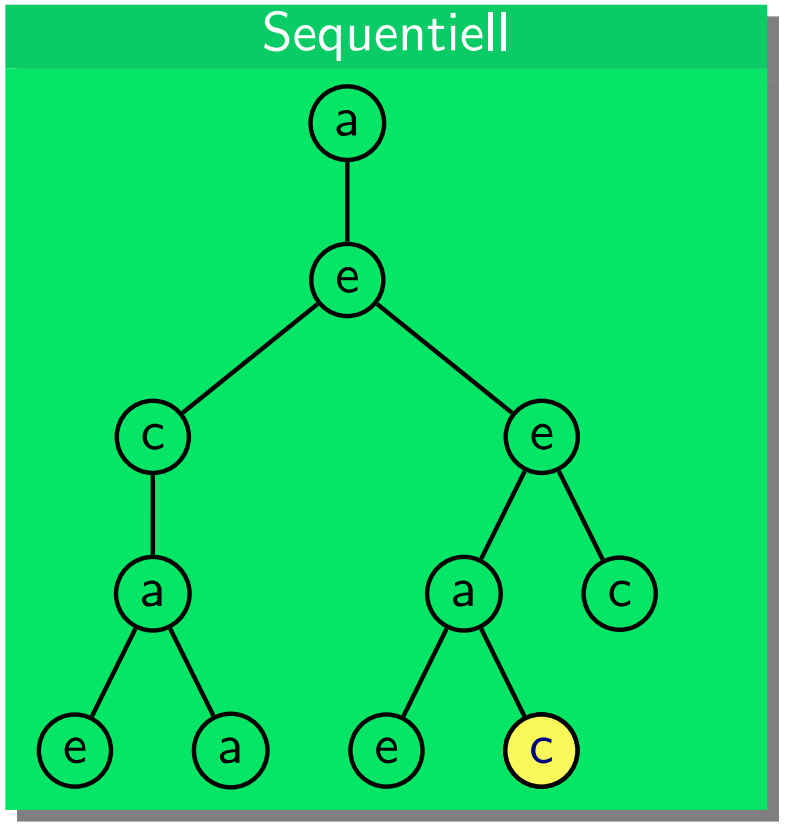
Frage: Wie lassen sich Automaten auf Bäumen definieren?

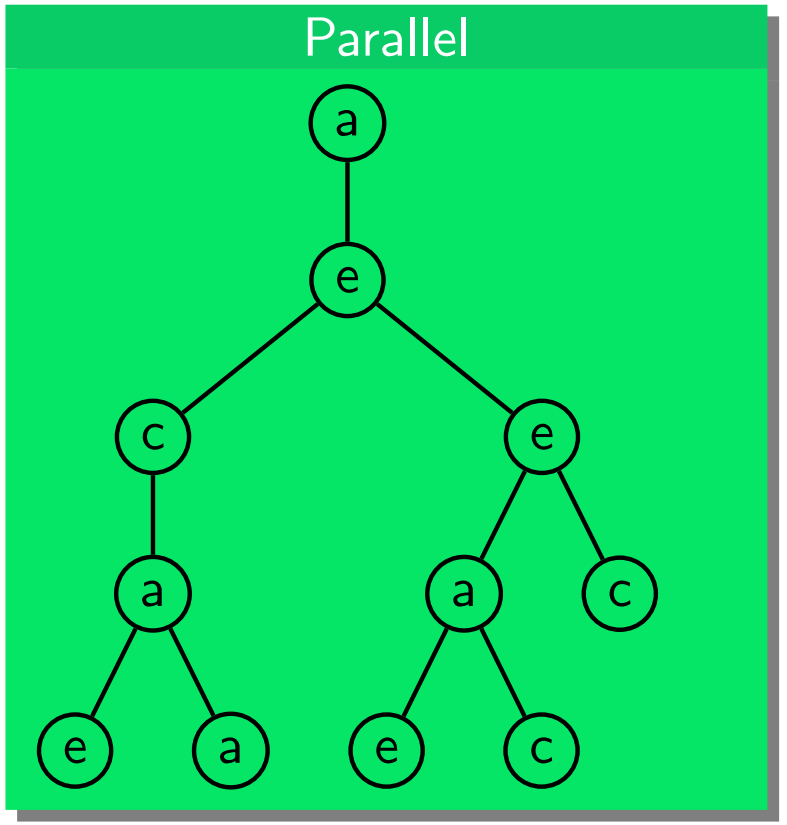




Von String-Automaten zu Baum-Automaten

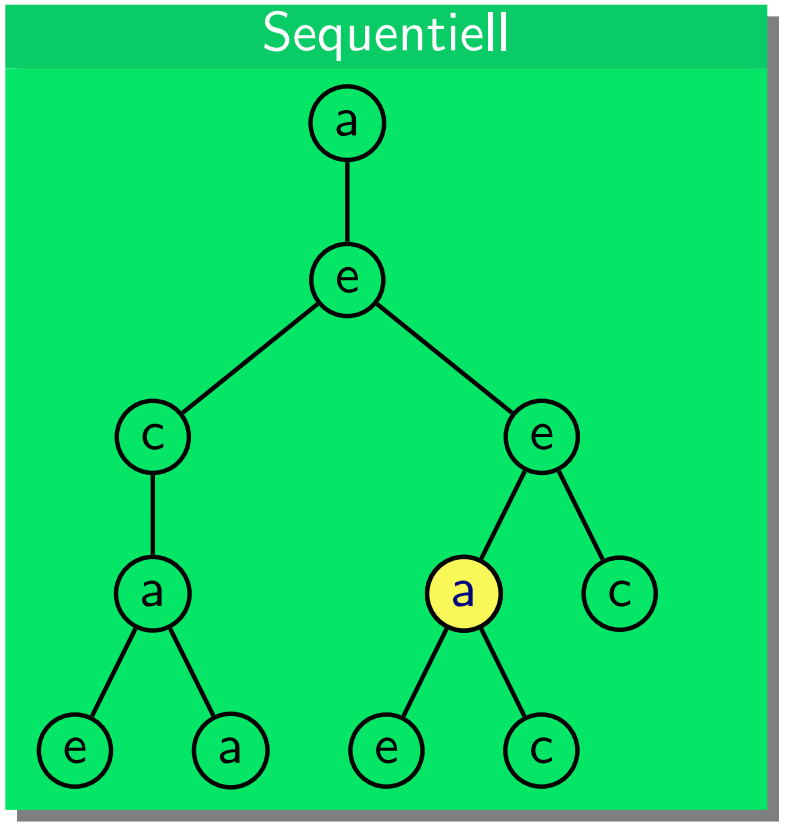
Frage: Wie lassen sich Automaten auf Bäumen definieren?

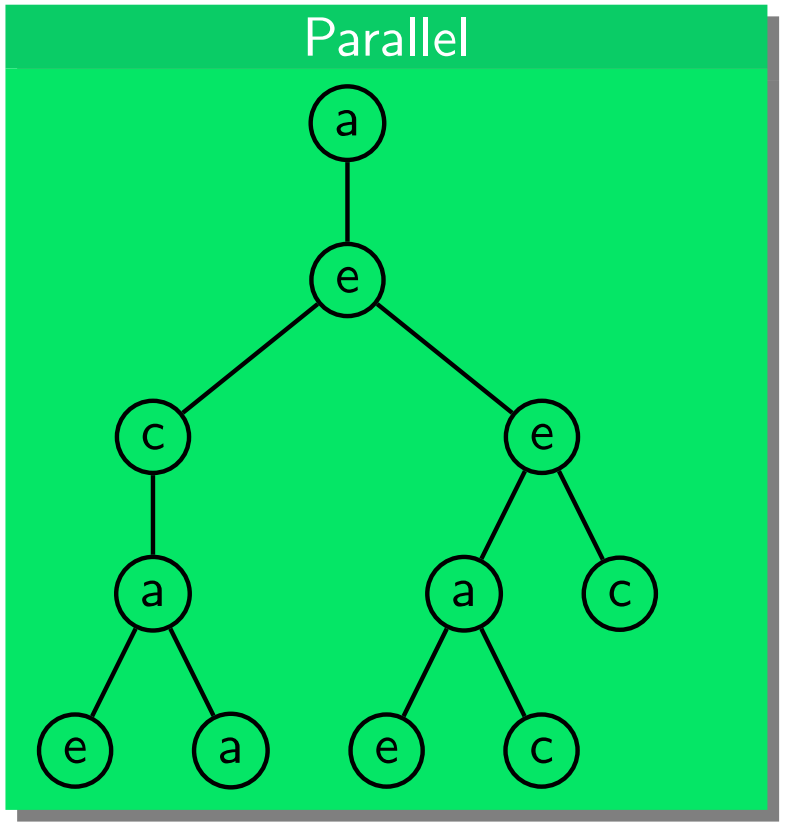




Von String-Automaten zu Baum-Automaten

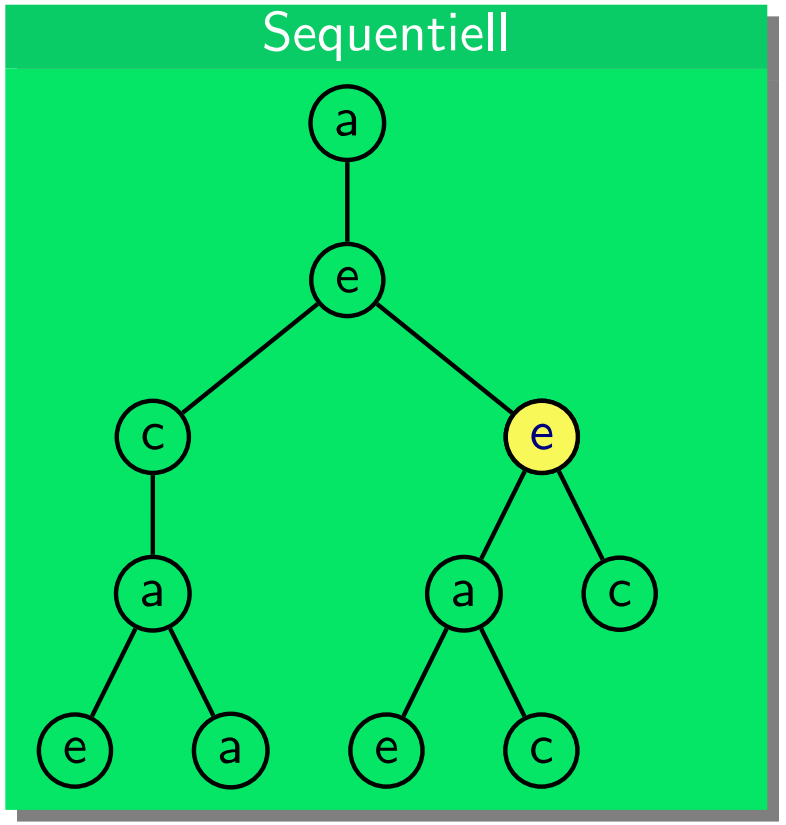
Frage: Wie lassen sich Automaten auf Bäumen definieren?

