Theoretische Informatik

Grundlegende Kursinformationen

Tutorium: Mittwoch, 16:15 – 18:00, SR 2 (ICT), Christina Kohl

Klausur: 10. Februar 2017

Inhalte:

- Aussagenlogik
- Schaltkreise
- Grammatiken
- Chomsky-Hierarchie
- Theorie der Formalen Sprachen
- Formale Modelle
- Berechenbarkeitstheorie
- Gleichungslogik
- Programmverifikation

Inhaltsverzeichnis

Einführung in die Logik

Eine Deduktion (syllogismos) ist ein Argument, in welchem sich, wenn etwas gesetzt wurde, etwas Anderes als das Gesetzte, mit Notwendigkeit durch das Gesetzte, ergibt.

Modus Ponens

Wenn A, dann B
A gilt
Also, gilt B

$$\frac{A \to B \qquad A}{B}$$

Formal wird der Modus Ponens mit dem Ableitungsoperator „ „ als Schlussregel $A, A \to B \vdash B$ notiert.

$$(A \wedge (A \to B)) \to B$$

Wenn das Kind schreit, hat es Hunger,

Das Kind schreit.

Also, hat das Kind Hunger.

Voraussetzungen und grundlegende Definitionen

Wahrheitswertkonstanten

True *False*

Junktoren

$\neg$	$\wedge$	$\vee$	$\rightarrow$
Negation	*AND*	*OR*	*Implikation*

Formeln der Aussagenlogik sind induktiv [1] definiert:

1. Eine atomare Formel *p* ist eine Formel
2. Ein Wahrheitswertsymbol ist eine Formel
3. Und wenn A und B Formeln sind, dann sind auch
   $\neg A$ $(A \wedge B)$ $(A \vee B)$ $(A \rightarrow B)$
   auch Formeln.

Präzedenzen

$$\neg > \vee, \wedge > \rightarrow$$

Gemäß der allgemeinen Konvention ist $\rightarrow$ rechts-assoziativ: $p \rightarrow (q \rightarrow r)$

Wahrheitswerte

Wahrheitswerte sind keine Wahrheitswertkonstanten!
True ist immer T, man kann gemäß der Definition aber T auch False zuweisen...

Belegung v

$v : AT \rightarrow \{T,F\}$	damit werden den Atomen [2] AT die Wahrheitswerte zugewiesen, wobei AT entweder T oder F annehmen kann.	
$\bar{v} : \bar{v}(p) = v(p)$	$\bar{v}$ (*True*) = v (T)	$\bar{v}$ wandelt den Wahrheitswert einer Aussage in die entsprechende Wahrheitswertkonstante um.

Syntax und Semantik

Unter einer Syntax verstehen wir Regeln, nach denen wir Texte strukturieren dürfen.

Semantik auf der anderen Seite ist die Zuordnung von Bedeutungen zu einem Text.

[1] Induktiv bedeutet, vom Einzelnen auf das Allgemeine zu folgern

[2] Atome sind Platzhalter für konkrete Aussagen (welche durch Junktoren verbunden werden können)

Tautologie und Erfüllbarkeit

Wenn eine Belegung v existiert, sodass $\bar{v}$ (A) = True, dann ist A **erfüllbar**.

Ist also **mindestens eine Belegung** der Aussage **A wahr**, ist A **erfüllbar**.

Existiert eine solche Belegung nicht, sprich **alle Belegungen** von **A** sind **False**, dann heißt A **unerfüllbar**.

Wenn **alle Belegungen** von **A gleich True** ($\bar{v}$ (A) = T), dann ist **A** eine **Tautologie**.

Eine **Formel A** ist eine **Tautologie gdw. ¬A unerfüllbar**.

Grundlegende Regeln

Assoziativität

Das **Assoziativgesetz** beschreibt, dass Operationen auch in **unterschiedlicher Reihenfolge** ausgeführt, zum **gleichen Resultat** führen. Das bedeutet, die **Klammerung ist irrelevant**.

Eine binäre Verknüpfung $\circ: A \times A \to A$ auf einer Menge A heißt assoziativ, wenn für alle a , b , c ∈ A das Assoziativgesetz

$$a \circ (b \circ c) = (a \circ b) \circ c$$

gilt.

Kommutativgesetz

Es beschreibt, dass alle **Argumente einer Operation vertauscht** werden können, **ohne** dass sich das **Resultat verändert**.

Eine binäre Verknüpfung $*: A \times A \to X,\ (a,b) \mapsto a * b$ heißt **kommutativ**, wenn für alle a, b ∈ A die Gleichheit **a ∗ b = b ∗ a** gilt.

Distributivgesetz

Auf einer Menge A seien zwei zweistellige Verknüpfungen ⋄ : A × A → A und ∗ : A × A → A definiert. Die Verknüpfung ∗ heißt

- **linksdistributiv** über ⋄ , wenn für alle a, b, c ∈ A gilt:
  a∗(b⋄c) = (a∗b)⋄(a∗c)
- **rechtsdistributiv** über ⋄ , wenn für alle a, b, c ∈ A gilt:
  (a⋄b)∗c = (a∗c)⋄(b∗c)
- **distributiv** über ⋄ , wenn sie links- und rechtsdistributiv über ⋄ ist.

Wahrheitstabelle

Eine Wahrheitstabelle listet alle relevanten Belegungen v, zusammen mit dem Wahrheitswert $\bar{v}$ (A). Wenn die Formel A aus n verschiedenen Atomen zusammengesetzt ist, hat die Wahrheitstabelle für A maximal 2^n Zeilen. Bei kleineren Formeln ist dieses Verfahren folglich sehr übersichtlich und recht schnell.

Methode von Quine

Dabei werden die Atome in einer Formel sukzessive durch True[1] bzw. False[1] ersetzt, bis sich elementare Äquivalenzen *(siehe nächste Seite)* ergeben. Trotz ihrer Ineffizienz ist sie weitgehend auch für große Formeln verwendbar.

Konsequenzrelation

$\bar{v}\ \{ A_1 , \dots , A_n \} \models B$

B gilt genau dann, wenn jede Belegung von $\bar{v}\ (A_1) = T, \dots , \bar{v}\ (A_n) = T$

Wenn die Konsequenz einer leeren Menge A ($\{ \} \models B$) ist eine Tautologie, bzw. gültig.

Semantisch gesehen, bezeugt die Konsequenzrelation, dass aus der Wahrheit der Prämissen A_1 bis A_n die Wahrheit der Konklusion B folgt.

Äquivalenz

Zwei Formeln A, B sind äquivalent, wenn $A \models B$ und $B \models A$ gilt. Wir schreiben dann

$$A \equiv B$$

Im folgenden Dokument wird für die Äquivalenz jedoch „ ≅ „ verwendet.

Elementare Äquivalenzen und Umformungen

$$\neg\neg A \equiv A$$

$$A \vee \text{True} \equiv \text{True} \qquad A \vee \text{False} \equiv A \qquad A \vee A \equiv A \qquad A \vee \neg A \equiv \text{True}$$

$$A \wedge \text{True} \equiv A \qquad A \wedge \text{False} \equiv \text{False} \qquad A \wedge A \equiv A \qquad A \wedge \neg A \equiv \text{False}$$

$$A \to \text{True} \equiv \text{True} \qquad A \to \text{False} \equiv \neg A \qquad \text{True} \to A \equiv A \qquad \text{False} \to A \equiv \text{True} \qquad A \to A \equiv \text{True}$$

Umformung von Junktoren

$$A \to B \equiv \neg A \vee B \qquad \neg(A \to B) \equiv A \wedge \neg B$$

$$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C) \qquad A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$$

Absorptionsgesetz

$$A \wedge (A \vee B) \equiv A \qquad A \vee (A \wedge B) \equiv A$$

$$A \wedge (\neg A \vee B) \equiv A \wedge B \qquad A \vee (\neg A \wedge B) \equiv A \vee B$$

Gesetze von de Morgan

$$\neg(A \wedge B) \equiv \neg A \vee \neg B \qquad \neg(A \vee B) \equiv \neg A \wedge \neg B$$

[1] Als Wahrheitskonstanten

Formales Beweisen

Ein Beweis wird oft auch als Ableitung oder Deduktion bezeichnet.

Eine Formel heißt beweisbar aus den Annahmen B, wenn es einen Beweis von A aus B gibt.

Axiome

$A \to (B \to C)$

$(A \to (B \to C)) \to ((A \to B) \ (A \to C))$

$(\neg A \to \neg B) \to (B \to A)$

$$p \to q \equiv \neg p \vee q$$

$$\neg(p \to q) = p \wedge \neg q$$

Beweisbarkeitsrelation

$\{ A_1 , \dots , A_n \} \vdash B$

Wir nennen ein System des formalen Beweisens **korrekt**,
wenn aus $\{ A_1 , \dots , A_n \} \vdash B$ auch $\{ A_1 , \dots , A_n \} \models B$ folgt.

