Grundlagen der Programmierung

Dr. Christian Herzog
Technische Universität München

Wintersemester 2017/2018

Kapitel 4: Algorithmen und Textersetzungssysteme

Überblick über dieses Kapitel

- Zentraler Begriff der Informatik: Algorithmus
- Unterschiedliche Definitionen
- Taxonomie von Algorithmen
- Textersetzungssysteme
  - Semi-Thue Systeme
  - Markov-Algorithmen
- Chomsky-Grammatik
- Backus-Naur-Form, Syntaxdiagramme

Arten der Funktionsbeschreibung

- Die einfachste Aufgabe eines Informatik-Systems ist die Realisierung einer Funktion $f: A \rightarrow B$. Wir können solche Funktionen auf verschiedene Weisen beschreiben.
- **Statisch und deklarativ**
  - In Form von Gleichungen zur Darstellung der Zusammenhänge von A und B, z.b. $g(a, b) = 0$, falls $b = f(a)$. Diese Beschreibung hilft nicht als konstruktive Regel, um $f(a)$ zu berechnen, reicht aber oft für die erste Modellierung.
- **Tabellarisch:**
  - Für alle Werte a des Definitionsbereichs A wird der zugeordnete Wert $f(a)$ in einer Wertetabelle für den Wertebereich B festgehalten.
    Werte werden “berechnet”, indem man sie nachschlägt.
- **Prozedural:**
  - Durch eine **algorithmische** Beschreibung.

Algorithmische Beschreibung

- Eine algorithmische Beschreibung dient zur Berechnung des Ergebnisses für beliebige Argumente des Definitionsbereichs.
- Die Beschreibung kann in einer beliebigen Sprache sein (natürliche Sprache, Graph, Pseudosprache, Programmiersprache)
  - Die Beschreibung muss endlich sein und muss die Berechnung als Folge von elementaren Operationen mit hinreichender Präzision spezifizieren.
- Eine solche Beschreibung heißt **Algorithmus**.
- Beispiele von Algorithmen:
  - Verfahren zur schriftlichen Addition, Subtraktion, Multiplikation, Division
  - Rezept zum Herstellen von Semmeln
  - Spielregel für Dame oder Mühle oder Halma ...

Algorithmus: Definition

- **Definition Algorithmus:**
  - Ein Algorithmus ist ein Verfahren zur Verarbeitung von Daten mit einer **präzisen**, **endlichen** Beschreibung unter Verwendung **effektiver** Arbeitsschritte.
- Präzise: In einer eindeutigen Sprache abgefasst
- Endlich: In einer endlichen Form beschrieben
- Effektiv: Die Arbeitsschritte sind tatsächlich ausführbar

- Ein Algorithmus muss nicht terminieren.
- **Definition Terminierender Algorithmus:** Das Verfahren hat *endlich* viele Schritte.
  - Broy verwendet diese Definitionen.

Was ist präzise, endlich, effektiv?

- ***Präzise**: in einer eindeutigen Sprache abgefasst*
  - maschinell lesbar und ausführbar
  - Eine Beschreibung ist unpräzise, wenn eine Bedingung nicht eindeutig ausgewertet werden kann.
    Beispiel: **Lasse den Braten schmoren, bis er gut durch ist.**
- ***Endlich:** In einer endlichen Form beschrieben*
  - Ich brauche nur endlich viel Papier (oder Elektronen:-).
- ***Effektiv:** Die Arbeitsschritte sind tatsächlich ausführbar*
  - Beispiel eines nicht effektiven Arbeitschrittes:
    Wenn bei optimalem Spiel Weiß bei Schach immer gewinnt, ...

Algorithmus: Zweite Definition

Definition (Knuth): Ein Algorithmus ist

1) Eine Beschreibung eines Verfahrens und seiner Daten mit Hilfe einer Sprache.
2) Eine ***präzise*** Beschreibung der einzelnen Schritte des Verfahrens.
3) Eine ***endliche*** Aufschreibung für die Darstellung eines endlichen, aber im Allgemeinen unbeschränkten Verfahrens.
4) Jeder Einzelschritt des Verfahrens ist in ***endlicher*** Zeit ausführbar.
5) Das Verfahren hat nur ***endlich*** viele Schritte.

Aus 4) und 5) folgt, dass das Verfahren terminieren muss, sonst ist es kein Algorithmus

- Goos verwendet diese alternative Definition.

Algorithmus Definition: Diskussion

- Einer der fundamentalen Begriffe der Informatik ist nicht allgemein akzeptiert. Warum?
  - Die zweite Definition (“Ein Algorithmus muss terminieren.”) bezieht sich auf die Berechnung einer einzelnen Funktion, und da ist es schon besser, wenn wir für eine Eingabe auch eine Ausgabe bekommen.
  - Die erste Definition (“Ein Algorithmus muss nicht terminieren.”) bezieht auch andere interessante Informatik-Systeme ein, von denen wir erwarten, dass sie nicht terminieren (z.B. ein Betriebssystem oder ein Web-Server)
- Was sagt das über den Reifegrad der Informatik?
  - Informatik ist immer noch ein junges Gebiet.
  - Unterschiedliche Definitionen in vielen Bereichen der Informatik sind leider gang und gäbe.
- In dieser Vorlesung benutzen wir die erste Definition, d.h. nicht jeder Algorithmus terminiert.