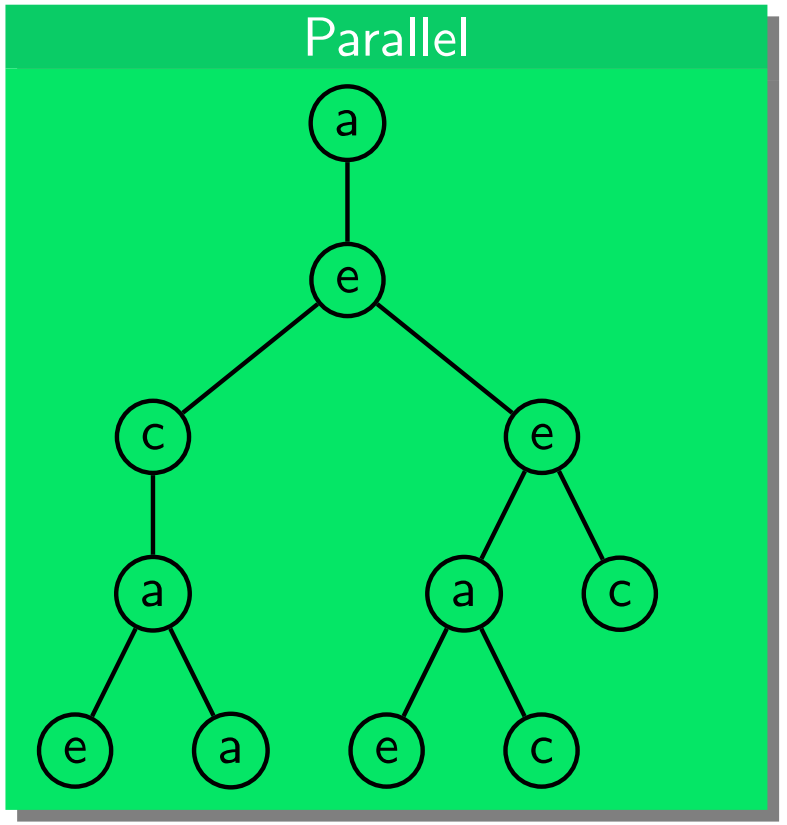




Von String-Automaten zu Baum-Automaten

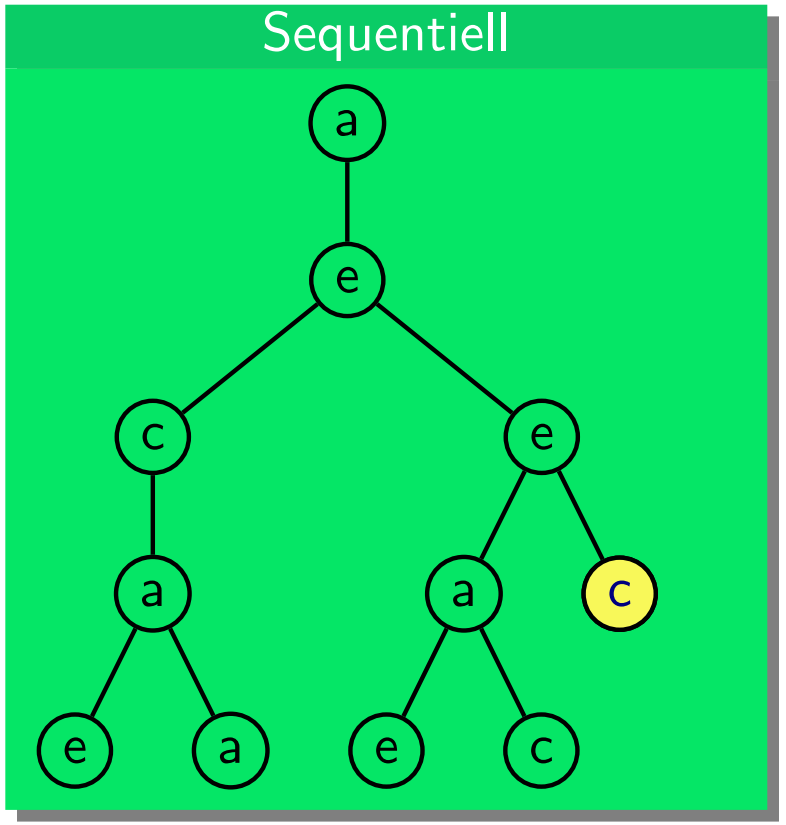
Frage: Wie lassen sich Automaten auf Bäumen definieren?

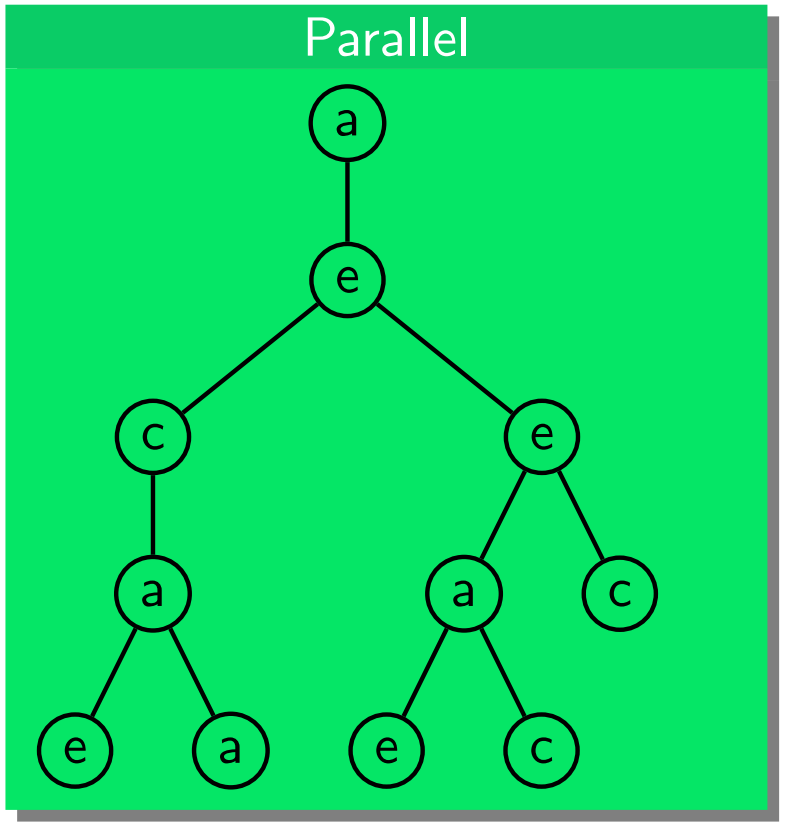




Von String-Automaten zu Baum-Automaten

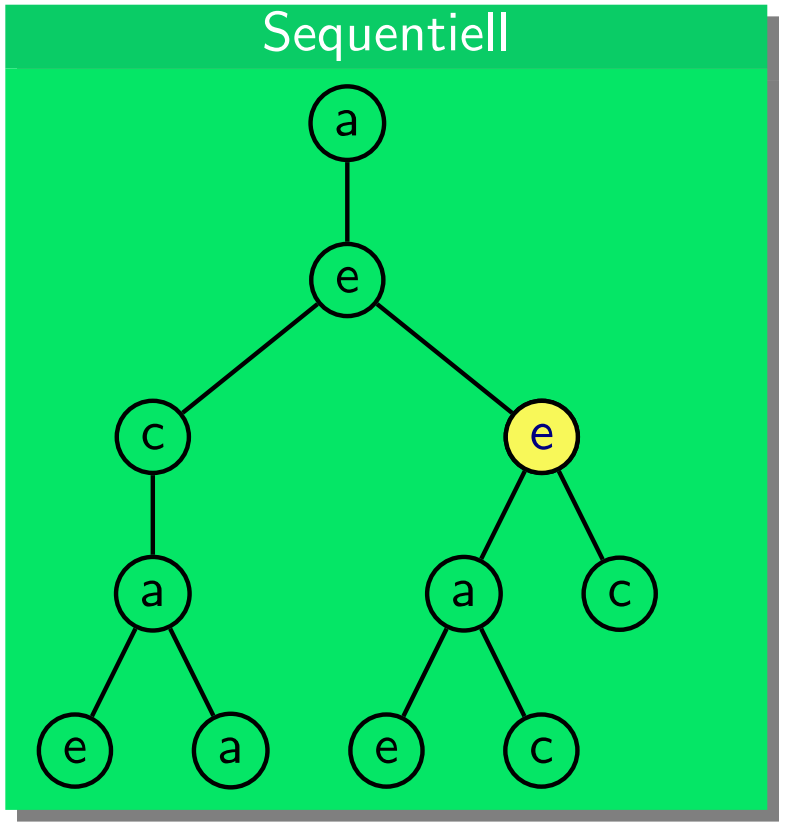
Frage: Wie lassen sich Automaten auf Bäumen definieren?

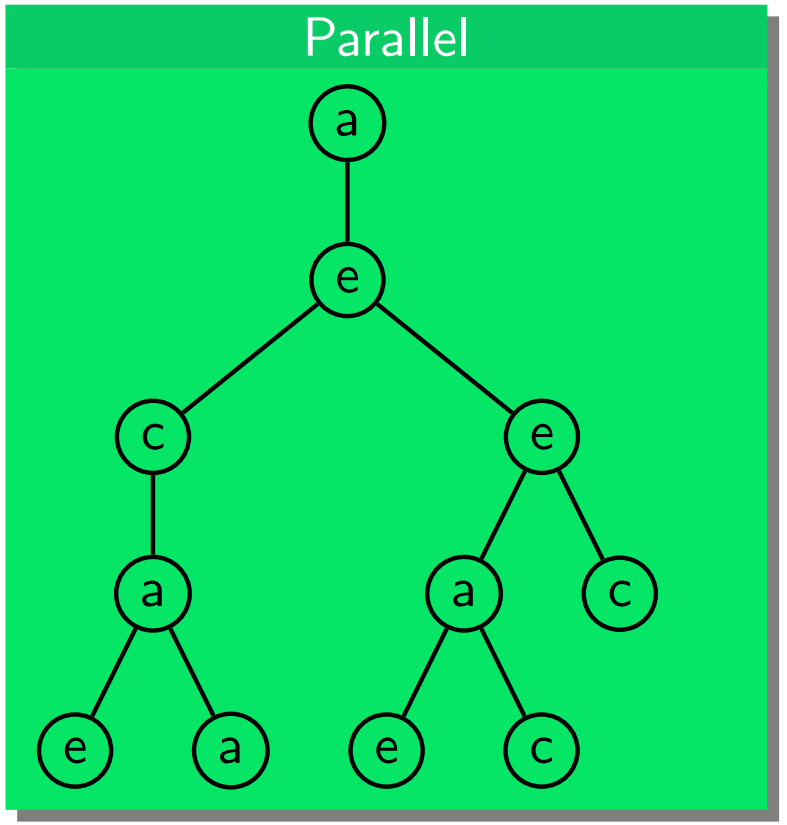




Von String-Automaten zu Baum-Automaten

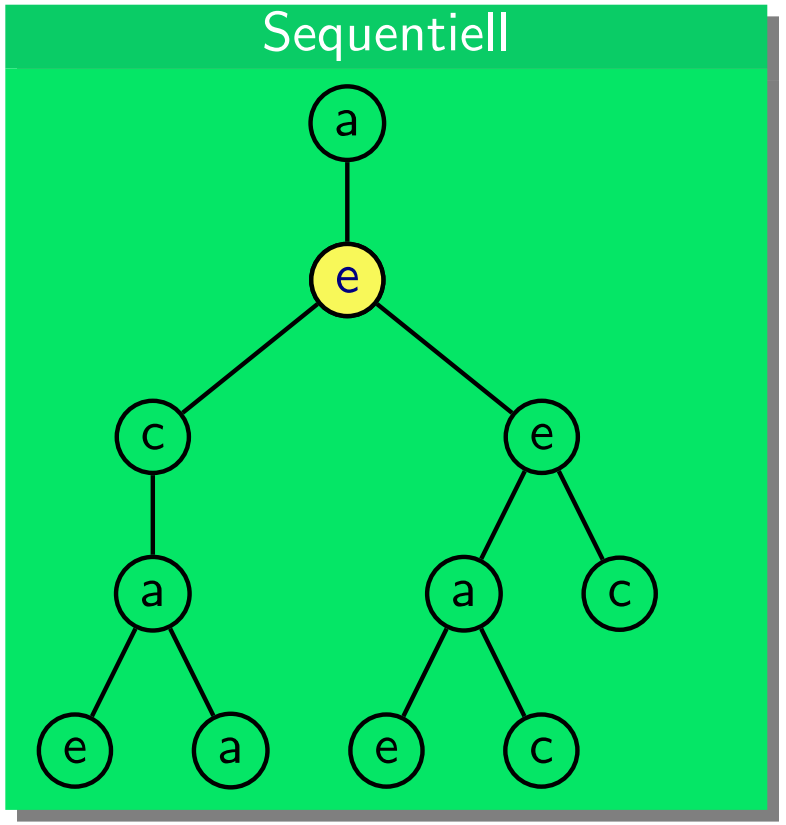
Frage: Wie lassen sich Automaten auf Bäumen definieren?