Vollständig heißt es, wenn aus $\{ A_1 , \dots , A_n \} \models B$ auch $\{ A_1 , \dots , A_n \} \vdash B$ folgt.

Der Unterschied zwischen Beweisbarkeitsrelation und Konsequenzrelation liegt darin, dass Beweisbarkeitsrelation mehr syntaktischen, während Konsequenzrelation eher semantischen Charakter besitzt.

In einem korrekten und vollständigen Beweissystem sind beide folglich äquivalent.

Deduktion

Eine Deduktion ist ein Beweis, wobei eine Formel F beweisbar aus den Annahmen G heißt, wenn es einen Beweis/eine Deduktion von F aus G gibt.

Ein Beweis von F aus G wiederrum ist eine Sequenz von Formeln A_1, ..., A_n = F, sodass für i = 1, ..., n gilt:

entweder $A_i \in G$
oder A_i ist eines der Axiome oben,
oder A_i folgt mittels Modus Ponens aus den Formeln A_{i1} und A_{i2}, wobei i1 und i2 < i

Deduktionstheorem

Wenn $A_1, ..., A, ..., A_n \vdash B$ gilt, dann auch $A_1, ..., A_n \vdash A \rightarrow B$

Das bedeutet, wenn A eine Prämisse in einem Beweis von B ist, dann existiert ein Beweis von $A \rightarrow B$, der A nicht als Prämisse hat.

Mit dem Deduktionstheorem erleichtern wir unsere formale Argumentation enorm.

Axiomatisierbar

Eine Logik heißt endlich axiomatisierbar, wenn es eine endliche Menge von Axiomen und Inferenzregeln gibt, die korrekt und vollständig für diese Logik sind.

Konjunktive und Disjunktive Normalform

Sei A eine Formel. Ein Literal ist eine Boolesche Variable x oder ihre Negation.

Konjunktive Normalform (KNF)

A ist in konjunktiver Normalform, wenn A eine Konjunktion von Disjunktionen von Literalen ist. (*Auch*: Wenn A das Produkt von Summentermen ist)

Disjunktive Normalform (DNF)

A ist in disjunktiver Normalform, wenn A eine Disjunktion von Konjunktionen von Literalen ist. (*Auch*: Wenn A die Summe von Produkttermen ist)

Für jede Formel A existiert eine DNF D und eine KNF K, sodass
$A = K = D$

Die Algebra

Algebren erlauben uns eine abstrakte Beschreibung von Objekten durch Operationen mit diesen Objekten.

Algebraische Strukturen

Algebra

Eine **Algebra $\mathcal{A} = \langle A_1, ..., A_n; \circ_1, ..., \circ_2 \rangle$** ist eine **Struktur**, welche aus der **Menge** (auch **Träger** genannt) **$\{ A_1, ..., A_n \}$** und den **Operationen $\circ_1, ..., \circ_2$** auf diesen besteht. **Nullstellige Operationen** nennt man dabei **Konstanten**.

Sätze zur Algebra

ALGEBRA	BEZEICHNUNG	BEDINGUNG
<A, ∘>	Halbgruppe	Wenn ∘ assoziativ
<A, ∘, 1>	Monoid	Wenn <A, ∘> eine Halbgruppe und 1 ein neutrales Element für ∘
<A, ∘, 1>	Gruppe	Wenn <A, ∘, 1> ein Monoid und jedes Element ein Inverses besitzt
<A, +, *, 0>	Ring	Wenn <A, +, 0> eine kommutative Gruppe und <A, *, 1> ein Monoid. Außerdem muss * über + distribuieren.
<A, +, *, 0>	Körper	Wenn A ein Ring ist und <A\{0}; *, 1> eine kommutative Gruppe

Wenn <A, ∘, 1> ein Monoid ist, dann ist das Inverse eindeutig.

Nullteilerfrei bedeutet, es gibt keine Elemente a, b, ∈ A gibt, sodass a*b=0, aber a ≠ 0 und b ≠ 0.

Algebraischer Ausdruck

Mit eigenen Worten ausgedrückt: **Ein algebraischer Ausdruck ist ein Term**, also Summen und Produkte von Variablen und Konstanten.

Wir definieren allerdings einen algebraischen Ausdruck **induktiv**, d.h.:

- **Konstanten und Variablen sind algebraische Ausdrücke**
- **Wenn ∘ eine Operation von *A* ist**, die *m*-Argumente hat, wobei alle Argumente $E_1, \dots, E_n$ algebraische Ausdrücke sind, **dann** ist **auch ∘(E_1, … , E_n) ein algebraischer Ausdruck.**

Konvention: Wenn möglich, verwenden wir die *Infix-Notation*, das heißt wir **schreiben a ∘ b, statt ∘(a,b)**

Algebraische Ausdrücke stellen eine textuelle Beschreibung bestimmter Objekte der Trägermenge dar.

Äquivalent sind zwei algebraische Ausdrücke genau dann, wenn „alle Instanzen der algebraischen Ausdrücke“ (hier E, F) nach Auswertung in der Algebra *A* den selben Wert annehmen. Das bedeutet, wir tauschen die Variablen in E und F in gleicher Weise durch Elemente aus der Trägermenge aus. Wir schreiben dann E ≈ F.

Binäre Algebra

Die Binäre ist ein Spezialfall der Booleschen Algebra, wobei für n=2 gilt.

B := {0,1}, wobei 0,1 ∈ **N**

<**B**; +, *, ~, 0, 1> und definieren die Operationen in der Operationstabelle.

+	1	0
1	1	1
0	1	0

·	1	0
1	1	0
0	0	0

~	
1	0
0	1

Binäre Operationen

Bezeichnung	**Definition**
Nullelement für ○	Wenn 0 ∈ A | ∀ a ∈ A: a ○ 0 = 0 ○ a = 0
Einselement bzw. Neutrales Element für ○	Wenn 1 ∈ A | ∀ a ∈ A: a ○ 1 = 1 ○ a = 1
Inverse bzw. Komplement von a	Wenn 1 = Einselement für ○ und für a ∈ A existiert b ∈ A | a ○ b = b ○ a = 1

Jede binäre Operation hat maximal ein neutrales Element.

Beweis: e & u sind neutrale Elemente für ○, wir zeigen e = u → e = e*u → e = u[1]

Boolesche Algebra

Die Boolesche Algebra ist eine spezielle algebraische Struktur, welche die Eigenschaften der logischen Operatoren UND, ODER, NICHT, sowie die Eigenschaften der mengentheoretischen Verknüpfungen Durchschnitt, Vereinigung und Komplement verinnerlichen.

Eine Algebra *B* = <*B* ; +, *, ~, 0, 1> heißt Boolesche Algebra, wenn gilt:

- **<*B*; +, 0>** und **<*B*; *, 1>** sind **kommutative Monoide**:

  Also, + ist assoziativ (=Halbgruppe) und 0 das neutrale Element für +
  Und, * ist assoziativ (=Halbgruppe) und 1 das neutrale Element für *
  Damit die beiden Monoide dann noch kommutativ sind, muss gelten:
  + ist kommutativ und * ist auch kommutativ

- Die Operationen + und * distribuieren **übereinander:**

  Es gilt für alle a, b, c ∈ *B* :
  a * (b + c) = (a * b) + (a * c) a + (b * c) = (a + b) * (a + c)

- Für alle a ∈ B gilt:

  a + ~(a) = 1 und a * ~(a) = 0
  ~(a) heißt also Komplement oder Negation von a

Konvention: * bindet stärker als +

Wir bezeichnen die Boolesche Algebra bezogen auf die Algebraischen Ausdrücke auch Boolesche Ausdrücke.

[1]Logisch oder? :^}

Beispiel für Boolesche Algebra

Die Mengenalgebra/Binäre Algebra ist eine Boolesche Algebra (Randinformation)

Sei **B** := {0, 1} und sei $\mathbf{B}^n$ das n-fache kartesische Produkt von **B**: $\mathbf{B}^n = \{(a_1, ..., a_n) \mid a_i \in \mathbf{B}\}$; wir betrachten

$$<\mathbf{B}^n; +, *, \sim, (0, ..., 0), (1, ..., 1)>$$

Komponentenweise Erweiterung

$$(a_1, \ldots, a_n) + (b_1, \ldots, b_n) = (a_1 + b_1, \ldots, a_n + b_n)$$
$$(a_1, \ldots, a_n) \cdot (b_1, \ldots, b_n) = (a_1 \cdot b_1, \ldots, a_n \cdot b_n)$$
$$\sim((a_1, \ldots, a_n)) = (\sim(a_1), \ldots, \sim(a_n))$$

Punktweise Erweiterung

$$(f + g)(a_1, \ldots, a_n) := f(a_1, \ldots, a_n) + g(a_1, \ldots, a_n)$$
$$(f \cdot g)(a_1, \ldots, a_n) := f(a_1, \ldots, a_n) \cdot g(a_1, \ldots, a_n)$$
$$\sim(f)(a_1, \ldots, a_n) := \sim(f(a_1, \ldots, a_n)) .$$

Gesetze Boolescher Algebren

Dualitätsprinzip

„Vertauschungsprinzip“ → Wir betrachten <B; +, *, ~, (0, ..., 0), (1, ..., 1)>

Wird jeder Operator + mit *, bzw. * mit +, sowie 0 und 1 vertauscht, gilt die Gleichheit.