Der Algorithmus als Grundlage aller Informatik-Systeme

- Was haben wir soweit gelernt?
  - Es gibt mehrere Definitionen für Algorithmus
    - Hält (terminiert) er oder hält er nicht?
- Was kommt jetzt?
  - **Zeichenfolgen**
  - **Textersetzungssysteme** als einfache Algorithmen
    - Beispiele von Textersetzungssystemen
    - Markov-Algorithmen als deterministische Algorithmen
  - **Grammatiken** zur Spezifikation von Textersetzungssystemen
  - Backus-Naur-Form und Syntax-Diagramme

Zeichenfolgen

- Die Darstellung von Information durch gesprochene Sprache (durch eine Folge von Lauten) und durch Schrift (durch eine Folge von Zeichen) ist eine fundamentale Errungenschaft der Menschheit.
- Auch in der Informatik verwenden wir sehr oft Zeichensequenzen bzw. Zeichenfolgen.
  - Ein Algorithmus kann beispielsweise als eine endliche Zeichenfolge angesehen werden.
- Mit V bezeichnen wir im folgenden den zugrunde liegenden **Zeichenvorrat**: eine endliche Menge von Zeichen.
- Gibt es auf V eine lineare Ordnung (d.h. eine feste Reihenfolge der Zeichen), so nennt man V ein **Alphabet**.
- Eine Zeichenfolge über einem Zeichenvorrat V wird auch als **Wort** über V bezeichnet.

Definitionen

- **Definition Wort:** Für eine gegebene Menge V von Zeichen bildet eine Folge $x = X_1, X_2, \ldots, X_n$ *mit* $X_i \in V$ ein Wort der Länge n über V. Wir schreiben auch $x = X_1 X_2 \ldots X_n$ für das Wort und sprechen von einem n-Tupel.
- $|x|$ bezeichnet die Länge von x, d.h. $|X_1 X_2 \ldots X_n| = n$
- ε bezeichnet die leere Zeichenfolge bzw. das leere Wort, $|\varepsilon| = 0$
- V^n bezeichnet die Menge aller n-Tupel bzw. die Menge aller Worte der Länge n über V.
  - $V^n =_{def} V \times V \times \ldots \times V$ (n-faches Kreuzprodukt über V)
  - $V^0 = \{\varepsilon\}$
- V^* bezeichnet die Menge aller Worte über V, d.h. aller endlichen Folgen von Zeichen aus V.
  - $V^* = V^0 \cup V^1 \cup V^2 \cup V^3 \cup \ldots \cup V^n \cup \ldots$ (unendliche Vereinigung)
- V^+ bezeichnet die Menge aller Worte über V ohne das leere Wort, d.h. $V^+ = V^* \setminus V^0 = V^* \setminus \{\varepsilon\}$.

Konkatenation von Zeichenfolgen

- Über V^* definieren wir als Operation die zweistellige Abbildung **Konkatenation**:

  conc: $V^* \times V^* \rightarrow V^*$
- Das Wort conc (v, w) mit $v, w \in V^*$ ist dasjenige Wort, das durch Aneinanderreihen („Konkatenieren“) der Zeichenfolgen v und w gebildet wird.
  - Ist $v = V_1 V_2 \ldots V_m$ und $w = W_1 W_2 \ldots W_n$, so ist conc $(v, w) = V_1 V_2 \ldots V_m W_1 W_2 \ldots W_n$
- Konkatenation wird oft in Infixweise geschrieben:
  - $v \circ w =_{def}$ conc (v,w)
- Für die Konkatenation gelten folgende Gesetze:
  - $(u \circ v) \circ w = u \circ (v \circ w)$ (Assoziativität)
  - $w \circ \varepsilon = w = \varepsilon \circ w$ (Neutrales Element)
- V^* ist also ein Monoid (eine Halbgruppe mit neutralem Element) bezüglich der Konkatenation.

Textersetzungssystem

- Ein Textersetzungssystem ist eine sehr einfache Form eines Algorithmus. Mit Hilfe von Regeln werden Worte in andere Worte übergeführt.
- Sei V ein endlicher Zeichenvorrat.
- **Definition Regel**
  - Seien l und r Worte aus V^*.
  - Eine **Regel** $l \rightarrow r$ bedeutet, dass wir das Auftreten des Wortes l durch das Auftreten des Wortes r ersetzen können.
  - l heißt die *linke Seite* der Regel, r die *rechte Seite*.