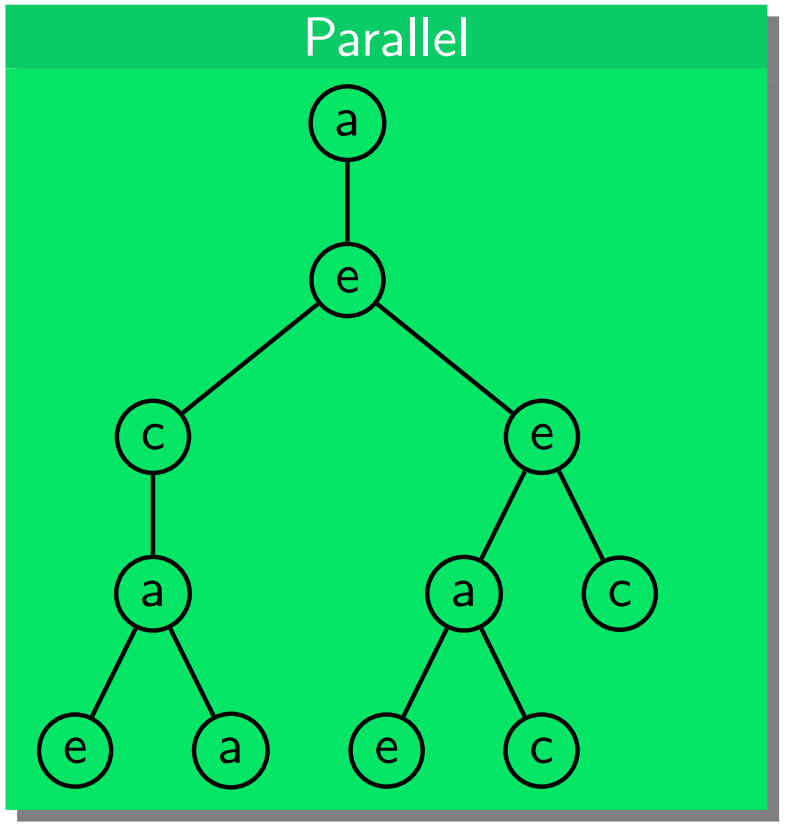




Von String-Automaten zu Baum-Automaten

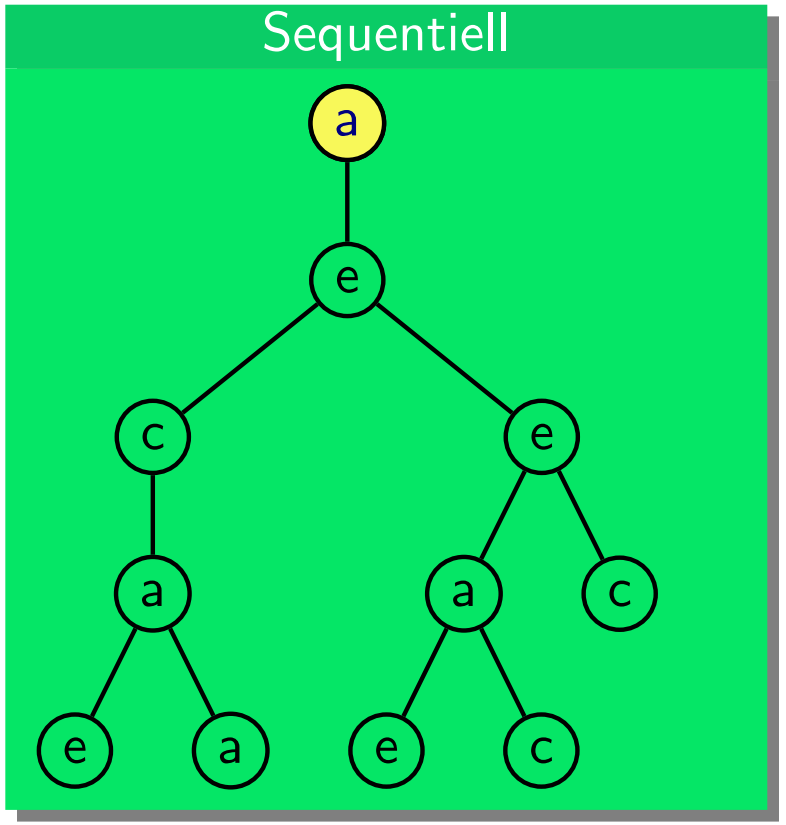
Frage: Wie lassen sich Automaten auf Bäumen definieren?

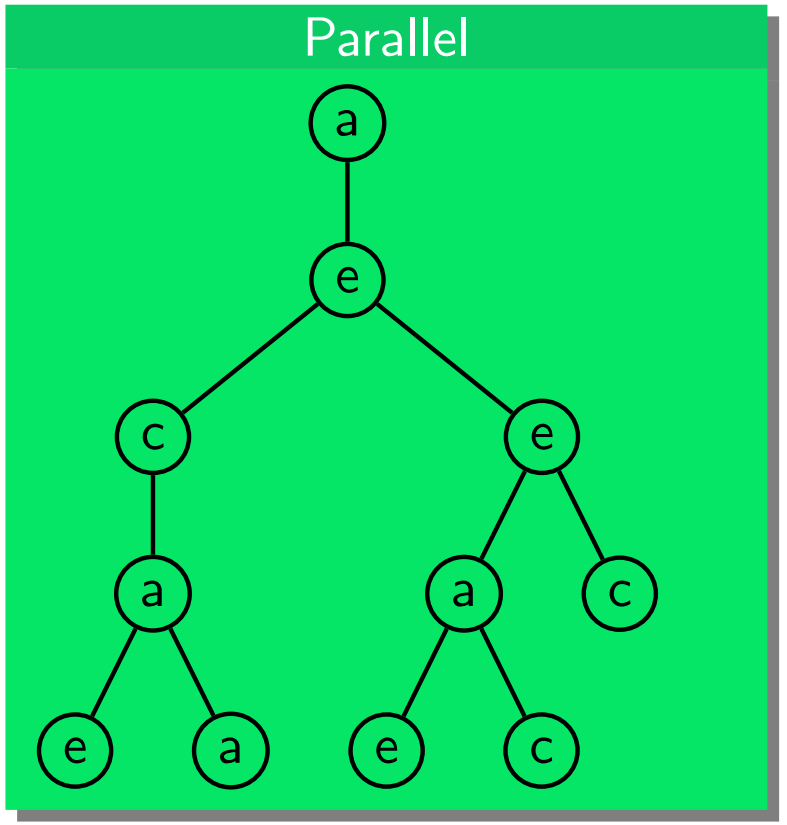




Von String-Automaten zu Baum-Automaten

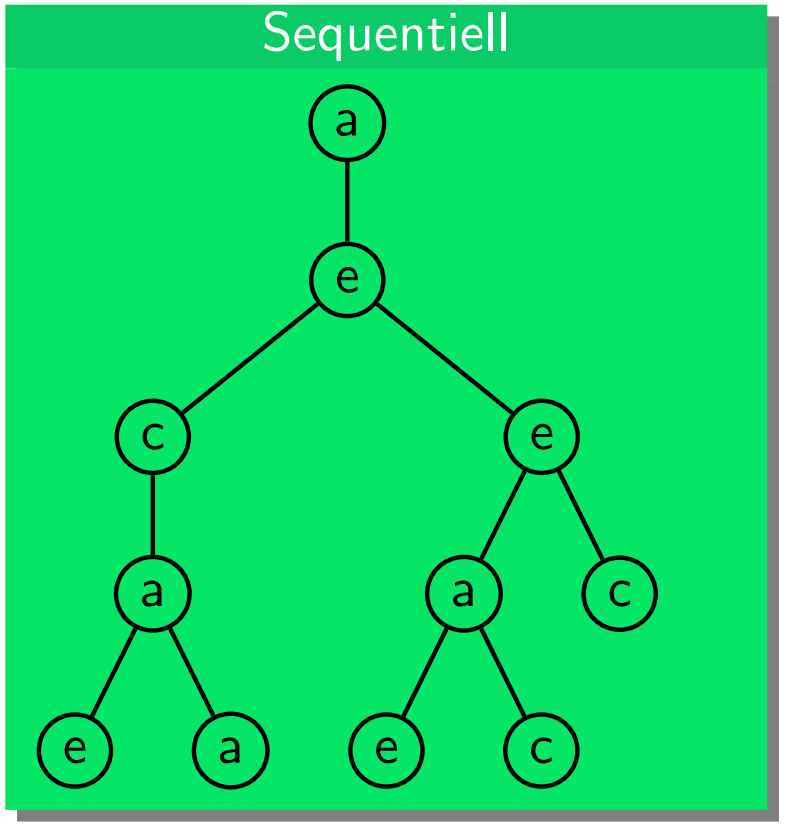
Frage: Wie lassen sich Automaten auf Bäumen definieren?

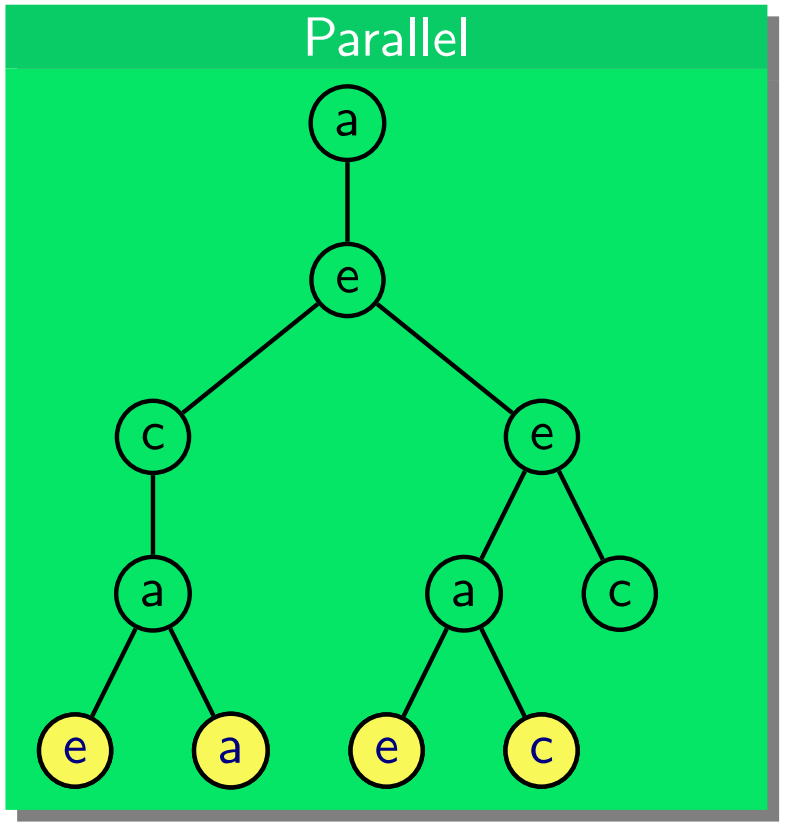




Von String-Automaten zu Baum-Automaten

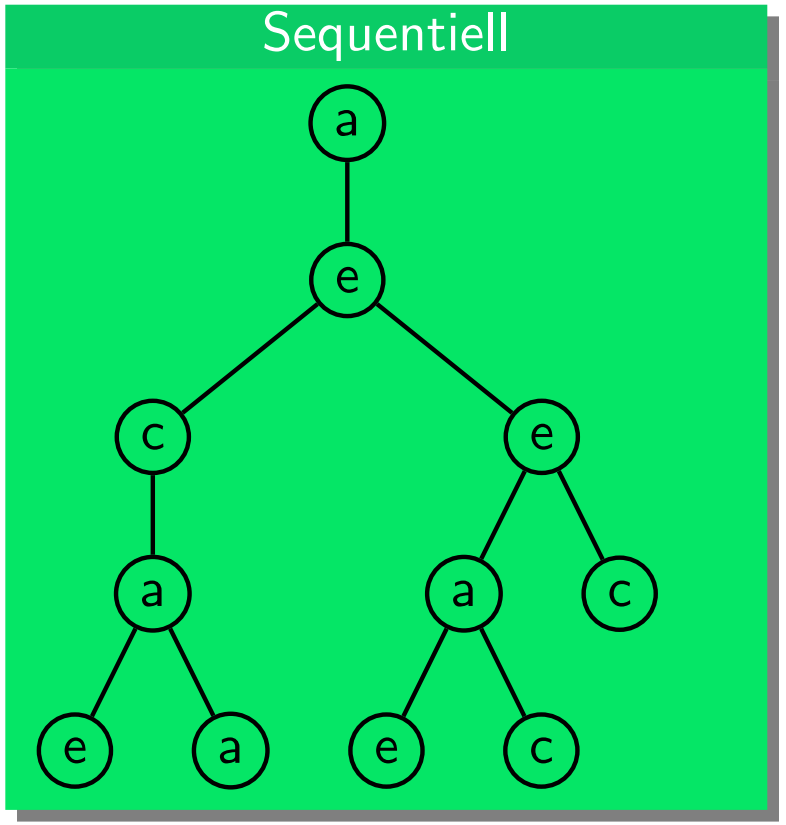
Frage: Wie lassen sich Automaten auf Bäumen definieren?