Idempotenzgesetz

a * a = a a + a = a

Substitution

a * 0 = 0 a + 1 = 1

Absorptionsgesetz

a + ab = a a(a + b) = a

a + ~(a) * b = a + b a(~(a) + b) = ab

Eindeutigkeit des Komplements

Wenn a + b = 1 und a * b = 0, dann b = ~(a)

Involutionsgesetz

~(~(a)) = a

Gesetz von de Morgan

~(a + b) = ~(a) * ~(b) ~(a * b) = ~(a) + ~(b)

[1]Logisch oder? :^}

Algebra der Booleschen Funktionen

Auch die Algebra der Booleschen Funktionen ist eine Boolesche Algebra.

Folgende Algebra nennt man Algebra der n-stelligen Booleschen Funktionen:

Sei **Abb** die Menge der Abbildungen von $\mathbf{B}^n$ nach $\mathbf{B}^m$, wir betrachten

<**Abb**; +, *, ~, (0, ..., 0), (1, ..., 1)>

$(\mathbf{0},\dots,\mathbf{0})\colon (a_1,\dots,a_n) \mapsto (0,\dots,0)$
$(\mathbf{1},\dots,\mathbf{1})\colon (a_1,\dots,a_n) \mapsto (1,\dots,1)$
$(f+g)(a_1,\dots,a_n) = f(a_1,\dots,a_n) + g(a_1,\dots,a_n)$
$(f\cdot g)(a_1,\dots,a_n) = f(a_1,\dots,a_n)\cdot g(a_1,\dots,a_n)$
$\sim(f)(a_1,\dots,a_n) = \sim(f(a_1,\dots,a_n))$

Boolesche Funktion (Definition)

Sei $\mathbb{B} = \{0,1\}$ und sei $\mathbb{B}^n$ das n-fache kartesische Produkt von $\mathbb{B}$

Definition (Boolesche Funktion)

1. Sei F ein Boolescher Ausdruck in den Variablen $x_1,\dots,x_n$
2. $F(s_1,\dots,s_n)$ die Instanz von F
3. Wir definieren die Funktion $f\colon \mathbb{B}^n \to \mathbb{B}$ wie folgt:

$$f(s_1,\dots,s_n) := F(s_1,\dots,s_n)\ .$$

Dann heißt f die Boolesche Funktion zum Ausdruck F

Beispiel (Boolsche Algebra $\mathcal{F}rm = \langle \mathrm{Frm}; \vee, \wedge, \neg, \mathrm{False}, \mathrm{True}\rangle$)

Sei $F = x_1 \wedge \neg(x_2 \vee x_1)$, dann ist $f\colon \mathbb{B}^2 \to Bin$ die Boolsche Funktion zu F

Sei $G = x_1 \wedge x_2 \wedge \neg x_2$, dann ist $g\colon \mathbb{B}^2 \to Bin$ die Boolsche Funktion zu G

s_1	s_2	$f(s_1,s_2)$	$g(s_1,s_2)$
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	0

Jede Boolesche Algebra ist isomorph zu einer Mengenalgebra

In der Mathematik ist ein Isomorphismus eine Abbildung zwischen zwei mathematischen Strukturen, durch die Teile einer Struktur auf „bedeutungsgleiche“ Teile einer anderen Struktur umkehrbar eindeutig abgebildet werden.

[1]Logisch oder? :^}

Universelle Algebra

Begriff	Erklärung
Signatur	Eine Signatur F ist eine Menge von Funktionssymbolen, sodass jedem Symbol $f \in F$ eine Stelligkeit n zugeordnet wird. Symbole mit **Stelligkeit 0** werden **Konstanten** genannt.
Term	Sei F eine Signatur und V eine (unendliche) Menge an Variablen, sodass $F \cap V = \emptyset$, dann ist die Menge T(F, V) aller Terme induktiv definiert: 1. Jedes Element von V ist ein Term 2. Wenn $f \in F$ mit Stelligkeit n sowie $t_1, ..., t_n$ Terme sind, dann ist auch $f(t_1, ..., t_n)$ ein Term.

Substitution

Eine Substitution ordnet einer Variable, bzw. einem Term, bzw. einem Ausdruck einen neuen Term/Ausdruck zu. Dadurch lässt sich ein Ausgangsterm in einen einfacher lösbaren Term umwandeln, dessen Ergebnis durch „Rück-substitutionierung“ auch dem tatsächlichen Ergebnis des Ausgangsterms entspricht.

Man ersetzt schwierige Terme durch Einfache.

Gleichung / Identität

Bei Identitäten geht es nicht darum, die Gleichung zu lösen, sondern es gilt eher mehr zur Definition. Das Resultat beider Seiten einer Identität sind.... Ident.

D.h. Egal welchen Wert des Definitionsbereichs ich in die Identität einsetze, die Gleichung ist immer korrekt.

Dieses Wissen/diese Identitäten sind wichtig, damit wir Ausgangsterme vereinfachen können (weil wir ja dann wissen was wir mit identen Termen ersetzen können).

Inferenzregeln

Reflexität [r]

$$\frac{}{E \vdash t \approx t}$$

Reflexivität ist logisch. a = a

Wir brauchen Sie, um „das linke x“ „dem rechten x“ „gleichzustellen“. Nur dadurch dürfen wir Schlussfolgern.

Symmetrie [s]

$$\frac{E \vdash s \approx t}{E \vdash t \approx s}$$

Damit darf man die Gleichung symmetrisch vertauschen (links → rechts & umgekehrt)

Transivität [t]

$$\frac{E \vdash s \approx t \quad E \vdash t \approx u}{E \vdash s \approx u}$$

Wir haben die ursprüngliche Schlussfolgerung **s = u**. Durch „herleiten“ (einsetzen) mithilfe der Identitäten können wir **s = t** und **t = u** folgern. Können wir dann noch diese beiden Folgerungen mit den anderen Inferenzregeln beweisen, haben wir bewiesen, dass auch **s = u**. Cool oder?

Anwendung [a]

$$\frac{s \approx t \in E}{E \vdash s \approx t}$$

Mithilfe der Anwendung können wir ein Element aus der Menge der Identitäten schlussfolgern.

Instanziierung [i]

$$\frac{E \vdash s \approx t}{E \vdash \sigma(s) \approx \sigma(t)}\ \sigma \text{ eine Substitution}$$

Wenn eine Substitution gegeben ist, können wir durch Instanziierung diese „Ersetzung“ instanziieren. (Beachte, Instanzen müssen auch „verworfen“ werden, also wieder „rücksubstitutioniert“ werden, um auf das korrekte Ergebnis zu kommen).

Kontext [k]

$$\frac{E \vdash s_1 \approx t_1 \quad \dots \quad E \vdash s_n \approx t_n}{E \vdash f(s_1, \dots, s_n) \approx f(t_1, \dots, t_n)}$$

Kontext dient zur Verwendung der Operationen. Zwei Elemente aus den Identitäten/deren Schlussfolgerungen können durch [k] mit den Operationen aus der Signatur verknüpft werden.

Ablauf zum Beweisen einer Schlussfolgerung aus Identitäten

Gegeben sind die Identitäten, Inferenzregeln, die Signatur und die Schlussfolgerung:

Wir betrachten als erstes die Schlussfolgerung und versuchen diese mithilfe der Identitäten „herzuleiten“. Das heißt, wir setzen in der schlussgefolgerten Gleichung unsere Identitäten ein, bis wir von links nach rechts kommen.

Danach drücken wir unsere Herleitung formal mithilfe der Transivität [t] aus.

Sobald wir dann also eine Formale Gleichungslogik haben, können wir die anderen Inferenzregeln verwenden, um unsere Schlussfolgerung zu beweisen.

Homomorphismus

Ein Homomorphismus ist eine Abbildung von A nach B, sodass für alle f ∈ F mit Stelligkeit n gilt:

$\varphi(f_A(a_1, ..., a_n) = f_B(\varphi(a_1), ..., \varphi(a_n))$

wobei $a_1, ..., a_n \in A$

Bijektive Homomorphismen heißen Isomorphismen

Satz von Birkhoff

Sei E eine Menge von Identitäten, dann sind die beiden Relationen $=_E$ und $\approx_E$ gleich, das heißt, für beliebige Terme s und t gilt: $s =_E t$ und $s \approx_E t$.