Definition Textersetzungssystem

- **Definition**: Eine Menge T = $\{a \rightarrow b, c \rightarrow d, \ldots\}$ von Regeln über einem Zeichenvorrat V heißt **Textersetzungssystem** (auch **Semi-Thue-System**), wenn die folgenden Metaregeln gelten:
  - Sei w ein Wort über V mit dem Teilwort a, d.h. w ist von der Form $w = v \circ a \circ x$ mit geeigneten $v, x \in V^*$
  - Wenn $a \rightarrow b$ eine Regel von T ist, dann kann man das Teilwort a in w durch b ersetzen: $v \circ a \circ x \Rightarrow v \circ b \circ x$
  - Wenn a in w mehrfach vorkommt oder mehrere Regeln anwendbar sind, so kann man das Teilwort bzw. die Regel beliebig wählen.
  - Die Regeln können beliebig oft angewandt werden.
  - $v \circ b \circ x$ heißt (**direkt**) **abgeleitet** aus $v \circ a \circ x$ über die Regel $a \rightarrow b$.
    - Wir verwenden für die direkte Ableitung das Symbol $\Rightarrow$.

Beispiel eines Textersetzungssystems: Addition von Strichzahlen

- Ein Algorithmus für die Addition von natürlichen Zahlen, dargestellt durch Folgen von Strichen **|**, die durch die Sonderzeichen **<** und **>** begrenzt sind.
  - z. B. die 5 wird durch **<|||||>** dargestellt.
- Ein Textersetzungssystem für diesen Algorithmus lautet:

  Zeichenvorrat V = {**<**, **|**, **>**, **+**}

  Die Regelmenge T besteht nur aus einer einzigen Regel:

  >+< → ε
- Beispiele für Ableitungen:

  <|>+<|> ⇒ **<||>**

  <|||>+<||> ⇒ **<|||||>**

Multiplikation von Strichzahlen

Beispiel:

<||>*<|||>
⇒ <|>*<|||| >

… ⇒ <|>*<|m|||>
… ⇒ <|>*<|m|m||>
… ⇒ <|>*<|m|m|m|>

… ⇒ <>*<|mm|mm|mm||>
… ⇒ <mmmmmm>
… ⇒ <||||||>

- ***Grundidee des Algorithmus:***

1. Ich schiebe einen Strich vom linken Faktor in den rechten (roter Strich).
2. Dann schiebe ich den roten Strich durch alle Striche des rechten Faktors, und schreibe ein m für jeden gefundenen blauen Strich.
3. Ich wiederhole 1. und 2., bis keine Striche mehr im linken Faktor sind.
4. Dann räume ich auf: Ich lösche den (leeren) linken Faktor, das Zeichen * und alle **|**.
5. Damit habe ich das Resultat: Ich muss nur noch alle m in **|** konvertieren.

Regeln für ein Textersetzungssystem für die Multiplikation von Strichzahlen

- Das Rüberziehen der | aus dem linken Fakor und das Verfielfachen der | im rechten Faktor erledigen wir mit 3 Hilfszeichen d, e, m (vgl. Broy):
  - d (Duplikator, steht für den „roten Strich“), e (Eliminator) und m (Marker):

(1) |>*< → >*<d	(1) Rüberziehen eines | aus dem linken Faktor („roter Strich“)
(2)	d| → |md	(2) Ein m für ein gefundenenes | im rechten Faktor
(3)	dm → md	(3) Weiterschieben des „roten Strichs“
(4)	d> → >	(4) Räume auf: Der „rote Strich“ hat seine Schuldigkeit getan
(5)	<>*< → <e	(5) Wenn der linke Faktor leer geworden ist: Lösche ihn zusammen mit dem * und setze den Eliminator e ganz links!
(6)	e| → e	(6) Aufräumregel: lösche jedes gefundene „alte“ |
(7)	em → |e	(7) Ersetzt jeweils einen Marker m durch ein „neues“ | und schiebt den Eliminator e weiter nach rechts
(8)	e> → >	(8) Räume auf: Der Eliminator hat seine Schuldigkeit getan!

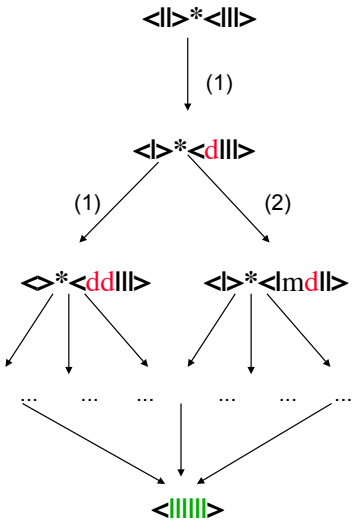
Strichzahlenmultiplikation am Beispiel

(1) |>*< → >*<d
(2) d| → |md
(3) dm → md
(4) d> → >
(5) <>*< → <e
(6) e| → e
(7) em → |e
(8) e> → >