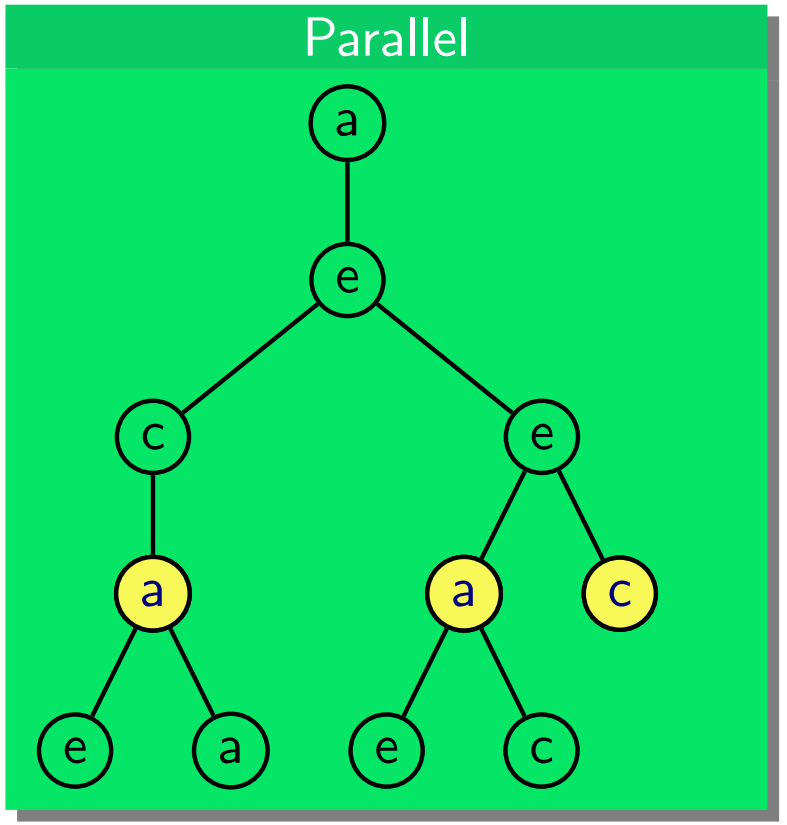




Von String-Automaten zu Baum-Automaten

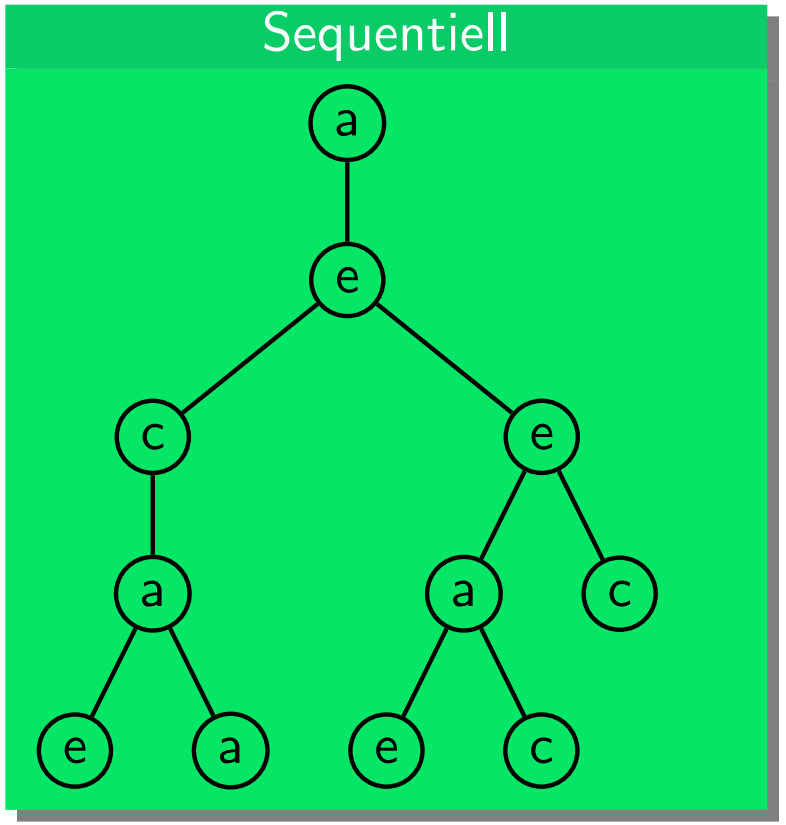
Frage: Wie lassen sich Automaten auf Bäumen definieren?

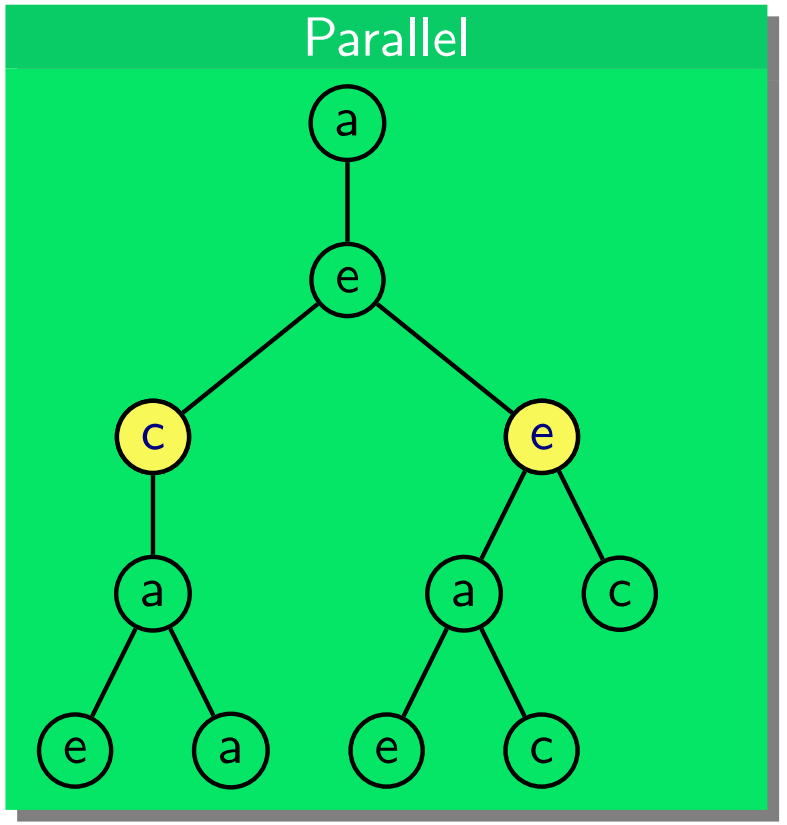




Von String-Automaten zu Baum-Automaten

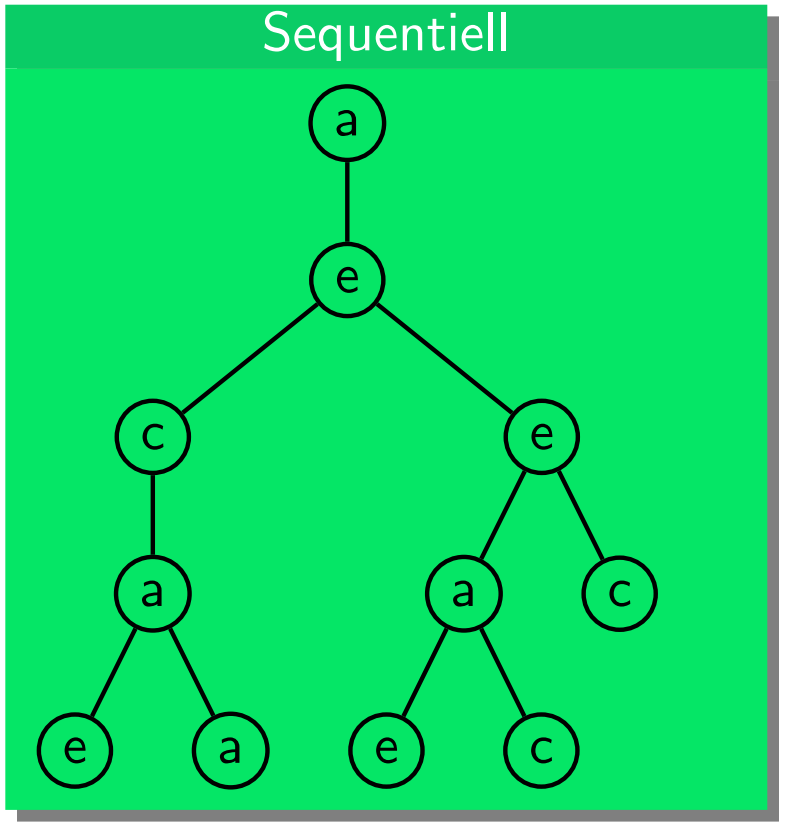
Frage: Wie lassen sich Automaten auf Bäumen definieren?

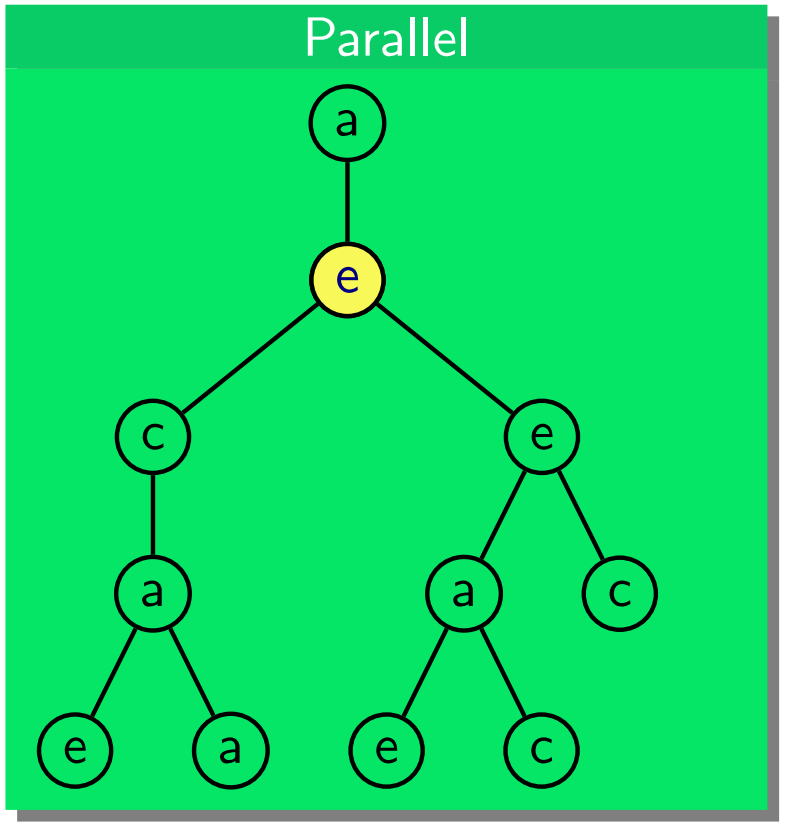




Von String-Automaten zu Baum-Automaten

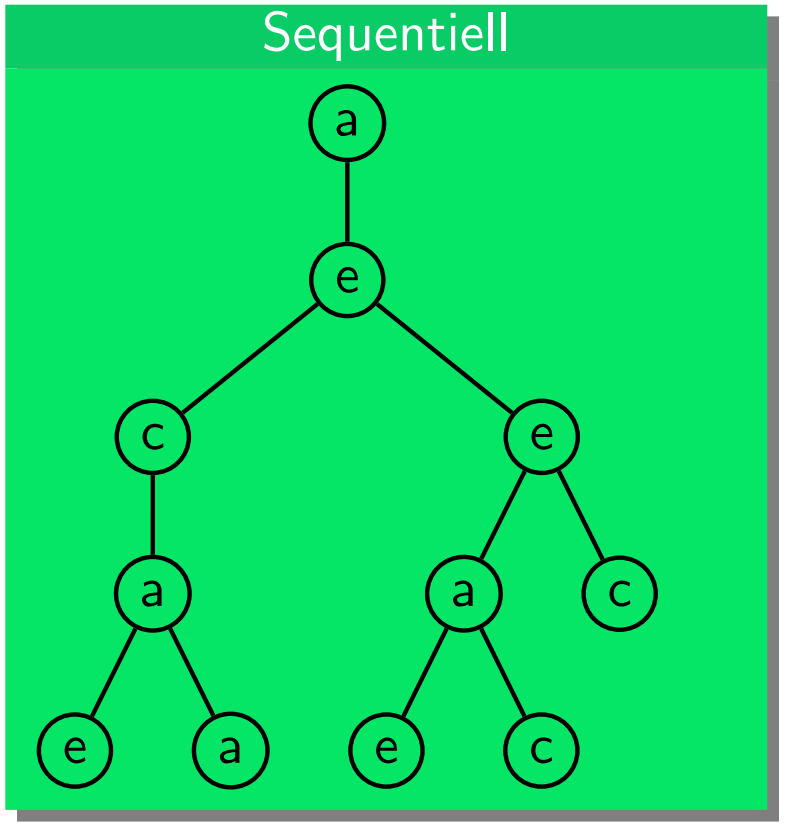
Frage: Wie lassen sich Automaten auf Bäumen definieren?