Formale Sprache

Alphabet, Wörter und Sprache

Wortmonoid

Ein Wortmonoid ist ein normales Monoid. D.h. es gilt die Assoziativität und es gibt ein neutrales Element.

Ein Wortmonoid ist eine Algebra < Σ* ; * , ∈ >, deren Konkatenation assoziativ ist und ein neutrales Element (das Leerwort ∈) besitzt.

Konkatenation ist die „Verkettung“, welche die Operation * in der Algebra definiert. Formal ausgedrückt:

Sei $x = a_1, a_2, ..., a_n$ und $y = b_1, b_2, ..., b_m$, dann gilt:

$$x * y = a_1 a_2 ... a_n b_1 b_2 ... b_m$$

Aber was ist „Σ*“? Σ sei ein Alphabet, dann können wir **die Menge aller Wörter einer bestimmten Länge** über Σ mithilfe einer Potenznotation bezeichnen.

Wir definieren also Σ^k als die Menge aller Wörter der Länge k, deren Symbole aus Σ stammen.

Mit diesem Wissen können wir auch folgende Definitionen verwenden:

$$\Sigma^+ := \Sigma^1 \cup \Sigma^2 \cup \cdots$$
$$\Sigma^* := \Sigma^+ \cup \{\epsilon\}$$

Alphabet

Selbsterklärend, eine endliche, nicht leere Menge von Symbolen, mit denen wir Wörter und damit Sätze bauen können. Sei w ein Wort, ist |w| die Länge des Wortes.

Formale Sprache

Eine Teilmenge L von Σ* heißt eine formale Sprache über dem Alphabet Σ.

Vereinigung

Diese ist wie in der Mengenlehre definiert:

$$L \cup M := \{x \mid x \in L \text{ oder } x \in M\}$$

Komplement

$$\sim L = \Sigma^* \setminus L := \{x \in \Sigma^* \mid x \notin L\}$$

Durchschnitt

$$L \cap M := \{x \mid x \in L \text{ und } x \in M\}$$

Produkt/Konkatenation

Wie oben schon erläutert:

$$LM := \{xy \mid x \in L \text{ und } y \in M\}$$

Schlussfolgerungen aus Wortmonoid und Definitionen

Assoziativität

$$(L_1L_2)L_3 = L_1(L_2L_3)$$

Neutrales Element

$$L\{\epsilon\} = \{\epsilon\}L = L$$

Potenznotation

$$L^k := \begin{cases} \{\epsilon\} & \text{falls } k = 0 \\ \underbrace{LL\cdots L}_{k\text{-mal}} & \text{falls } k \geqslant 1 \end{cases}$$

Kleene-Stern (Abschluss)

$$L^* := \bigcup_{k \geqslant 0} L^k = \{x_1 \cdots x_k \mid x_1, \ldots, x_k \in L \text{ und } k \in \mathbb{N}, k \geqslant 0\}$$

$$L^+ := \bigcup_{k \geqslant 1} L^k = \{x_1 \cdots x_k \mid x_1, \ldots, x_k \in L \text{ und } k \in \mathbb{N}, k \geqslant 1\}$$

Grammatiken

Eine Grammatik ist ein Quadrupel G = (V, Σ, R, S)

- **V** ist eine endliche Menge an Variablen (auch **Nichtterminale** genannt)
- **Σ** ist das Alphabet, auch **Terminale** genannt
- **R** ist die endliche Menge an **Regeln**
- **S** ∈ **V** ist das **Startsymbol** von G

Terminale, weil sie die Grammatik beenden

*Nichtterminale hingegen werden durch weitere (Nicht-)Terminale entsprechend den Regeln **R** ersetzt werden.*

$$V \cap \Sigma = \varnothing$$

Aber was ist eine Regel?

Eine Regel ist ein Paar P → Q von Wörtern, sodass P, Q ∈ (V ∪ Σ)* und in P mindestens eine Variable vorkommt.

Check'sch?

Also kurz, die Regeln legen fest, wie die Nicht-Terminale verwendet werden können. z.B.:

B → 0 | 1 | 0B | 1B
K → ∈ | (K) | KK

wobei B, K ∈ V und „∈", 0, 1 ∈ Σ

Man sagt auch, **P** ist die **Prämisse** und **Q** die **Konklusion**.

Konventionen

Variablen (Nicht-Terminale) werden großgeschrieben, die Terminale aber klein.

Statt $P \to Q_1$, $P \to Q_2$, $P \to Q_3$ schreiben wir $P \to Q_1 \mid Q_2 \mid Q_3$

Definitionen

Direkt Ableitbar

Wir sagen y ist aus x in G direkt ableitbar, wenn gilt:

$$\exists\, u, v \in (V \cup \Sigma)^*,\ \exists\, (P \to Q) \in R \text{ sodass } (x = uPv \text{ und } y = uQv)$$

In diesem Fall schreiben wir $x \underset{G}{\Rightarrow} y$

Wenn G aus dem Kontext folgt schreiben wir $x \Rightarrow y$

Ableitbar

y ist aus x in G ableitbar, wenn k ∈ **N** und $w_0, \ldots, w_k \in (V \cup \Sigma)^*$, sodass

$x = w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_k \Rightarrow y$

Dann schreiben wir $x \overset{*}{\underset{G}{\Rightarrow}} y$, beziehungsweise $x \overset{*}{\Rightarrow} y$

Sprache einer Grammatik

Vom Startsymbol S ableitbare Wörter heißen Satzformen.

Elemente von Σ^* heißen Terminalwörter.
Sätze heißen alle Satzformen, die Terminalwörter sind.

Die **Menge aller Sätze heißt** die **von der Grammatik G erzeugte Sprache**:

$$L(G) = \{x \in \Sigma^* \mid S \overset{*}{\underset{G}{\Rightarrow}} x\}$$

Zwei **Grammatiken G_1 und G_2** heißen **äquivalent**, wenn $L(G_1) = L(G_2)$.

Klassen von Grammatiken

Im Folgenden gelten die Typen nur für die Formale Sprache, auf der die Grammatik angewandt wird. Sprich, gibt es eine **Grammatik G** und eine **Formale Sprache L**, sodass **L = L(G)**, dann hat L den Typ i, welcher rechts neben dem Typ der Grammatik steht.

Rechtslinear „(Typ 3)“

Eine Grammatik ist **rechtslinear**, wenn für alle Regeln $P \to Q$ gilt:

- $P \in V$
- $Q \in \Sigma^* \cup \Sigma^+ V$

Die Grammatik $G_1 = (\{B\}, \{0,1\}, R, B)$ ist rechtslinear, wobei R wie folgt definiert:

$$B \to 0 \mid 1 \mid 0B \mid 1B$$

Es gilt:

$$L(G_1) = \{0,1\}^+$$

Kontextfrei „(Typ 2)“

Eine Grammatik ist **kontextfrei**, wenn für alle Regeln $P \to Q$ gilt:

- $P \in V$
- $Q \in (V \cup \Sigma)^*$

Die Grammatik $G_2 = (\{K\}, \{(,)\}, R, K)$ ist kontextfrei, wobei R wie folgt definiert:

$$K \to \epsilon \mid (K) \mid KK$$

Es gilt:

$$K \Rightarrow KK \Rightarrow (K)K \Rightarrow (\epsilon)K = ()K \Rightarrow ()(K) \Rightarrow ()(KK) \overset{*}{\Rightarrow} ()(()(()))$$

Kontextsensitiv „(Typ 1)“

Eine Grammatik ist **kontextsensitiv**, wenn für alle Regeln $P \to Q$ gilt:

- Entweder existieren u, v, $w \in (V \cup \Sigma)^*$ und $A \in V$, sodass

  $$P = uAv \text{ und } Q = uwv \quad \text{wobei } |w| \geq 1$$

- Oder $P = S$ und $Q = \epsilon$

Sollte $S \to \epsilon \in G$, dann kommt S nicht in einer Konklusion vor.

$G_3 = (\{S, B, C, H\}, \{a, b, c\}, R, S)$ ist kontextsensitiv, wobei R:

$$\begin{array}{lllllllll} S & \to & aSBC \mid aBC & & HC & \to & BC & & bC & \to & bc \\ CB & \to & HB & & aB & \to & ab & & cC & \to & cc \\ HB & \to & HC & & bB & \to & bb & & & & \end{array}$$

$L(G_3) = \{a^n b^n c^n \mid n \geqslant 1\}$

beschränkt

Eine Grammatik ist **beschränkt**, wenn für alle Regeln $P \to Q$ gilt:

- Entweder $|P| \leq |Q|$
- Oder $P = S$ und $Q = \epsilon$

Auch hier gilt, wenn $S \to \epsilon \in G$, dann kommt S nicht in einer Konklusion vor.

Typen

Nochmals in Erinnerung holen: Eine Grammatik hat keinen Typ, nur die formale Sprache mit einer Grammatik.