Beispiel: <||>*<|||>

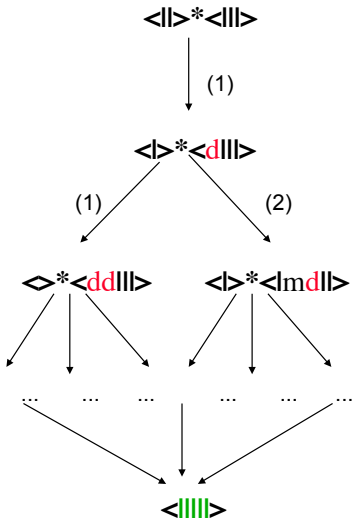
⇒	<|>*<d|||>	(1)
⇒	<|>*<|md||>	(2)
⇒	<|>*<|m|md|>	(2)
⇒	<|>*<|m|m|md>	(2)
⇒	<|>*<|m|m|m>	(4)
⇒	<>*<d|m|m|m>	(1)
⇒	<>*<|mdm|m|m>	(2)
⇒	<>*<|mmd|m|m>	(3)
⇒	<>*<|mm|mdm|m>	(2)
⇒	<>*<|mm|mmd|m>	(3)
⇒	<>*<|mm|mm|mdm>	(2)
⇒	<>*<|mm|mm|mmd>	(3)
⇒	<>*<|mm|mm|mm>	(4)
⇒	<e|mm|mm|mm>	(5)
⇒	<emm|mm|mm>	(6)
⇒	<|em|mm|mm>	(7)
… ⇒	<||||||e>	(6),(7)
⇒	<||||||>	(8)

Textersetzungssysteme und Determinismus



(1) |>*< → >*<d
(2) d| → |md
(3) dm → md
(4) d> → >
(5) <>*< → <e
(6) e| → e
(7) em → |e
(8) e> → >

Textersetzungssysteme und Determinismus



- **Wichtig:** Die Folge der Anwendung von Regeln des Textersetzungssystems braucht nicht immer dieselbe zu sein, d.h. der Ableitungsgraph (die Menge aller möglichen Ableitungen) kann mehr als einen Pfad enthalten.
- **Definition: Ein Textersetzungssystem** heißt **deterministisch**, wenn zu jeder Zeit immer nur eine Regel auf genau ein Teilwort anwendbar ist.
- **Definition: Ein Textersetzungssystem** heißt **determiniert**, wenn dieselbe Eingabe immer dieselbe Ausgabe ergibt.
- Die Strichzahlenmultiplikation ist also determiniert, aber nicht deterministisch.

Textersetzungssysteme und Algorithmen

- Ein Textersetzungssystem zeigt die Grundform eines Algorithmus:
  - Die Eingabe ist ein Wort *x*
  - Es gibt nur endlich viele Operationen o_1 , o_2 , …., o_n , nämlich die Regeln des Textersetzungssystems.
  - Der Algorithmus ist ausführbar, indem man Regeln auf die Eingabe *x* anwendet.

Taxonomie von Algorithmen

Algorithmen unterscheiden sich nach

- **der Anwendbarkeit von Regeln**
  - Terminierender Algorithmus
  - Nichtterminierender Algorithmus
  - Deterministischer Algorithmus
  - Nichtdeterministischer Algorithmus
- **der Beziehung zwischen Ein- und Ausgabe**
  - Determinierter Algorithmus
  - Nichtdeterminierter Algorithmus

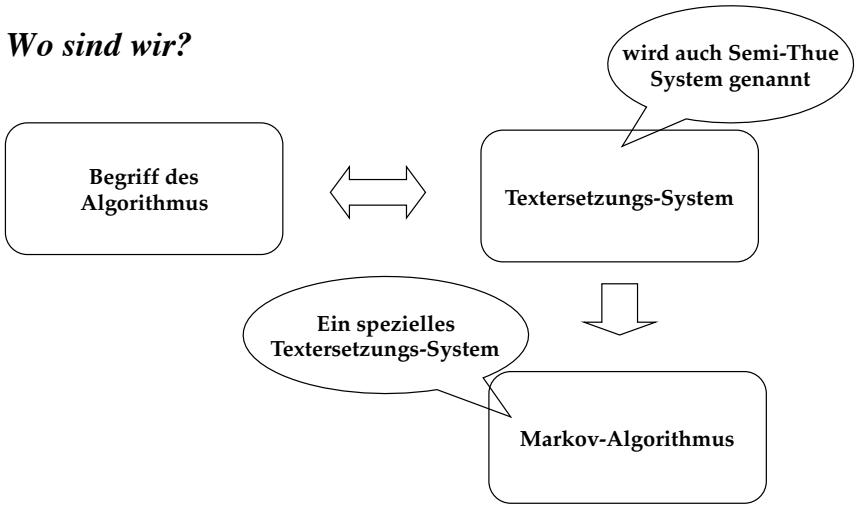
Algorithmenarten: Anwendbarkeit von Regeln

- **Terminierender Algorithmus:**
  - Keine Regel ist mehr anwendbar.
- **Nichtterminierender Algorithmus**:
  - Es ist immer mindestens eine Regel anwendbar.
- **Deterministischer Algorithmus:**
  - Zu jeder Zeit ist immer nur höchstens eine Regel auf genau ein Teilwort anwendbar, d.h. es gibt genau eine Berechnung.
  - Falls er terminiert, liefert ein deterministischer Algorithmus genau eine Ausgabe.
- **Nichtdeterministischer Algorithmus:**
  - Zu einer gegebenen Eingabe *x* sind mehrere Regeln bzw. eine Regel auf mehrere Teilwörter von *x* anwendbar.