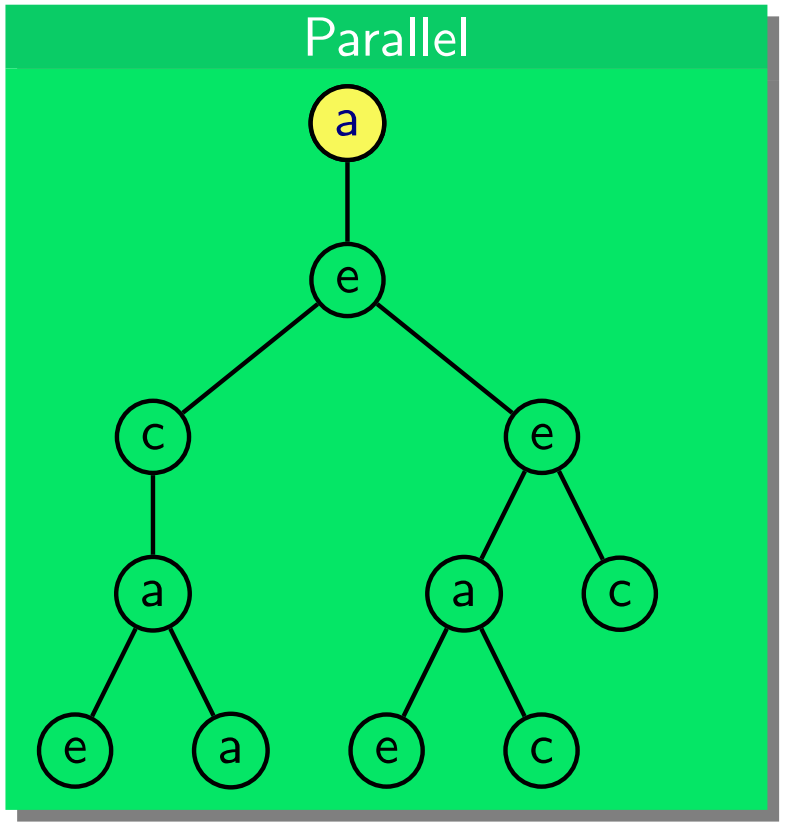




Von String-Automaten zu Baum-Automaten

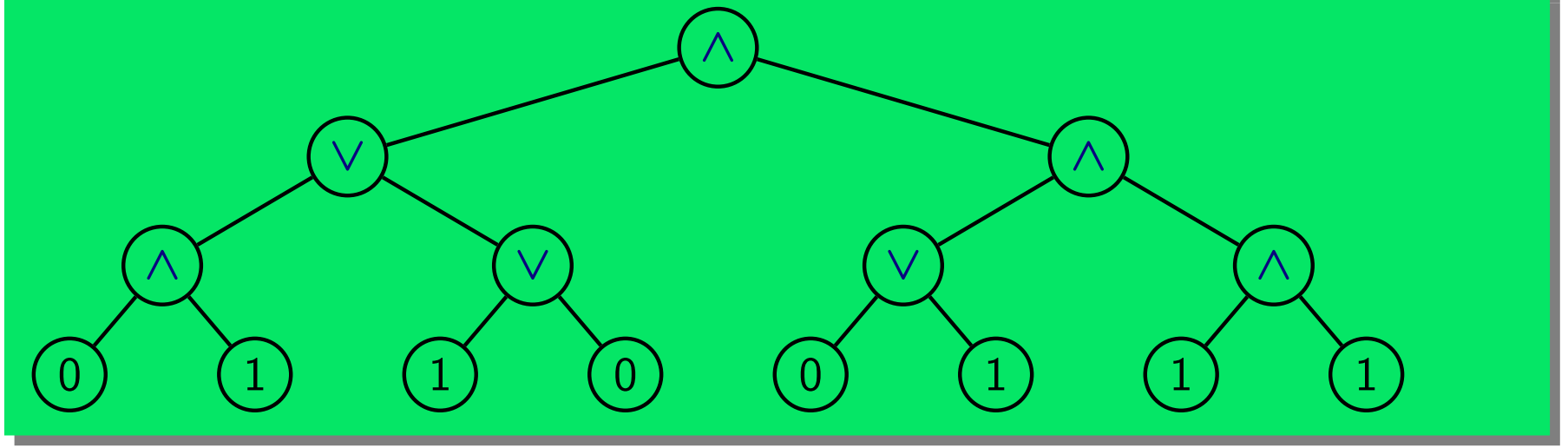
Frage: Wie lassen sich Automaten auf Bäumen definieren?