Ist sie rechtslinear, ist der Typ 3.

Formale Sprachen mit kontextfreier Grammatik den Typ 2,

mit kontextsensitiver Grammatik den Typ 1 und

wenn sie eine **beschränkte Grammatik** verwendet, nennt man die **Sprache ebenfalls beschränkt**.

Wer jetzt aufgepasst hat, hat bemerkt, es fehlt der Typ 0. Dieser ist nämlich praktisch alles. Wirklich. Einfach alles. **Jede Grammatik**, welche keinem Typ 1, 2, 3 zugewiesen werden kann, ist **rekursiv aufzählbar** und damit vom **Typ 0**. Auch die Typen 1, 2, 3 sind theoretisch vom Typ 0, bringt nur nix.

Eine Sprache L ist kontextsensitiv gdw. L beschränkt ist.

Chomsky-Hierarchie

Sie gibt die Hierarchie der Typen aller Klassen wieder. Grundsätzlich besagt sie, dass die Typ-0-Grammatik uneingeschränkt, mit steigender Stufe immer beschränkter wird.

Formal gelten also folgende Inklusionen:

$$\mathcal{L}_3 \subsetneq \mathcal{L}_2 \subsetneq \mathcal{L}_1 \subsetneq \mathcal{L}_0 \subsetneq \mathcal{L}$$

Wobei $\mathcal{L}$ die Klasse der Formalen Sprachen und L_i die Klasse vom Typ i ist.

a) $G_1 = (\{S, B\}, \{a, b, c\}, R_1, S)$

R1 : $S \to aaS \mid aacB$ $B \to bB \mid \epsilon$

Die erzeugte Sprache $L(G_1)$ ist vom **Typ 3**, weil ihre **Grammatik rechtslinear** ist, weil für alle Regeln gilt:

- $P \in V$
- $Q \in \Sigma^* \cup \Sigma^+ V$

Das heißt, jedes Nicht-Terminal wird durch beliebig viele oder ein einziges Element aus dem Alphabet (Terminalen) und einem abschließenden Nicht-Terminal ersetzt.

Beispiele:

S -> aaS -> aaaacB -> aaaacbB -> aaaacb

S -> aacB -> aac

b) $G_2 = (\{S\}, \{a, b, c\}, R_2, S)$

R_2 : $S \rightarrow aaSb \mid aac$

Die formale Sprache $L(G_2)$ ist vom **Typ 2**, weil die Grammatik **kontextfrei** ist:

- $P \in V$
- $Q \in (V \cup \Sigma)^*$

P ist ein Element der Nichtterminale und Q ein Element von Nicht-Terminalen vereinigt mit Alphabet in beliebiger Länge und Reihenfolge.

Beispiel:

S -> aaSb -> aaaacb

c) $G_3 = (\{S, A, B\}, \{a, b, c\}, R_3, S)$

R_3 : $S \rightarrow aAacB$ $A \rightarrow aAa \mid \in$ $B \rightarrow Bb \mid \in$

Die formale Sprache $L(G_3)$ ist vom **Typ 1**, weil die Grammatik **kontextsensitiv** ist:

- Entweder existieren u, v, w $\in (V \cup \Sigma)^*$ und A $\in$ V, sodass

  $P = uAv$ und $Q = uwv$ wobei $|w| \geq 1$

- Oder P = S und Q = $\in$

S wird auf aAacB abgebildet, also uAuvw, weil w = B $\in (V \cup \Sigma)$. Für B gilt Q = $\in$

d) $G_4 = (\{S, A, B, C\}, \{a, b, c\}, R_4, S)$

R_4 : $S \rightarrow aaA$ $aaA \rightarrow aaaaA \mid aaC$ $aaC \rightarrow aaacB \mid aac$ $B \rightarrow b \mid bB$

Die formale Sprache $L(G_4)$ ist vom **Typ 1**, weil die Grammatik **kontextsensitiv** ist:

- Entweder existieren u, v, w $\in (V \cup \Sigma)^*$ und A $\in$ V, sodass

  $P = uAv$ und $Q = uwv$ wobei $|w| \geq 1$

- Oder P = S und Q = $\in$

Der Kontext bleibt stets erhalten.

e) $G_5 = (\{S, A, B\}, \{a, b, c\}, R_5, S)$

R_5 : $S \rightarrow aaS \mid aaB$ $A \rightarrow aaA \mid \in$ $aB \rightarrow aBb \mid ac$

Die formale Sprache $L(G_3)$ ist vom **Typ 1**, weil die Grammatik **kontextsensitiv** ist:

- ℘ Gleichen Gründe wie oben

A wird auf aaA oder $\in$ abgebildet und aB erhält immer den Kontext.

Reguläre Sprache

Eine Sprache ist Regulär, wenn sie von einer rechtslinearen Grammatik beschrieben wird. Sie sind die einfachste Klasse von Sprachen in der Chomsky-Hierarchie. Aufgrund ihrer enormen Bedeutung, können sie auch mit Hilfe von endlichen Automaten und regulären Ausdrücken untersucht werden.

Im Skriptum auf Seite 36 findet sich eine Einleitung zu den Deterministisch endlichen Automaten, empfehlenswert zum Lesen für ein besseres Verständnis des folgenden Kapitels.

Deterministischer endlicher Automat (DEA)

Grundsätzlich ist ein deterministischer endlicher Automat ein Quintupel $A = (Q, \Sigma, \delta, q_0, F)$, wobei

Q eine endliche Menge von Zuständen,
Σ eine endliche Menge von Eingabesymbolen, (wird auch Eingabealphabet genannt)
δ ist die Übergangsfunktion : $Q \times \Sigma \rightarrow Q$
$s \in Q$ der Startzustand
und $F \subseteq Q$ ist eine endliche Menge von akzeptierten Zuständen

Die Übergangsfunktion definiert, wie sich der Zustand des Automaten bei einer Eingabe ändert, daher ist es wichtig, **dass δ für alle möglichen Argumente definiert ist!** Wie bei der Wahrheitstabelle, kann eine Zustandstabelle dazu genutzt werden.

Alternativ bietet sich ein **Zustandsgraph** an, der ein gerichteter Graph ist, sodass

- Die Ecken die Zustände sind
- Für die Zustände p, q $\in$ Q sind die Kanten von p nach q alle Triple:

$$(p, a, q) \quad \text{mit} \quad a \in \Sigma \quad \text{und} \quad \delta(p, a) = q$$

Konvention:
Zu jeder Kante (p, a, q) die Eingabe a dazuschreiben, dann den Startzustand mit einem Pfeil markieren und die akzeptierten Zustände mit doppelten Kreis kennzeichnen

Wie ist die Sprache L(M) eines Automaten M definiert?

Zunächst müssen wir die erweiterte Übergangsfunktion $\delta^\wedge$ definieren: $\delta^\wedge = \hat{\delta}$

$\delta^\wedge := Q \times \Sigma^* \to Q$ ist induktiv.

$\delta^\wedge(q, \varepsilon) := q$

$\delta^\wedge(q, xa) := \delta(\delta^\wedge(q, x), a)$ $x \in \Sigma^*, a \in \Sigma$

$\delta^\wedge(p, x) = q$, dann sagen wir, der Automat liest das Wort x am Weg von Zustand p nach Zustand q. Damit können wir dann die Sprache definieren:

$$L(M) := \{ x \in \Sigma^* \mid \delta^\wedge(s, x) \in F \}$$

Das heißt, die Sprache von M ist die Menge der Zeichenreihen x, die vom Starzustand s in einen akzeptierenden Zustand führen, auch akzeptierte Sprache genannt.

Sätze zu DEAs

Für jeden DEA A ist L(A) regulär. Umgekehrt existiert zu jeder regulären Sprache L ein DEA A, sodass L = L(A)

Die Vereinigung $L \cup M$ zweier regulärer Sprachen L, M ist regulär.

Wenn L und M reguläre Sprachen sind, dann ist auch $L \cap M$ regulär.
(Beweis: $L \cap M = \sim\sim L \cup \sim M$, siehe oben [de Morgan])

Wenn L und M reguläre Sprachen sind, dann ist auch $L \setminus M$ regulär.

Sei L eine reguläre Sprache über dem Alphabet Σ, dann ist das Komplement $\sim L = \Sigma^* \setminus L$ ebenfalls regulär.