Algorithmenarten: Beziehung zwischen Ein- und Ausgabe

- **DeterminierterAlgorithmus:**
  - Zu jeder Eingabe gibt es immer dieselbe Ausgabe. Der Algorithmus kann nichtdeterministisch sein, d.h. es gibt verschiedene Berechnungspfade für dieselbe Eingabe.
    - Beispiel: Das Textersetzungsystem für die Multiplikation von Strichzahlen
- **Nichtdeterminierter Algorithmus:**
  - Liefert bei wiederholter Anwendung auf die gleiche Eingabe unterschiedliche Ergebnisse.
    - Beispiel: Ein Zufallszahlengenerator ist ein nichtdeterminierter Algorithmus.

Wo sind wir?



Markov-Algorithmen

- **Definition:** Ein Markov-Algorithmus ist ein Textersetzungssystem mit folgenden *zusätzlichen Eigenschaften*:
  - Die Menge der Regeln ist linear geordnet: die Reihenfolge der Aufschreibung der Regeln ist wesentlich.
  - Markov-Algorithmen enthalten spezielle Regeln der Form $a \rightarrow . b$, die durch einen Punkt nach dem Pfeil gekennzeichnet sind. Sie werden *haltende Regeln* genannt.
  - Für die Anwendung von Regeln gelten die folgenden beiden *Metaregeln*:
    - Wähle in jedem Schritt die (von der Aufschreibung her) erste anwendbare Regel. Falls sie auf mehrere Teilwörter anwendbar ist, wende sie auf das am weitesten links stehende Teilwort an.
    - Wende die Regeln solange an, bis eine haltende Regel angewandt wurde, oder bis keine Regel mehr anwendbar ist.
- **Bemerkung**: Durch die erste der beiden Metaregeln sind Markov-Algorithmen deterministisch und damit auch immer determiniert.

Markov-Algorithmus: Beispiel

Zeichenvorrat V = {0,L,a,b}

Regelsystem:

(1) aL → La
(2) a0 → 0a
(3) a → b
(4) Lb → b0
(5) 0b →. L
(6) b →. L
(7) ε → a

Sonderzeichen wie a und b werden auch *Schiffchen* genannt.

- Anwendung der Regeln auf das Eingabewort LOLL

L0LL ⇒ aL0LL (7)
⇒ La0LL (1)
⇒ L0aLL (2)
⇒ L0LaL (1)
⇒ L0LLa (1)
⇒ L0LLb (3)
⇒ L0Lb0 (4)
⇒ L0b00 (4)
⇒ LL00 (5)

Frage: Welches Problem löst der Algorithmus?

Antwort: Addition von L!

Noch ein Beispiel

Zeichenvorrat V = {0,L,a,b}

Regelsystem:

(1) ε → a
(2) aL → La
(3) a0 → 0a
(4) a → b
(5) Lb → b0
(6) 0b →. L
(7) b →. L

- Anwendung der Regeln auf das Eingabewort LOLL

L0LL ⇒ aLOLL (1)
⇒ aaLOLL (1)
⇒ aaaLOLL (1)
⇒ aaaaLOLL (1)
⇒ …

Frage: Was macht dieser Algorithmus?

Vergleich der Algorithmen

Algorithmus 1 terminiert	Algorithmus 2 terminiert nicht
(1) aL → La	(1) ε → a
(2) a0 → 0a	(2) a0 → 0a
(3) a → b	(3) a → b
(4) Lb → b0	(4) Lb → b0
(5) 0b → . L	(5) 0b → . L
(6) b → . L	(6) b → . L
(7) ε → a	(7) aL → La

Die Reihenfolge der Regeln ist hier wesentlich!