Bottom-Up Automaten

Beispiel: Auswertung baumartiger Schaltkreise



Idee

- Zwei Zustände: q_0, q_1
- Akzeptierend an der Wurzel: q_1

Transitionen

$\delta(\wedge, q_1) = \{(q_1, q_1)\}$

$\delta(\wedge, q_0) = \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\}$

$\delta(\vee, q_1) = \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\}$

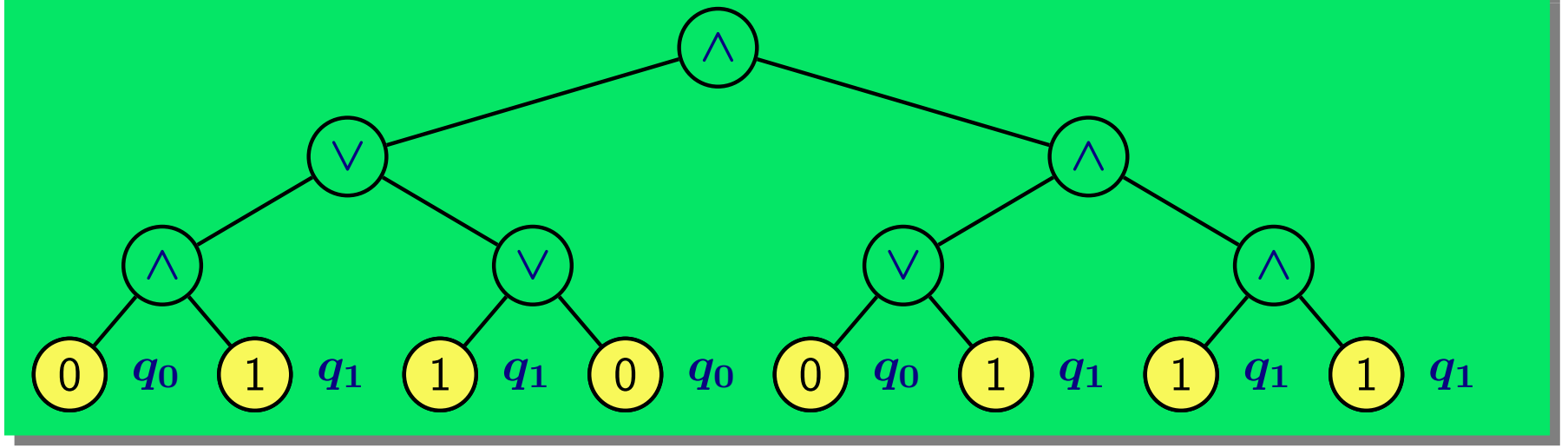
$\delta(\vee, q_0) = \{(q_0, q_0)\}$

$\delta(0, q_0) = \{\epsilon\}; \delta(0, q_1) = \emptyset$

$\delta(1, q_1) = \{\epsilon\}; \delta(1, q_0) = \emptyset$

Bottom-Up Automaten

Beispiel: Auswertung baumartiger Schaltkreise



Idee

- Zwei Zustände: q_0, q_1
- Akzeptierend an der Wurzel: q_1

Transitionen

$\delta(\wedge, q_1) = \{(q_1, q_1)\}$

$\delta(\wedge, q_0) = \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\}$

$\delta(\vee, q_1) = \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\}$

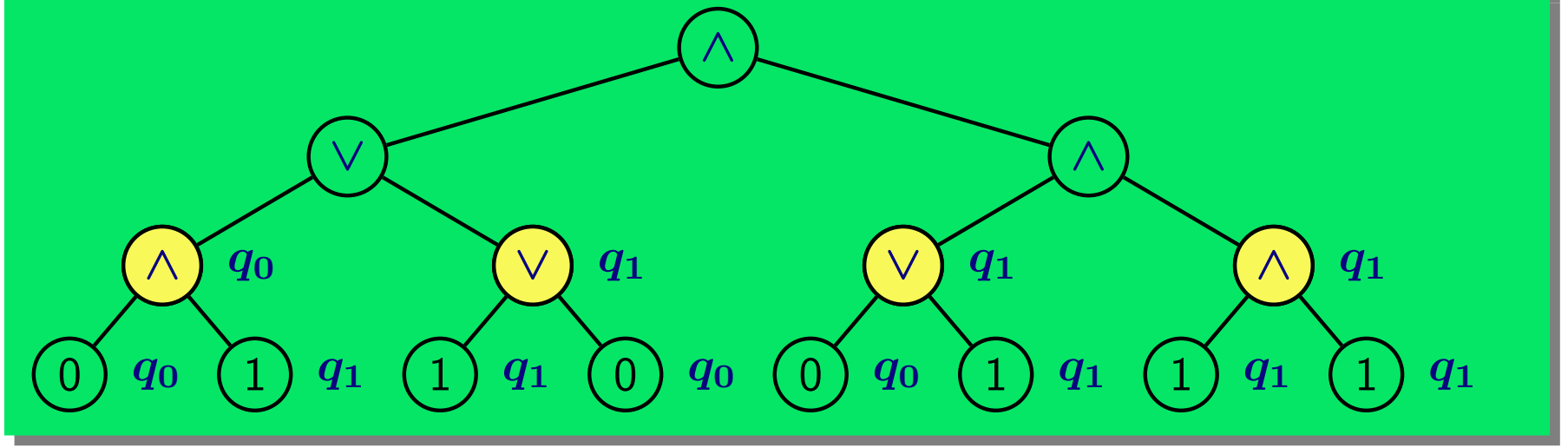
$\delta(\vee, q_0) = \{(q_0, q_0)\}$

$\delta(0, q_0) = \{\epsilon\}; \delta(0, q_1) = \emptyset$

$\delta(1, q_1) = \{\epsilon\}; \delta(1, q_0) = \emptyset$

Bottom-Up Automaten

Beispiel: Auswertung baumartiger Schaltkreise



Idee

- Zwei Zustände: q_0, q_1
- Akzeptierend an der Wurzel: q_1

Transitionen

$\delta(\wedge, q_1) = \{(q_1, q_1)\}$

$\delta(\wedge, q_0) = \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\}$

$\delta(\vee, q_1) = \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\}$

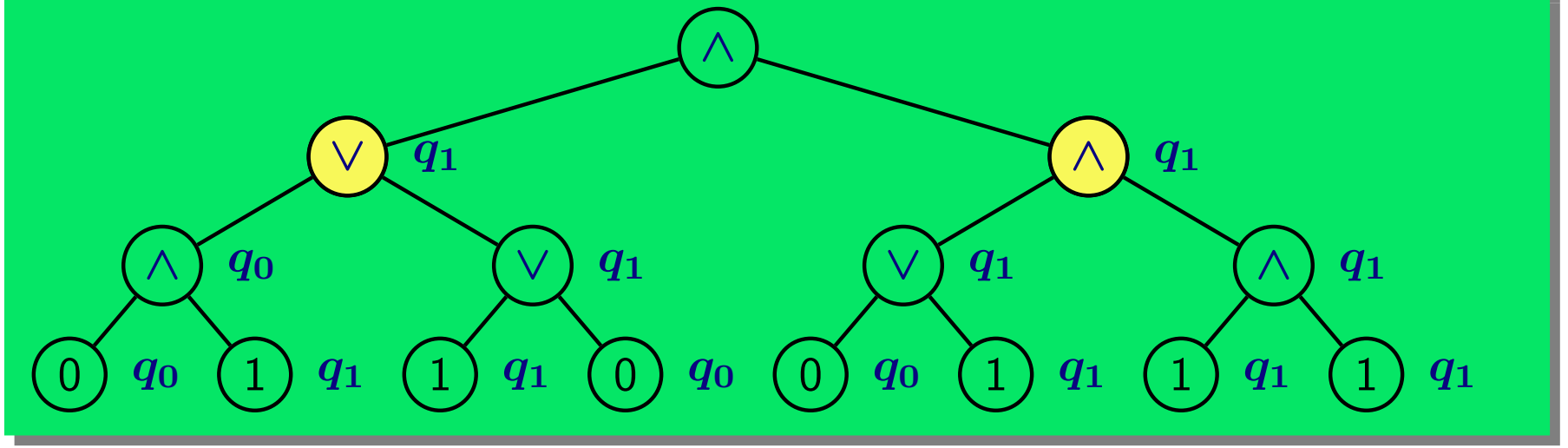
$\delta(\vee, q_0) = \{(q_0, q_0)\}$

$\delta(0, q_0) = \{\epsilon\}; \delta(0, q_1) = \emptyset$

$\delta(1, q_1) = \{\epsilon\}; \delta(1, q_0) = \emptyset$

Bottom-Up Automaten

Beispiel: Auswertung baumartiger Schaltkreise



Idee

- Zwei Zustände: q_0, q_1
- Akzeptierend an der Wurzel: q_1

Transitionen

$\delta(\wedge, q_1) = \{(q_1, q_1)\}$

$\delta(\wedge, q_0) = \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\}$

$\delta(\vee, q_1) = \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\}$

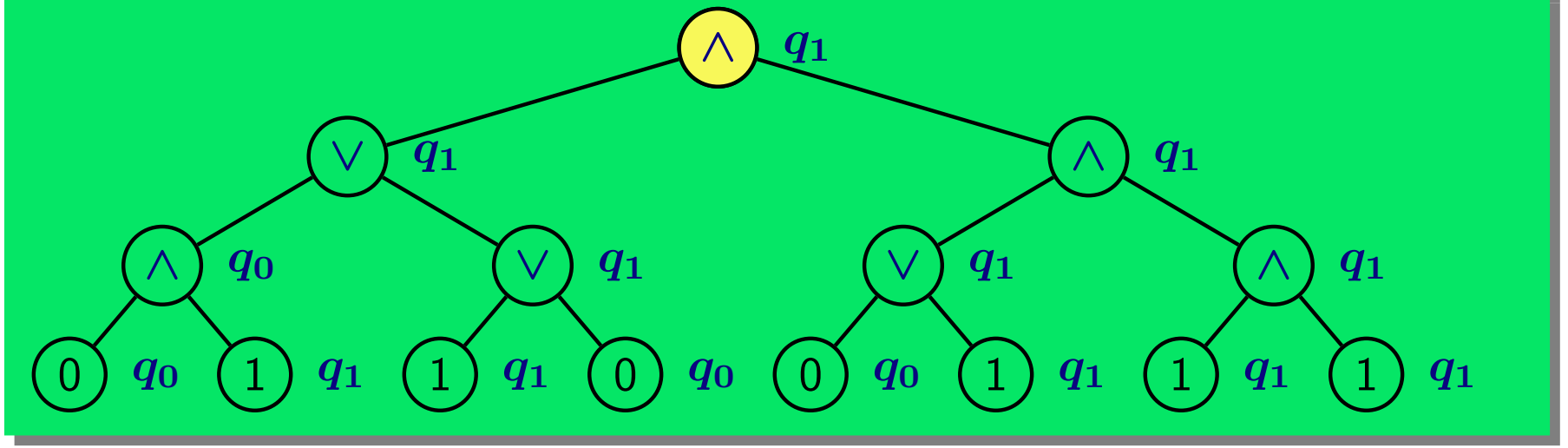
$\delta(\vee, q_0) = \{(q_0, q_0)\}$

$\delta(0, q_0) = \{\epsilon\}; \delta(0, q_1) = \emptyset$

$\delta(1, q_1) = \{\epsilon\}; \delta(1, q_0) = \emptyset$

Bottom-Up Automaten

Beispiel: Auswertung baumartiger Schaltkreise



Idee

- Zwei Zustände: q_0, q_1
- Akzeptierend an der Wurzel: q_1

Transitionen

$\delta(\wedge, q_1) = \{(q_1, q_1)\}$

$\delta(\wedge, q_0) = \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\}$

$\delta(\vee, q_1) = \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\}$

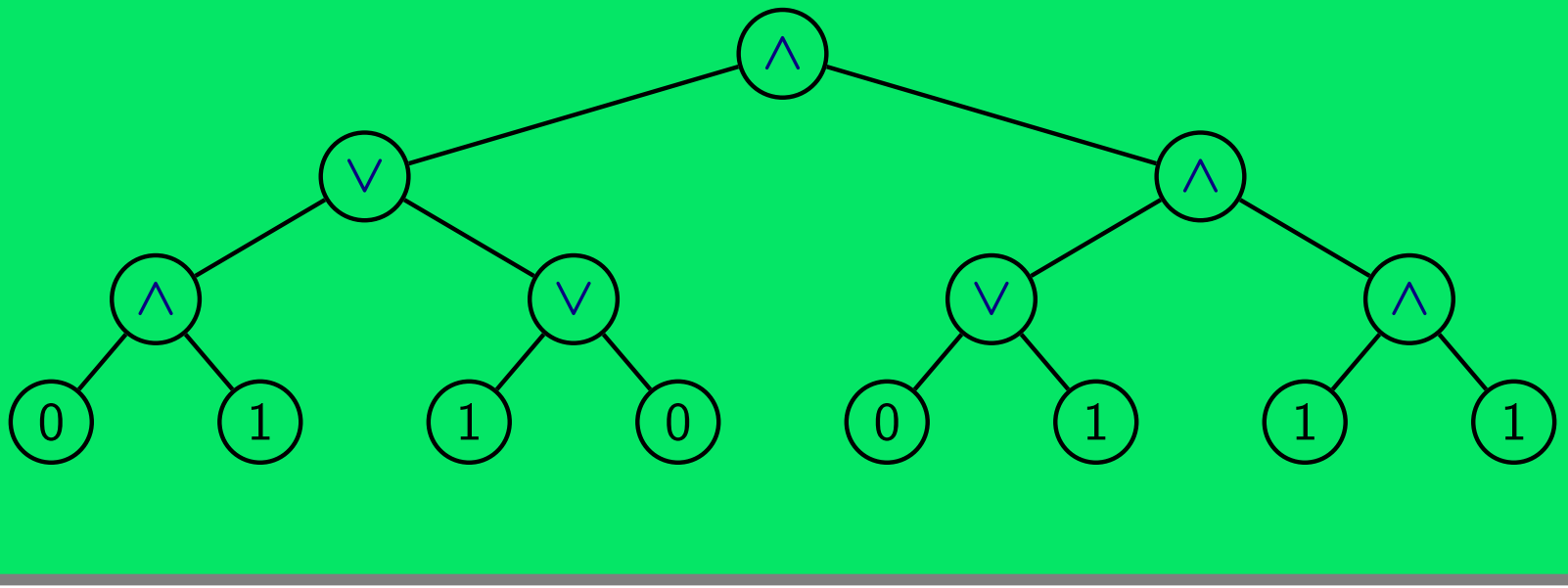
$\delta(\vee, q_0) = \{(q_0, q_0)\}$

$\delta(0, q_0) = \{\epsilon\}; \delta(0, q_1) = \emptyset$

$\delta(1, q_1) = \{\epsilon\}; \delta(1, q_0) = \emptyset$

Top-Down Automaten

Beispiel



Idee

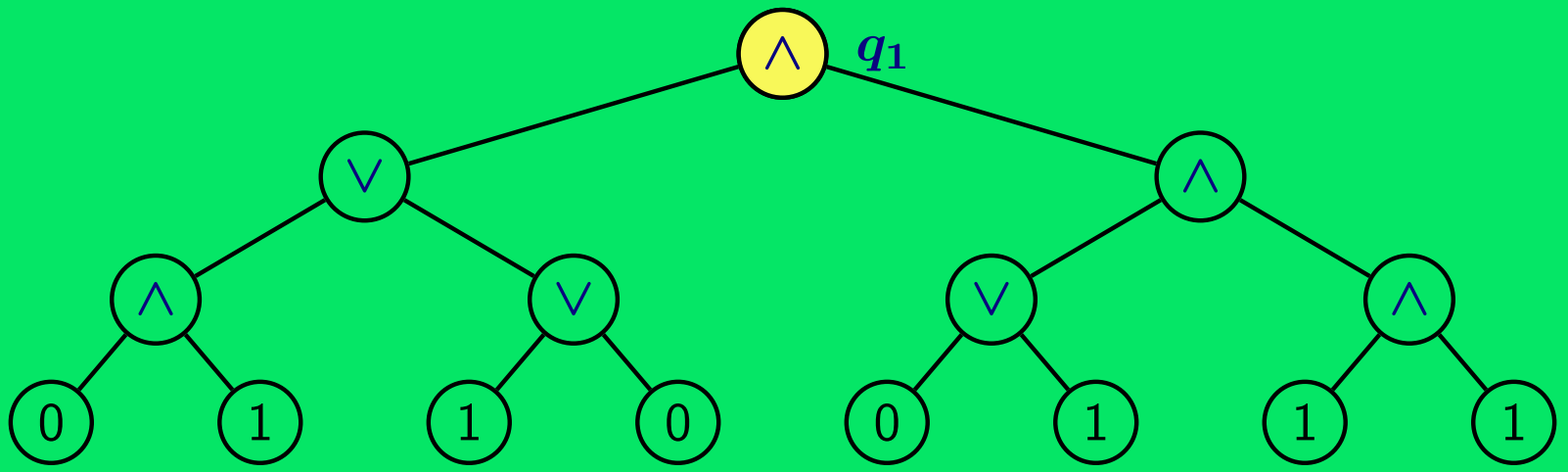
- Rate richtige Werte von Wurzel aus
- Teste an den Blättern
- Zustände: q_0, q_1, acc
- Startzustand q_1 an der Wurzel
- Akzeptieren, wenn alle Blätter acc annehmen

Transitionen

$$\delta(\wedge, q_1) = \{(q_1, q_1)\}$$
$$\delta(\wedge, q_0) = \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\}$$
$$\delta(\vee, q_1) = \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\}$$
$$\delta(\vee, q_0) = \{(q_0, q_0)\}$$
$$\delta(0, q_0) = \{\text{acc}\}; \delta(0, q_1) = \emptyset$$
$$\delta(1, q_1) = \{\text{acc}\}; \delta(1, q_0) = \emptyset$$

Top-Down Automaten

Beispiel



Idee

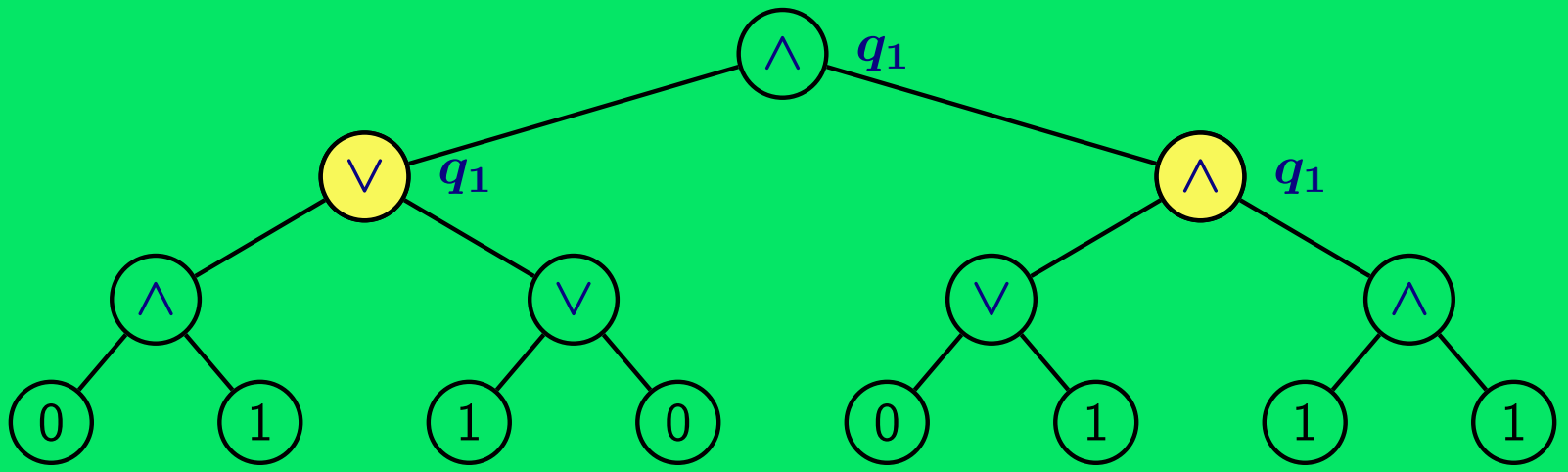
- Rate richtige Werte von Wurzel aus
- Teste an den Blättern
- Zustände: q_0, q_1, acc
- Startzustand q_1 an der Wurzel
- Akzeptieren, wenn alle Blätter acc annehmen

Transitionen

$$\delta(\wedge, q_1) = \{(q_1, q_1)\}$$
$$\delta(\wedge, q_0) = \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\}$$
$$\delta(\vee, q_1) = \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\}$$
$$\delta(\vee, q_0) = \{(q_0, q_0)\}$$
$$\delta(0, q_0) = \{\text{acc}\}; \delta(0, q_1) = \emptyset$$
$$\delta(1, q_1) = \{\text{acc}\}; \delta(1, q_0) = \emptyset$$