Konstruktion

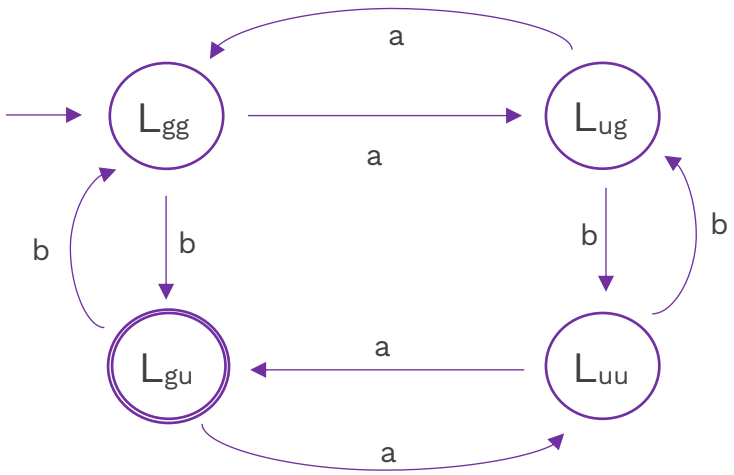
Folgend ein DEA A, der die Sprache L über dem Alphabet {a, b} akzeptiert, wobei L wie folgt definiert ist:

L = {x | x enthält eine gerade Anzahl von as und eine ungerade Anzahl von bs}

$Q = \{L_{gg}, L_{ug}, L_{gu}, L_{uu}\}$

$\Sigma = \{a, b\}$

$DEA := \{ Q, \Sigma, \delta, L_{gg}, F\}$



Zustandsgraph:

$(L_{gg}, a. L_{ug}), (L_{ug}, a, L_{gg})$ $(L_{gg,} b, L_{gu}), (L_{gu,} b, L_{gg})$

$(L_{ug,} b, L_{uu}), (L_{uu,} b, L_{ug})$ $(L_{uu,} a, L_{gu}), (L_{gu,} a, L_{uu})$

Tests:

ε: $\delta(L_{gg}, \varepsilon) = L_{gg} \neq L_{gu}$

abab: $\delta(L_{gg}, a) = L_{ug} \rightarrow \delta(L_{ug}, b) = L_{uu} \rightarrow \delta(L_{uu}, a) = L_{gu} \rightarrow \delta(L_{gu}, b) = L_{gg} \neq L_{gu}$

ababb: $\delta(L_{gg}, a) = L_{ug} \rightarrow \delta(L_{ug}, b) = L_{uu} \rightarrow \delta(L_{uu}, a) = L_{gu} \rightarrow \delta(L_{gu}, b) = L_{gg} \rightarrow \delta(L_{gg}, b) = \mathbf{L_{gu} == L_{gu}}$

Kontextfreie Sprache

Eine Sprache heißt kontextfrei, wenn sie von einer kontextfreien Grammatik beschrieben wird. Ehhhhhhhhhhhh

Palindrome

Induktive Definition:

1. ε, 0, 1 sind Palindrome
2. Wenn x ein Palindrom ist, dann ist auch 0x0 und 1x1 ein Palindrom

Die KFG G = ({P}, Σ, R, P), wobei R wie folgt definiert ist, beschreibt die Sprache der Palindrome.

$P \rightarrow \varepsilon \mid 0 \mid 1$

$P \rightarrow 0P0 \mid 1P1$

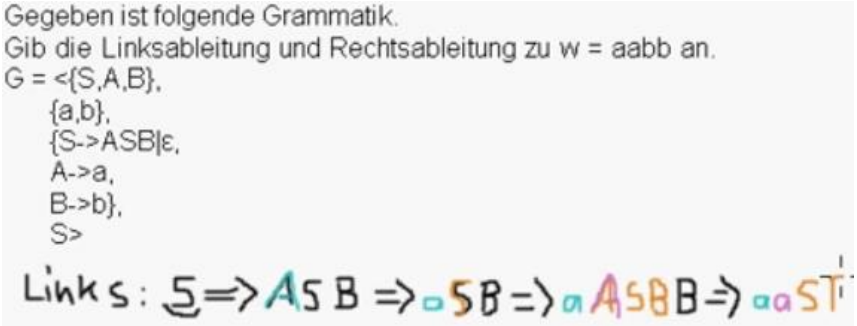
L(G) ist dann genau die Menge der Palindrome über dem Alphabet {0, 1}.

Definitionen

Ableitungen

Ableitungen sind, frei formuliert, einfach Anwendungen (ich möchte hier nicht Instanzen sagen) einer Grammatik. Also wenn wir vom Startsymbol die Regeln anwenden und irgendeine Ableitung erhalten, ist das ... tamtam ... eine Ableitung.

Linksableitungen ersetzen dabei sukzessive immer die linkste Regel,
die Rechtsableitung beginnt immer ganz rechts nach links:



Linksableitungen nennen wir auch *Top-Down*
Rechtsableitungen auch *Bottom-Up*

Eindeutigkeit

Eine Grammatik G heißt eindeutig, wenn jedes Wort $x \in L(G)$ genau eine Linksableitung besitzt, ansonsten nennt man G mehrdeutig.

Wir können eine Grammatik auch bottom-up auswerten

Rekursive Inferenz

Sie ist induktiv definiert:

Wenn $x_i \in L(X_i)$ oder $X_i = x_i \in \Sigma$, dann gilt $x_1x_2 \dots x_n \in L(A)$

Syntaxbaum

Sei $G = (V, \Sigma, R, S)$ eine KFG (kontextfreie Grammatik)

Der Syntaxbaum für G ist ein Baum B, sodass die folgenden Bedingungen gelten:

1. Jeder innere Knoten von B ist eine Variable von V
2. Jedes Blatt in B ist entweder ein Terminal aus Σ, ein Nichtterminal aus V oder ε.
   Wenn das Blatt ε ist, dann ist dieser Knoten das einzige Kind seines Vorgängers.
3. Sei A ein innerer Knoten, $X_1, \dots, X_n$ seine Kinder, dann ist $A \to X_1, \dots, X_n \in R$

Ergebnis eines Syntaxbaumes

Sei G = G von oben

Dann ist das Ergebnis eines Syntaxbaums B für G das Wort über $(V \cup \Sigma)^*$, welches wir erhalten, wenn wir die Blätter in S von links nach rechts lesen.

Sätze

Äquivalenz

Sei G = G von oben und $A \in V$ und $x \in \Sigma^*$

Dann sind folgende Beschreibungen kontextfreier Sprachen äquivalent:

$x \in L(A)$ nach dem rekursiven Inferenzverfahren

$A \overset{*}{\Rightarrow} x$

$A \overset{*}{\underset{l}{\Rightarrow}} x$ *Das l bedeutet Linksableitung*

$A \overset{*}{\underset{r}{\Rightarrow}} x$ *Rate, für was das r steht ☺*

Es existiert ein Syntaxbaum mit Wurzel A und Ergebnis x

XML

Sind Kontextfreie Sprachen – genauere Infos (weil wahrscheinlich nicht Prüfungsrelevant) im Skriptum ab Seite 46.

Berechenbarkeitstheorie

Beschäftigt sich grundlegend mit der Frage, ob Probleme algorithmisch lösbar sind und wenn ja, wie sie mit Hilfe einer Maschine lösbar sind.

Dabei heißt ein Problem **entscheidbar**, wenn es durch einen Algorithmus gelöst werden kann. Aber **nicht alle Probleme können algorithmisch** gelöst werden.

Wir stellen die These auf, dass jedes **algorithmisch lösbare Problem auch mit einer Turingmaschine lösbar** ist.

Unentscheidbare Probleme

Postsches Korrespondenzproblem

Das Postsche Korrespondenzproblem ist ein Entscheidungsproblem, bei dem eine Instanz dieses Problems aus einer Menge von Paaren $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ besteht, wobei alle x_i und y_i Wörter über einem gemeinsamen Alphabet sind. Die einzelnen Paare dieser Menge werden auch Dominos genannt.

Beispiel für das Problem:

Instanz:

01	10	1
0	11	011

Gibt es eine Möglichkeit, die Dominos so zu legen, dass eine Konkatenation der x_i's und der y_i's dasselbe Wort ergeben? Dabei darf jeder Domino beliebig oft wiederverwendet werden.

Lösung für das obige Beispiel:

01	10	01	1
0	11	0	011

Gibt oben und unten 0110011.

Turingmaschine

Grundlegende Begriffe

Begriff	Erklärung
Deterministisch	Bedeutet einfach, dass alle Ereignisse durch Vorbedingungen eindeutig festgelegt sind
	Mehr isses nit worden hahahaha

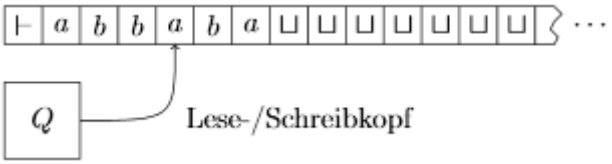
Definition

Eine deterministische, einbändige Turingmaschine M ist ein 9-Tupel:

$$M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$$

wobei

1. Q eine endliche Menge von **Zuständen**
2. $\Sigma \subseteq \Gamma$ eine endliche Menge von **Eingabesymbolen**
3. Γ eine endliche Menge von **Bandsymbolen**
4. $\vdash \in \Gamma \setminus \Sigma$ der linke **Endmarker**
5. $\sqcup \in \Gamma \setminus \Sigma$ das **Blanksymbol**
6. δ: Q x $\Gamma \to$ Q x Γ x {L, R} die **Übergangsfunktion**
7. s $\in$ Q der **Startzustand**
8. t $\in$ Q der **akzeptierende Zustand**
9. r $\in$ Q der **verwerfende Zustand** mit t $\neq$ r



Informell bedeutet $\delta(p, a) = (q, b, d)$: „Wenn die TM M im Zustand p das Symbol a liest, dann ersetzt M das Zeichen a durch das Zeichen b, der Lese-/Schreibkopf bewegt sich einen Schritt in die Richtung d und M wechselt in den Zustand q."