Chomsky-Grammatiken

- Die natürliche Sprache enthält Regeln wie
  - Ein „Satz“ besteht aus „Substantiv“ und „Verb“.
  - Wenn es stimmt, dass „Fisch“ ein Substantiv ist, und „schwimmt“ ein Verb, dann ist „Fisch schwimmt“ ein „Satz“.
- Die natürliche Sprache hat viele solcher Regeln.
- Noam Chomsky war der erste, der versucht hat, die Regeln der natürlichen Sprache in Form eines Textersetzungssystems (Semi-Thue-Systems) zu formulieren.
  - Chomsky nannte diese Semi-Thue-Systeme dann **Grammatiken** und die Regeln **Produktionen.**

Syntaktische Variable und Terminale

- In einer Grammatik unterscheidet man syntaktische Begriffe wie *Satz, Substantiv, Verb* von den Wörtern der zu beschreibenden Sprache wie *Fisch, schwimmen*.
- Die Einzelzeichen des Zeichenvorrates T einer Grammatik heißen **Terminale.**
- Die syntaktischen Begriffe bilden den Zeichenvorrat *N* der **Nichtterminale** oder **syntaktische Variablen** einer Grammatik.
- Ziel einer Grammatik ist es, die terminalen Zeichenreihen zu beschreiben, die aus einer speziellen syntaktischen Variablen abgeleitet werden können, dem Startsymbol, **Axiom** oder **Ziel** *Z* der Grammatik.

Definition Chomsky-Grammatik

- Definition **Chomsky-Grammatik:**
  - Eine Grammatik $G = (T, N, P, Z)$ in der T ein Zeichenvorrat von Terminalen, N ein Zeichenvorrat von Nichtterminalen, P eine endliche Menge von Regeln und Z ein Zeichen aus N ist, heißt eine Chomsky-Grammatik.
- Eine Regel in einer Chomsky-Grammatik heißt **Produktion.**
- Z ist das **Axiom** oder **Ziel** von G.
- Die Vereinigung $V = T \cup N$ heißt das **Vokabular** der Grammatik G.
- Eine Zeichenreihe x aus V^*, die durch endlich viele Anwendungen von Regeln aus dem Ziel Z abgeleitet werden kann, heißt **Satzform** oder **Phrase** von G.
  - Satzformen können also terminale **und** nichtterminale Zeichen enthalten.
- Satzformen von G, die **nur aus terminalen** Zeichen bestehen, heißen **Sätze** von G.
- $L(G)$, die Menge der Sätze von G, heißt **Sprachschatz** von G.

Einfaches Beispiel einer Chomsky Grammatik

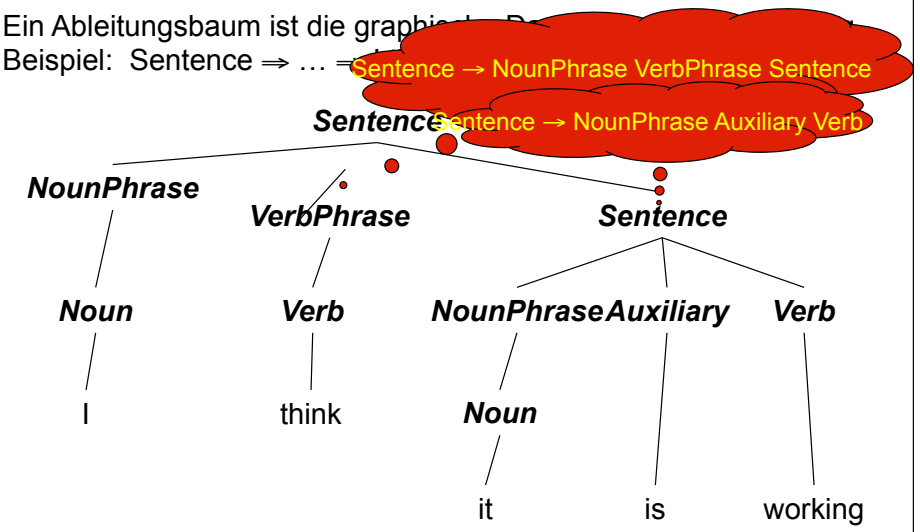
- Gegeben sei eine Chomsky-Grammatik $G_A = (T, N, P, Z)$ mit
- **Nichtterminale N** = {Sentence, NounPhrase, VerbPhrase, Noun, Verb, Auxiliary}
- **Terminale T** = {I, is, it, think, working}
- **Axiom Z** = Sentence
- **Produktionen P** = {

  Sentence → NounPhrase VerbPhrase Sentence
  Sentence → NounPhrase Auxiliary Verb
  NounPhrase → Noun
  VerbPhrase → Verb
  Verb → think
  Verb → working
  Noun → I
  Noun → it
  Auxiliary → is

  }

Ableitungsbaum

Ein Ableitungsbaum ist die graphi[illegible]
Beispiel: Sentence ⇒ … =



Ein etwas komplizierteres Beispiel

- Die Menge aller arithmetischen Ausdrücke mit den Bezeichnern a, b, c und den Operatoren + und * ist gegeben durch die folgende Chomsky-Grammatik $G_A = (T, N, P, Z)$ mit
- T= {a, b, c, +, *, (,)}
- N = {Ausdruck, Term, Faktor, Bezeichner}
- P = { Ausdruck → Term,
  Ausdruck → Ausdruck + Term,
  Term → Faktor,
  Term → Term * Faktor,
  Faktor → Bezeichner,
  Faktor → (Ausdruck),
  Bezeichner → a, Bezeichner → b, Bezeichner → c }
- *Z* = Ausdruck