Top-Down Automaten

Beispiel



Idee

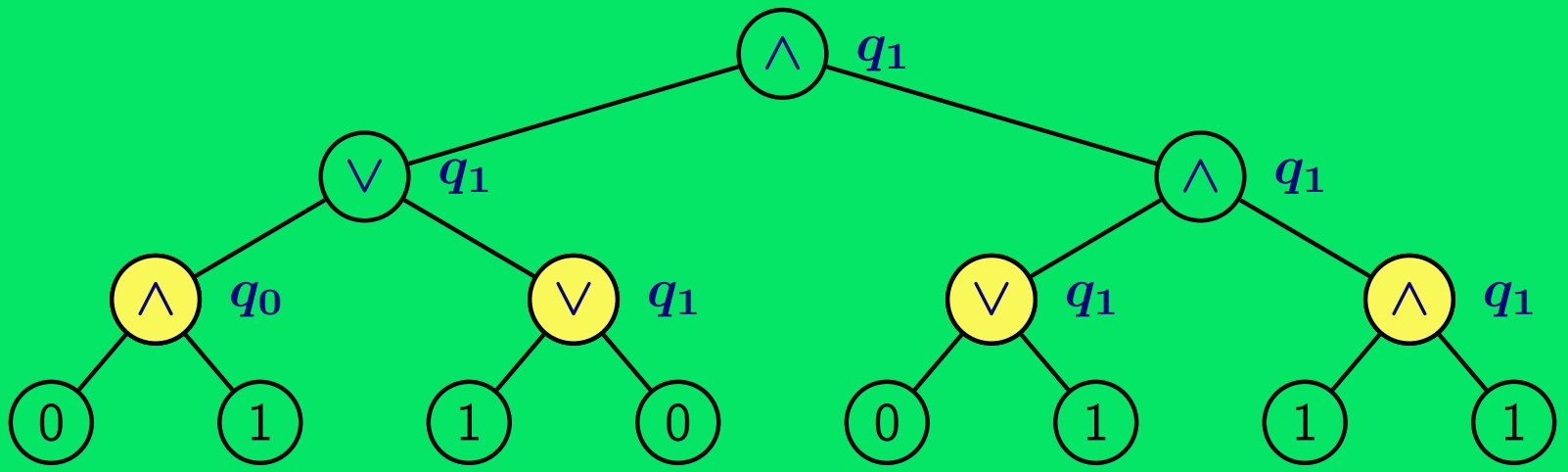
- Rate richtige Werte von Wurzel aus
- Teste an den Blättern
- Zustände: q_0, q_1, acc
- Startzustand q_1 an der Wurzel
- Akzeptieren, wenn alle Blätter acc annehmen

Transitionen

$$\delta(\wedge, q_1) = \{(q_1, q_1)\}$$
$$\delta(\wedge, q_0) = \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\}$$
$$\delta(\vee, q_1) = \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\}$$
$$\delta(\vee, q_0) = \{(q_0, q_0)\}$$
$$\delta(0, q_0) = \{\text{acc}\}; \delta(0, q_1) = \emptyset$$
$$\delta(1, q_1) = \{\text{acc}\}; \delta(1, q_0) = \emptyset$$

Top-Down Automaten

Beispiel



Idee

- Rate richtige Werte von Wurzel aus
- Teste an den Blättern
- Zustände: q_0, q_1, acc
- Startzustand q_1 an der Wurzel
- Akzeptieren, wenn alle Blätter acc annehmen

Transitionen

$\delta(\wedge, q_1) = \{(q_1, q_1)\}$

$\delta(\wedge, q_0) = \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\}$

$\delta(\vee, q_1) = \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\}$

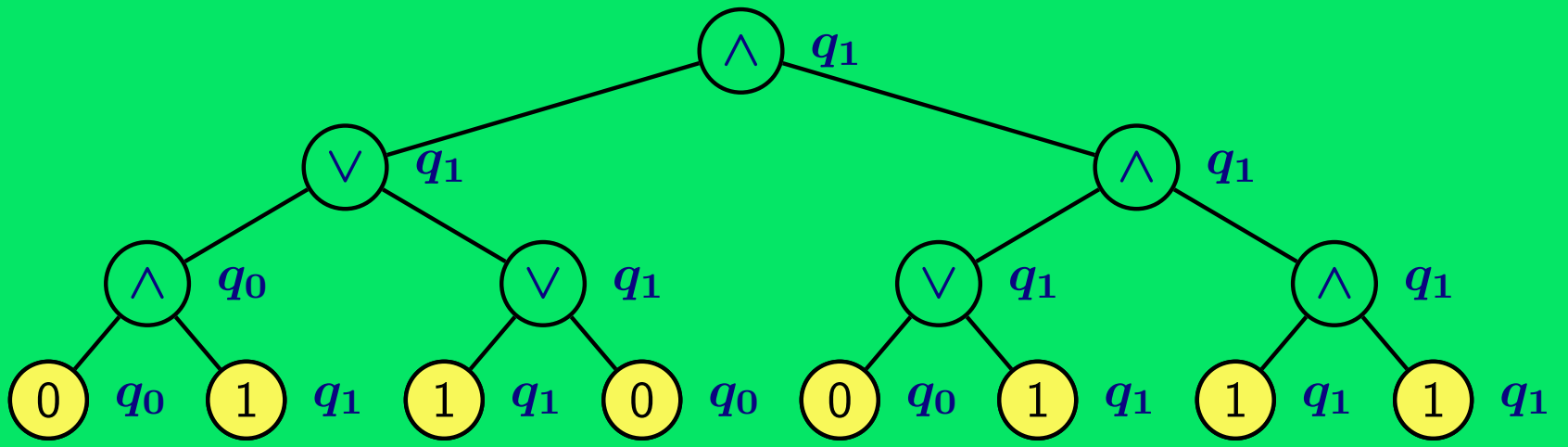
$\delta(\vee, q_0) = \{(q_0, q_0)\}$

$\delta(0, q_0) = \{\text{acc}\}; \delta(0, q_1) = \emptyset$

$\delta(1, q_1) = \{\text{acc}\}; \delta(1, q_0) = \emptyset$

Top-Down Automaten

Beispiel



Idee

- Rate richtige Werte von Wurzel aus
- Teste an den Blättern
- Zustände: q_0, q_1, acc
- Startzustand q_1 an der Wurzel
- Akzeptieren, wenn alle Blätter acc annehmen

Transitionen

$\delta(\wedge, q_1) = \{(q_1, q_1)\}$

$\delta(\wedge, q_0) = \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\}$

$\delta(\vee, q_1) = \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\}$

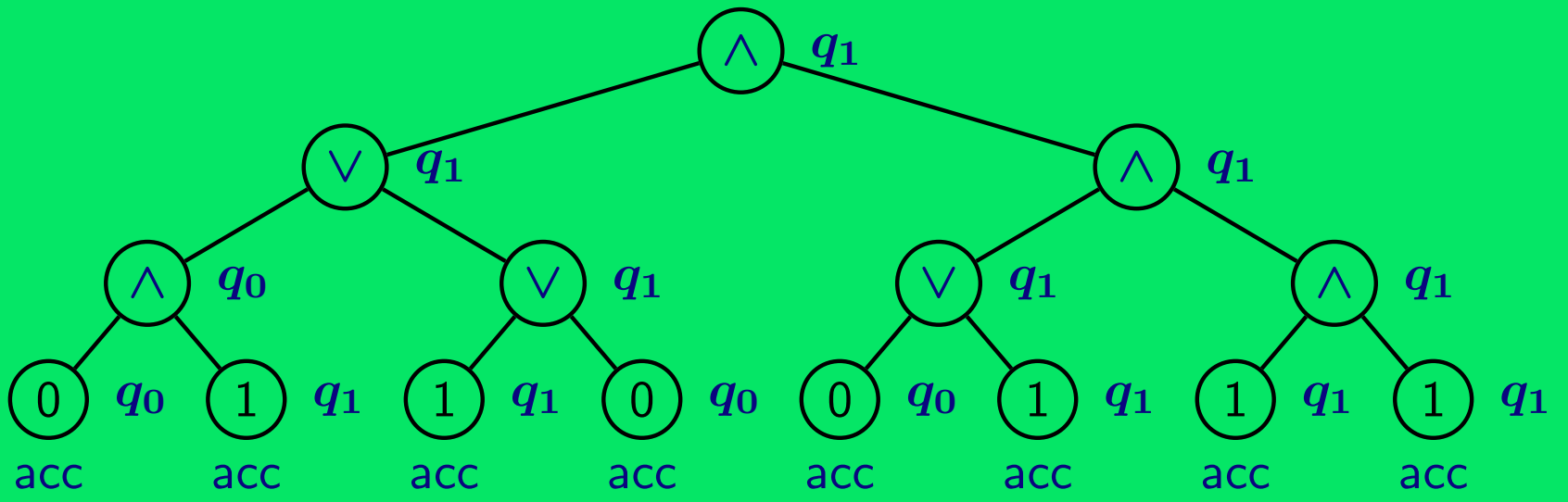
$\delta(\vee, q_0) = \{(q_0, q_0)\}$

$\delta(0, q_0) = \{\text{acc}\}; \delta(0, q_1) = \emptyset$

$\delta(1, q_1) = \{\text{acc}\}; \delta(1, q_0) = \emptyset$

Top-Down Automaten

Beispiel



Idee

- Rate richtige Werte von Wurzel aus
- Teste an den Blättern
- Zustände: q_0, q_1, acc
- Startzustand q_1 an der Wurzel
- Akzeptieren, wenn alle Blätter acc annehmen

Transitionen

$\delta(\wedge, q_1) = \{(q_1, q_1)\}$

$\delta(\wedge, q_0) = \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\}$

$\delta(\vee, q_1) = \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\}$

$\delta(\vee, q_0) = \{(q_0, q_0)\}$

$\delta(0, q_0) = \{\text{acc}\}; \delta(0, q_1) = \emptyset$

$\delta(1, q_1) = \{\text{acc}\}; \delta(1, q_0) = \emptyset$

Reguläre Baumsprachen

Satz

Für eine Baumsprache L sind äquivalent:

(a) L wird von einem nichtdeterministischen Bottom-up Automaten akzeptiert

(b) L wird von einem deterministischen Bottom-up Automaten akzeptiert

(c) L wird von einem nichtdeterministischen Top-down Automaten akzeptiert

$\longrightarrow$ **Reguläre Baumsprachen**

Beweisidee

(a) $\Longrightarrow$ (b): Potenzmengen-Konstruktion

(a) $\Longleftrightarrow$ (c): Die gleiche Sache - aus zwei Richtungen betrachtet

Bemerkung

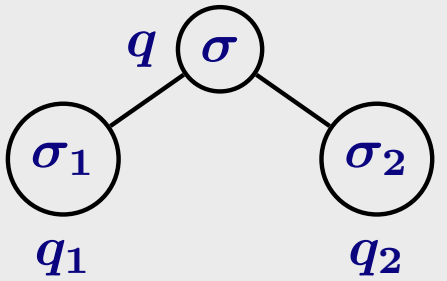
Es gibt viele weitere äquivalente Beschreibungen

Von binären zu unbeschränkten Baumautomaten

Binäre Baumautomaten

Transitionen durch endlichen Menge gegeben:

$\delta(\sigma, q) = \{(q_1, q_2), (q_3, q_4), \ldots\}$

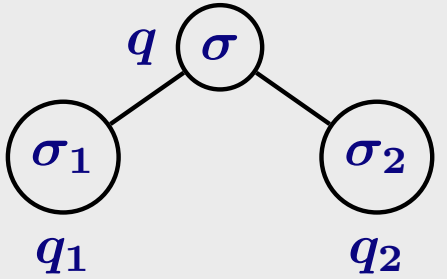


Von binären zu unbeschränkten Baumautomaten

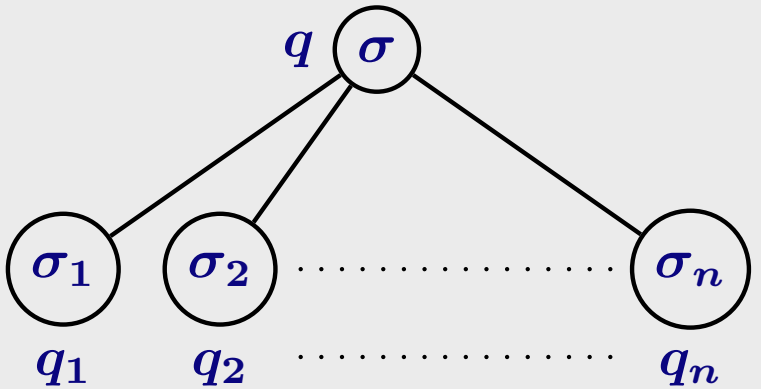
Binäre Baumautomaten

Transitionen durch endlichen Menge gegeben:

$\delta(\sigma, q) = \{(q_1, q_2), (q_3, q_4), \ldots\}$



Unbeschränkte Baumautomaten

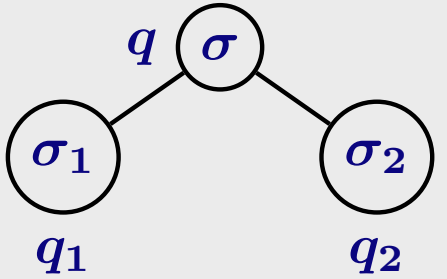


Von binären zu unbeschränkten Baumautomaten

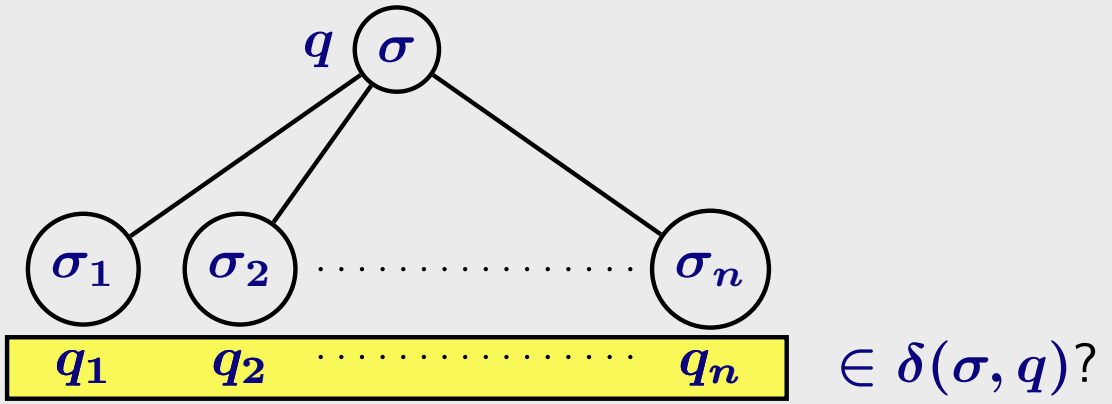
Binäre Baumautomaten

Transitionen durch endlichen Menge gegeben:

$\delta(\sigma, q) = \{(q_1, q_2), (q_3, q_4), \ldots\}$



Unbeschränkte Baumautomaten



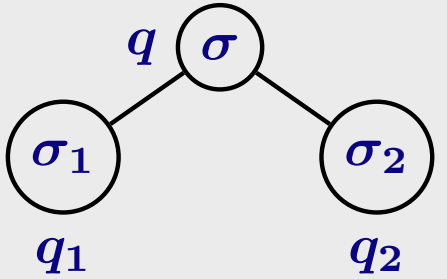
$\in \delta(\sigma, q)$?