Das Symbol $\vdash$ darf niemals überschrieben werden (die Maschine kann also nicht über die linke Begrenzung hinausfahren). Formal lässt sich das durch die folgende Bedingung festlegen:

Für alle p $\in$ Q existiert ein q $\in$ Q mit:

$$\delta(p, \vdash) = (q, \vdash, R)$$

Zusätzlich brauchen wir noch einen akzeptierenden, bzw. verwerfenden Zustand, den die Maschine nicht mehr verlassen kann. Das heißt, für alle b $\in \Gamma$ existieren c, c‘ $\in \Gamma$ und d, d‘ $\in$ {L, R}, sodass:

$$\delta(t, b) = (t, c, d)$$

$$\delta(r, b) = (r, c‘, d‘)$$

Beachte, obwohl die TM in einen Zustand gerät, den sie nicht mehr verlassen kann, ist sie dennoch immer noch in Bewegung. Sie **hält**.

Wir nennen die Zustandsmenge Q und die Übergangsfunktion δ auch als die endliche Kontrolle von M.

Auch gut zu wissen, die TM ist **immer** unendlich lange, das heißt es gibt unendlich viele Blanksymbole $\sqcup$ (wir schreiben $y\sqcup^{\infty}$), wobei es dennoch endlich repräsentierbar ist, weil nur die Darstellung von y relevant ist und $|y| \in$ **N**.

Relation

Die Relation:

$$\xrightarrow[M]{1}$$

ist definiert:

$$(p, z, n) \xrightarrow[M]{1} \left\{ \begin{array}{l|l} (q, z', n-1) & wenn\ \delta(p, z_n) = (q, b, L) \\ (q, z', n+1) & wenn\ \delta(p, z_n) = (q, b, R) \end{array} \right\}$$

z‘ bezeichnet das Wort, welches wir aus z erhalten, wenn wir z_n durch b ersetzen.

Reflexive, transitive Hülle

Wir definieren die reflexive, transitive Hülle

$$\xrightarrow[M]{*} \text{ von } \xrightarrow[M]{1}$$

induktiv:

1. $\alpha \xrightarrow[M]{0} \alpha$
2. $\alpha \xrightarrow[M]{i+1} \beta$ wenn $\alpha \xrightarrow[M]{i} \gamma \xrightarrow[M]{1} \beta$ für eine Konfiguration γ und
3. $\alpha \xrightarrow[M]{*} \beta$ wenn $\alpha \xrightarrow[M]{i} \beta$ für ein $i \geq 0$

Wobei i die Induktionsannahme sei.

Konfiguration

Eine Konfiguration einer TM M ist das Tripel (p, z, n), sodass
$p \in Q$ der aktuelle Zustand,
$z = y\sqcup^\infty$ der aktuelle Bandinhalt ($y \in \Gamma^*$)
n eine natürliche Zahl die Position des Lese-/Schreibkopfs angibt

Die **Startkonfiguration** bei Eingabe $x \in \Sigma^*$ ist die Konfiguration

$$(s, \vdash x\sqcup^\infty, 0)$$

Sprache einer TM

Definition 4.5 (Sprache einer TM). Sei $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$ eine TM. Die Turingmaschine M *akzeptiert* die Eingabe $x \in \Sigma^*$, wenn gilt

$$(s, \vdash x\sqcup^\infty, 0) \xrightarrow[M]{*} (t, y, n) ,$$

für ein $y \in \Gamma^*$ und $n \in \mathbb{N}$. M *verwirft* x, wenn

$$(s, \vdash x\sqcup^\infty, 0) \xrightarrow[M]{*} (r, y, n) ,$$

für ein $y \in \Gamma^*$ und $n \in \mathbb{N}$. Wir sagen M *hält* bei Eingabe x, wenn M die Eingabe x entweder akzeptiert oder verwirft. Andernfalls *hält* M auf x *nicht*. Eine TM M heißt *total*, wenn sie auf allen Eingaben hält. Die Menge $\mathsf{L}(M)$ bezeichnet die Menge aller von M akzeptierten Wörter.

Turingmaschinen können die gleichen Sprachen beschreiben, wie Grammatiken.

Außerdem heißt eine Sprache L rekursiv, wenn es eine *totale* TM M gibt, mit L = L(M). Die Klasse der **rekursiven Sprachen** ist **größer** als die Klasse der **beschränkten Sprachen**, *aber* **kleiner** als die Klasse der **rekursiv aufzählbaren Sprachen**.

Also *rekursiv aufzählbare Sprachen > rekursive Sprachen > beschränkte Sprachen*

Registermaschinen

Um sie mit Turingmaschinen vergleichen zu können, definieren wir eine Funktion f: $\mathbf{N}^k \to \mathbf{N}$, das heißt, eine natürliche Zahl n wird auf dem Band der TM durch n Wiederholungen des Zeichens ⊔ dargestellt. Wir schreiben $\sqcup^n$

Eine Registermaschine ist eine Maschine, die eine endliche Anzahl von Registern besitzt (x_1, ..., x_n). Sie führt ein Programm P aus, dessen Befehle an eine stark vereinfachte imperative Sprache erinnern. Wir verwenden dafür sogenannte while-Programme, womit uns neben Zuweisungen auch bedingte Schleifenaufrufe zur Verfügung stehen.

Informell definiert:

$$R = ((x_i)_{1 \le i \le n}, P)$$

$((xi)_{1 \le i \le n}$ ist eine Sequenz von *n Registern* x_i und P ist ein Programm. Diese Programme wiederrum sind endliche Folgen von Befehlen und induktiv definiert:

1. Für jedes Register x_i sind die folgenden Instruktionen sowohl Befehle, als auch Programme:

   $x_i := x_i + 1$ $\qquad$ $x_i := x_i - 1$

2. Wenn P_1 und P_2 Programme sind, dann ist

   $P_1; P_2$

   ein Programm und

   while $x_i \neq 0$ do P_1 end

   ist sowohl ein Befehl als auch ein Programm.

Zusatzinformationen

Erwähnenswert ist, dass der Befehl $x_i := x_i - 1$ nur dann das Register dekrementiert, wenn $x_i > 0$ ist. Ansonsten verändert sich nichts.

$P_1; P_2$ bedeutet, dass zuerst P_1, dann P_2 ausgeführt wird.

Die Registermaschine *hält*, wenn es keinen weiteren auszuführenden Befehl mehr gibt.

Berechenbarkeit mit einer RM

Sei $R = ((x_i)_{1 \le i \le n}, P)$ eine Registermaschine und $f: \mathbf{N}^k \to \mathbf{N}$ eine partielle Funktion, dann heißt diese Funktion R-Berechenbar, wenn gilt

$f(n_1, ..., n_k) = m$ gdw. R mit n_i in den Registern x_i für $1 \le i \le k$ startet
und die Programmausführung mit n_i in den Registern x_i für $1 \le i \le k$
und m im Register x_{k+1} hält.

Jede partielle Funktion, die auf einer RM berechenbar ist, ist auch auf einer TM berechenbar und umgekehrt.

Programmverifikation

Grundsätzlich wollen wir fehlerfreien Code schreiben. Dies ist in der Praxis nicht möglich, daher müssen wir unser Programm spezifizieren, diese Spezifikation verifizieren (Beweisen) und wenn dann unsere Implementierung nicht von der Spezifikation abweicht, können wir fehlerfreien Code garantieren.

Eine (sehr schlechte, unvollständige) Methode ist, einfach Testen. Hunderte, Tausende mal. Wir können aber nie garantieren, dass es beim Tausend und ersten Mal nicht doch einen Fehler gibt.

Formale Methoden der Verifikation dienen zum Zeigen der Korrektheit eines Programms. Diese Verifikation muss von der Validierung abgegrenzt werden, weil letztere nur überprüft, ob die Spezifikation unseren Anforderungen entspricht.

Verifikation überprüft, ob wir die Dinge richtig machen,
Validierung checkt, ob wir das Richtige tun.

Verifikation nach Hoare

Das Hoare-Kalkül spielt in der Praxis eine untergeordnete Rolle. Es ist eine nicht vollständig automatisierbare Verifikationsmethode und stößt schnell an ihre Grenzen. Dennoch ist es eine hervorragende Methode zum Einführen in die Programmverifikation.