*Beispiel: (a+b)*a+c ist ein Ausdruck*

Produktionen P =
{ Ausdruck → Term -- 1
Ausdruck → Ausdruck + Term -- 2
Term → Faktor -- 3
Term → Term * Faktor -- 4
Faktor → Bezeichner -- 5
Faktor → (Ausdruck) -- 6
Bezeichner → a -- 7
Bezeichner → b -- 8
Bezeichner → c -- 9
}

Axiom *Z* = Ausdruck

Terminale T = {a, b, c, (,), +, *}

Nichtterminale N = {Ausdruck, Term, Faktor, Bezeichner}

Vokabular V = {a, b, c, (,), +, *, Ausdruck, Term, Faktor, Bezeichner}

- Abkürzungen: A = Ausdruck, T = Term, F = Faktor, B = Bezeichner

A ⇒ A+T -- 2
⇒ T+T -- 1
⇒ T*F+T -- 4
⇒ F*F+T -- 3
⇒ (A)*F+T -- 6
⇒ (A+T)*F+T -- 2
⇒ (T+T)*F+T -- 1
⇒ (F+T)*F+T -- 3
⇒ (B+T)*F+T -- 5
⇒ (B+F)*F+T -- 3
⇒ (B+B)*F+T -- 5
⇒ (B+B)*B+T -- 5
⇒ (B+B)*B+F -- 3
⇒ (B+B)*B+B -- 5
⇒ … ⇒ (a+b)*a+c -- 7,8,7,9

Kompaktere Notation für Grammatiken

- Grammatiken verwendet man zur Beschreibung der Syntax von Programmiersprachen. Um die Beschreibung kompakt zu halten, werden folgende Konventionen benutzt.
  - Statt → schreibt man ::=
  - Nichtterminale werden in <> eingeschlossen, z.B. <Verb>
  - Der senkrechte Strich "|" trennt rechte Seiten zur gleichen linken Seite von Produktionen.
- Beispiel: Die beiden Produktionen

  Verb → think

  Verb → working

  schreibt man als

  <Verb> ::= think | working
- Die Notation stammt von John Backus (1959) und wurde von Peter Naur bei der Definition der Grammatik für Algol60 eingesetzt. Deshalb wird sie Backus-Naur-Form (BNF) genannt.

Konvertierung von Produktionen in Backus-Naur-Form

P = { Sentence → NounPhrase VerbPhrase Sentence
Sentence → NounPhrase Auxiliary Verb
NounPhrase → Noun, VerbPhrase → Verb
Verb → think, Verb → working
Noun → I, Noun → it
Auxiliary → is
}

<Sentence>	::=	<NounPhrase> <VerbPhrase> <Sentence> |
		<NounPhrase> <Auxiliary> <Verb>
<NounPhrase>	::=	<Noun>
<VerbPhrase>	::=	<Verb>
<Verb>	::=	think | working
<Noun>	::=	I | it
<Auxiliary>	::=	is

Backus-Naur-Form für Ausdrücke

Produktionen

P = {
Ausdruck → Term
Ausdruck → Ausdruck + Term
Term → Faktor
Term → Term * Faktor
Faktor → Bezeichner
Faktor → (Ausdruck)
Bezeichner → a
Bezeichner → b
Bezeichner → c
}

Backus-Naur-Form

<Ausdruck> ::= <Term> | <Ausdruck> + <Term>
<Term> ::= <Faktor> | <Term> * <Faktor>
<Faktor> ::= <Bezeichner> | (<Ausdruck>)
<Bezeichner> ::= a | b | c

Erweiterungen in der Backus Naur Form

- Gleich nach Erscheinen der Backus-Naur Form wurden zusätzliche Notationen eingeführt:
  - […] bezeichnet optionale Teile auf einer rechten Seite
  - (…) umschließt eine Gruppe von Zeichen
  - Ein Stern * nach einem Zeichen oder einer Gruppe bezeichnet die optionale oder n-fache Wiederholung des Zeichens oder der Gruppe
  - Ein Pluszeichen + nach einem Zeichen oder einer Gruppe bezeichnet die n-fache Wiederholung des Zeichens oder der Gruppe.

Grammatik für Dezimalzahlen in der erweiterten Backus-Naur-Form

Beispiele: 0.34 3.4 -3.4

Erster Vorschlag für eine Grammatik:

<Ziffer> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<Zahl> ::= <Ziffer>* .<Ziffer>*

Ist 34 eine Dezimalzahl?

Iteration 1: **Punkt und Dezimalstellen sind optional**

<Ziffer> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<Zahl> ::= <Ziffer>* [.<Ziffer>*]

Ist .34 eine Dezimalzahl?

Iteration 2: **Vor dem Punkt muss mindestens eine Ziffer stehen**

<Ziffer> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<Zahl> ::= <Ziffer>$^+$ [.<Ziffer>*]

Weitere Iterationen ...