Von binären zu unbeschränkten Baumautomaten

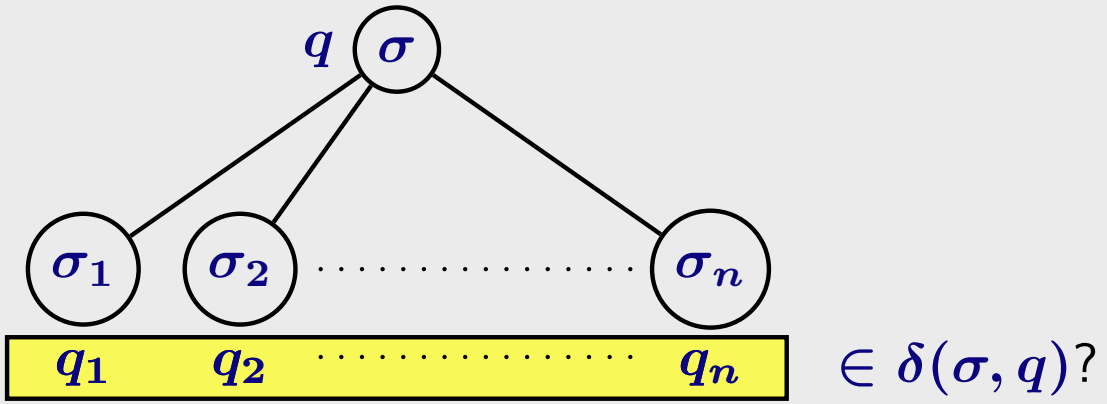
Binäre Baumautomaten

Transitionen durch endlichen Menge gegeben:

$\delta(\sigma, q) = \{(q_1, q_2), (q_3, q_4), \ldots\}$



Unbeschränkte Baumautomaten



$\in \delta(\sigma, q)$?

$\delta(\sigma, q)$

- Bei unbeschränkten Baumautomaten ist $\delta(\sigma, q)$ eine reguläre Sprache
- $\delta(\sigma, q)$ kann z.B. durch einen regulären Ausdruck spezifiziert werden

  [Brüggemann-Klein, Murata, Wood 2001]

XML-Schema und Reguläre Baumsprachen

- XML-Schemasprachen lassen sich als eingeschränkte reguläre Baumsprachen auffassen [Murata, Lee, Mani 2000]
- Effiziente Validierung ebenfalls für eingeschränkte Klassen möglich
  (umfasst Kern von XML-Schema)
- Erweiterung: Reguläre Anfragen

$\longrightarrow$ Reguläre Baumsprachen bilden einen soliden Hintergrund für das Studium von XML-Schemasprachen

Inhalt

Weitere Zusammenhänge

Übersicht

XPath

- Vertikale Navigation:

  $$\texttt{//Vita/Died/*} \equiv \Sigma^* \cdot \texttt{Vita} \cdot \texttt{Died} \cdot \Sigma$$

- Auch Navigation längs anderer **Achsen**: parent, descendant, following-sibling,...
- Verallgemeinung regulärer Ausdrücke für Bäume: **Caterpillars**
  - Navigation & Tests
  - $\text{down}^* \cdot \text{first} \cdot \text{right} \cdot \text{right} \cdot a \cdot \text{up}$:
    Gehe zu Knoten, dessen drittes Kind Label a hat
- XPath-Auswertung: Polynomielle Zeit mit dynamischer Programmierung [Gottlob, Koch, Pichler 03]

XSLT Enge Korrespondenz zu Treewalk-Automaten mit Pebbles

XQuery ...noch viel zu tun

Inhalt

Fazit

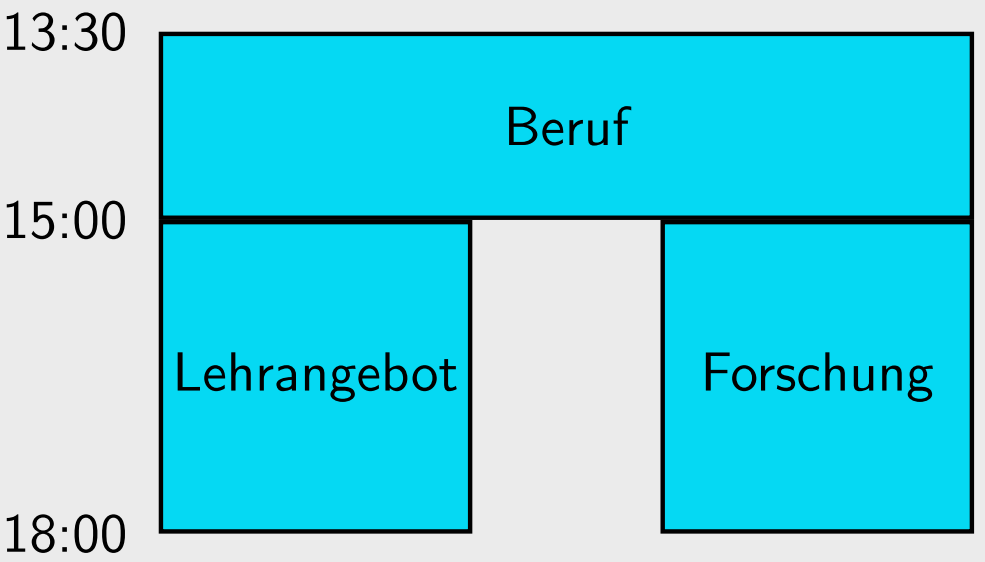
- Formale Sprachen spielen für die Grundlagen von XML-Srachen eine wichtige Rolle
- Viele formalsprachliche Konzepte lassen sich also im Rahmen eines aktuellen Anwendungsbereiches motivieren
- Geeignet für die Schule???

Literatur

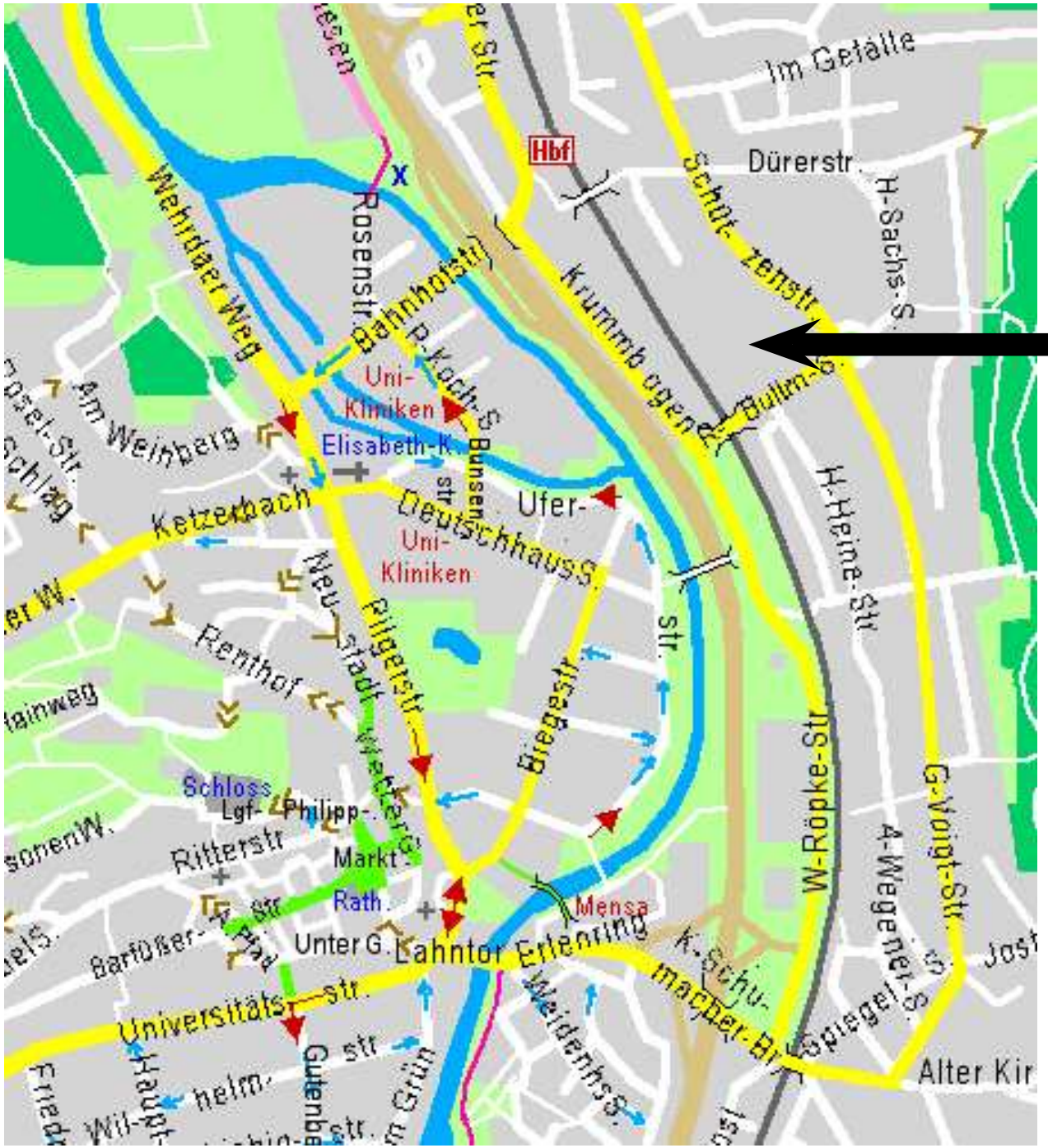
- Anne Brüggemann-Klein, Derick Wood. *One-Unambiguous Regular Languages*. Inf. Comput. 142(2): 182-206 (1998)
- Murata, Lee, Main. *Taxonomy of XML Schema Languages using Formal Language Theory*. Technischer Bericht, 2000
- Geert Jan Bex, Wim Martens, Frank Neven, Thomas Schwentick. *Expressiveness of XSDs: from Practice to Theory, There and Back Again*. International World Wide Web Conference, WWW 2005
- Anne Brüggemann-Klein, Derick Wood. *Caterpillars: A Context Specification Technique*. Markup Languages 2(1): 81-106 (2000)
- Frank Neven. *Automata, Logic, and XML*. CSL 2002: 2-26
- Georg Gottlob, Christoph Koch, Reinhard Pichler. *XPath Processing in a Nutshell*. SIGMOD Record 32(1 + 2) (2003)
- Nils Klarlund, Thomas Schwentick and Dan Suciu. *XML: Model, Schemas, Types, Logics, and Queries*. In *Logics for Emerging Applications of Databases 2003*, Springer, 1-41, 2003
- `www.w3.org`
- Harald Schöning. *XML und Datenbanken : Konzepte und Systeme*. Hanser, 2003

Ankündigung: „Tag der Informatik“

- Mittwoch, 27.4.05, nachmittags
- Informationen für Schüler und Studenten des Grundstudiums über
  - Studium und Lehre der Informatik in Marburg
  - Forschungsschwerpunkte (Doktoranden berichten)
  - Berufsfelder (Ehemalige berichten)
- Vorläufige Planung:



Waggonhalle: Rotkehlchen



Rudolf-Bultmann-Straße