Ist 4. eine Dezimalzahl?

Iteration 3: **Nach dem Punkt muss mindestens auch eine Ziffer stehen**

<Ziffer> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<Zahl> ::= <Ziffer>$^+$ [.<Ziffer>$^+$]

Wie ist es mit -3.4?

Iteration 4: **Wir erlauben negative Dezimalzahlen**

<Ziffer> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<Zahl> ::= [-] <Ziffer>$^+$ [.<Ziffer>$^+$]

Sind 0 oder 03.4 oder -03.04 Dezimalzahlen?

Iteration 5: **Wir verbieten führende Nullen, indem wir zwischen Ziffern und positiven Ziffern unterscheiden:**

<Pziffer> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<Ziffer> ::= 0 | <Pziffer>

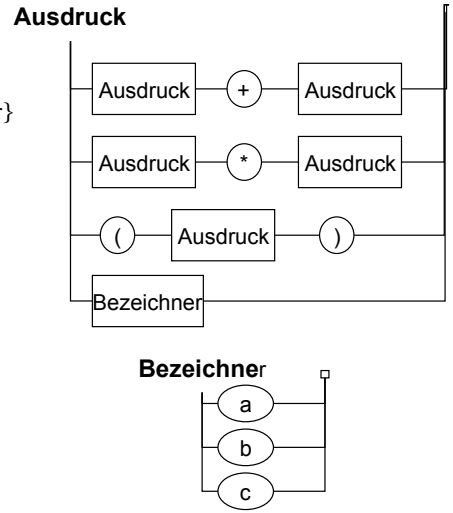
<Zahl> ::= [-] (0 |<Pziffer> <Ziffer>*)[.<Ziffer>$^+$]

Syntaxdiagramm

- Statt in BNF kann man die Produktionen auch graphisch durch Syntaxdiagramme darstellen.
- Jedes Diagramm hat einen Namen, nämlich das Nichtterminal, das es repräsentiert
  - Es hat genau einen Eingang und einen Ausgang.
  - Jeder Weg vom Eingang zum Ausgang ergibt eine gültige Ableitung.
- Eine Syntaxdiagramm besteht aus Ovalen (Kreisen) und Rechtecken, verbunden durch Kanten.
  - Ovale oder Kreise bezeichnen Terminale.
  - Rechtecke bezeichnen Nichtterminale.

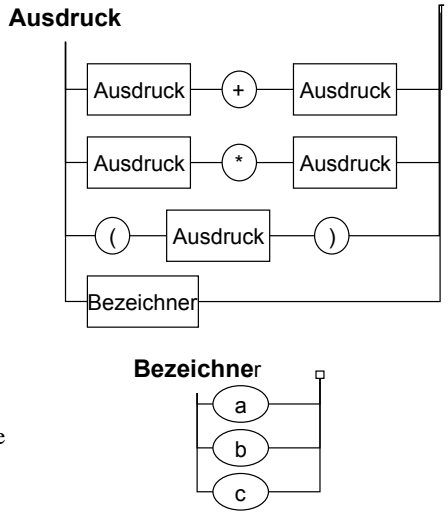
Beispiel: Konvertierung von Produktionen in ein Syntaxdiagramm

- $G_A = (T, N, P, Z)$
- T= {a, b, c, +, *}
- N = {Ausdruck, Term, Faktor, Bezeichner}
- P = { Ausdruck → Term,
  Ausdruck → Ausdruck + Term,
  Term → Faktor,
  Term → Term * Faktor,
  Faktor → Bezeichner,
  Faktor → (Ausdruck)
  Bezeichner → a
  Bezeichner → b
  Bezeichner → c }
- Z = Ausdruck



Eine einfachere Grammatik G'$_A$ für arithmetische Ausdrücke

- ***G'$_A$ = (T, N, P, Z)***
- T= {a, b, c, +, *}
- *N* = {Ausdruck, Bezeichner}
- P = { Ausdruck → Ausdruck + Ausdruck,
  Ausdruck → Ausdruck * Ausdruck,
  Ausdruck → (Ausdruck),
  Ausdruck → Bezeichner,
  Bezeichner → a
  Bezeichner → b
  Bezeichner → c }
- *Z* = Ausdruck



- Der **Sprachschatz** ist derselbe.
- **Strukturinformation** ging verloren (keine Unterscheidung von Ausdruck, Term und Faktor).

Zusammenfassung

- Der Begriff des Algorithmus ist eine der wichtigsten Säulen der Informatik (Leider gibt es mehrere Definitionen).
- Klassifikation von Algorithmen nach Anwendbarkeit der Regeln und nach der Beziehung zwischen Ein- und Ausgabe
- Textersetzungsysteme
- Markov-Algorithmen als Beispiel für deterministische Algorithmen
- Definitionen: Grammatik
- Die Backus-Naur-Form und Syntaxdiagramme sind nützlich für die kompakte Beschreibung von Grammatiken