Weil wir mit der Aussagenlogik nicht genügend Möglichkeiten haben, die interessanten Eigenschaften eines Programmes auszudrücken, erweitern wir unseren Formalismus um die **prädikatenlogischen Ausdrücke**. Sie sind die Grundlage der Verifikation nach Hoare.

Prädikatensymbole

Wir beschränken uns bei diesen prädikatenlogischen Ausdrücken aber auf atomare Formeln und deren Abschluss unter aussagenlogischen Junktoren (siehe Seite 5). Dabei ist die wichtigste Erweiterung die sogenannten **Prädikatensymbole**.

Damit erlauben uns Prädikatensymbole über Elemente einer Menge, Aussagen zu treffen. Um diese Elemente in einer Menge zu referenzieren, verwendet man Terme.

Terme sind Ausdrücke mit Variablen

Beispiele

Angenommen wir haben eine Konstante **7** und das Prädikatensymbol **ist_prim** in unserer Sprache, dann können wir **ist_prim(7)** schreiben, um auszudrücken, dass 7 eine Primzahl ist.

Wäre bei s = t das s und das t Terme. Das = können wir als Prädikat auffassen, weil es eine Beziehung ausdrückt.

Oder aber folgender „semantischer“ Ausdruck:

(Sokrates $\in$ Mensch) $\wedge$ Für alle x $\in$ Mensch | x ist sterblich.
Damit ist auch Sokrates sterblich.

Lässt sich in eine prädikatenlogische Aussage umformen:

$$\underbrace{M(\text{Sokrates}) \wedge (\forall\ x \mid M(x) \rightarrow S(x))}_{\text{Prädikat}} \rightarrow S(\text{Sokrates})$$

Wobei M und S die eigentlichen Prädikate sind, die eine Eigenschaft über das enthaltene Element ausdrücken (S = sterblich, M = Mensch)

Zusicherungen

Nehmen wir zusätzlich noch atomare Formeln ins Boot, können wir Zusicherungen induktiv definieren:

1. Atome sind Zusicherungen
2. Wenn A und B Zusicherungen sind, dann sind $\neg$A, (A $\wedge$ B), (A $\vee$ B) und (A$\rightarrow$B) auch Zusicherungen

Zusicherungen erlauben uns, den Zustand eines Programmes zu beschreiben. Sie sind rein syntaktisch definiert, das heißt wir müssen den Formeln eine Bedeutung zuordnen. Dazu verwenden wir die Interpretationen, welche wir als Verallgemeinerung von Algebren verstehen können.

Eine Interpretation **I** gibt an, wie Symbole einer Zusicherung zu verstehen sind. Zum Beispiel wollten wir oben in den Beispielen das **Symbol ist_prim** so interpretieren, dass das **Atom ist_prim(n)** wahr wird, gdw. n eine Primzahl ist.

Das bedeutet also, in der Interpretation I ist definiert, wann Atome zu wahr evaluiert werden. Zusicherungen sind dann im Grunde nur noch aussagenlogische Verknüpfungen dieser Interpretationsresultate.

Konsequenzrelation

Eine Zusicherung B kann auch aus einer Prämisse A folgen. Dazu definieren wir

$$A \vDash B$$

$x < 5 \vDash x+1 < 7$

Hoare-Tripel

Sei P ein while-Program, dann ist das Hoare-Tripel definiert als:

$$\{Q\}\ P\ \{R\}$$

Wobei Q und R Zusicherungen sind. Q wird auch als Vorbedingung und R als Nachbedingung bezeichnet.

Beispiel:

$$\{x = 0\}\ x := x-1\ \{x = 0\}$$

Das ist eine Registermaschine!

Regeln nach Tony Hoare

Die Regeln werden von unten nach oben gelesen, das Problem, die Korrektheit des Programms zu zeigen, wird sukzessive in kleinere Teile aufgespaltet.

Zuweisungsaxiom [z]

$$\overline{\{Q\{x \mapsto t\}\}\ x := t\ \{Q\}}$$

Es besagt, dass nach einer Zuweisung jede Aussage (welche vorher für die rechte Seite der Zuweisung galt) nun auch für die Variable gilt.

$$\{x + 1 = 43\}\ y := x + 1\ \{y = 43\}$$

Diese Regel ist ein Axiom des Kalküls, weil keine Vorbedingungen erfüllt sein müssen. Es wird verwendet, um Zuweisungsbefehle korrekt in den Zusicherungen abbilden zu können.

Sequenzregel [s]

$$\frac{\{Q\}\ P_1\ \{R\} \quad \{R\}\ P_2\ \{S\}}{\{Q\}\ P_1; P_2\ \{S\}}$$

Diese Regel dient zum „beweisen“ der Aussage unter dem Strich. Wenn die Aussage über dem Strich bewiesen wurde, kann auch die Aussage drunter als bewiesen angesehen werden.

$$\{x + 1 = 43\}\ y := x + 1\ \{y = 43\}$$

und

$$\{y = 43\}\ z := y\ \{z = 43\}$$

kann man die folgende Aussage daraus folgern:

$$\{x + 1 = 43\}\ y := x + 1; z := y\ \{z = 43\}$$

Damit beweisen wir die Korrektheit der einzelnen Teile eines Programms.

Konsequenz Regel [a]

$$\frac{\{Q'\}\ P\ \{R'\}}{\{Q\}\ P\ \{R\}}\ Q \models Q', R' \models R$$

Sie erlaubt es, die Vorbedingung abzuschwächen (Q |= Q‘) und die Nachbedingung zu verstärken (R‘ |= R) und so die Anwendung anderer Beweisregeln zu ermöglichen. Von unten nach oben gelesen, wird also die Aussage abgeschwächt, deshalb die Bezeichnung a.

Iterations-Regel (While-Regel) [w]

$$\frac{\{I \wedge B\}\ P\ \{I\}}{\{I\}\ \text{while}\ B\ \text{do}\ P\ \text{end}\ \{I \wedge \neg B\}}$$

Hierbei wird I als die Schleifeninvariante bezeichnet, welche während als auch nach der Ausführung der Schleife gültig ist.

Sie überprüft generell die Korrektheit von while-Schleifen. Wir setzen die Terminierung von B vorraus.

Hoare-Kalkül

Seien Q, R Zusicherungen und P ein while-Programm, dann heißt P korrekt in Bezug auf Q und R wenn {Q} P {R} mithilfe der obigen Regeln abgeleitet werden kann.

Wir nennen P partiell korrekt für eine Spezifikation S, wenn P korrekt ist in Bezug auf die Zusicherung Q und R die der Spezifikation S entsprechen.

Angenommen P ist ein partiell korrektes Programm für eine gegebene Spezifikation. Wenn es noch gelingt die Terminierung von P nachzuweisen, dann ist auch der Beweis der *totalen Korrektheit* gelungen.

Gutserlen

Verschiebechiffre (Cäsar-Verschlüsselung)

SQTQUY → GEHEIM

#(K) = Anzahl möglicher Schlüssel, also die Anzahl an möglichen Verschiebungen bis wir das Ergebnis erhalten.
= 25

Bei einem 26 Zeichen langen Alphabet entspricht das **#(K) = 25**. (weil ja 1er der 26 Schlüssel schon vorliegt und dieser nicht mehr ein möglicher Schlüssel ist).

***u* =** Formel zum Berechnen der Anzahl an Zeichen aus der ein Chiffretext bestehen muss, damit die erwartete Anzahl sinnvoller Schlüssel 1 ist.
= 1,35337

u = $(\log_2(\#(K))) / (R^* \log_2(\#(A))) = (\log_2(25)) / (0{,}73 * \log_2(26))$ **= 1,35337**

Was heißt das?

Das bedeutet, nach spätestens 25 Verschiebungen hat ein Angreifer eine sinnvolle Nachricht. Allerdings gibt es oft mehrere sinnvolle Nachrichten mit verschiedenen Schlüsseln. Nur wenn die Geheimnachricht aus 1,35 Zeichen besteht, kann der

Angreifer mit nur einem sinnvollen Schlüssel rechnen. Damit ist aber die Nachricht relativ sinnbefreit.

Substitutionschiffre

FIMHMIWIWIQWXVMRKWXMQQIRHMILEIYJMKOIMXIRFIMRELI

#(K) = 26! = 4,03... * 10^{26}

Weil bei 26 Möglichkeiten, der erste Buchstabe auf 26 Buchstaben substituioniert werden kann, beim zweiten Buchstaben aber nur mehr 25, dann 24, 23, ... = 26!

u = ($\log_2$(#(K))) / (R* $\log_2$(#(A))) = ($\log_2$(26!)) / (0,73 * $\log_2$(26)) **= 25,7574**

Das heißt, besteht der Chiffretext aus ~26 Zeichen, gibt es einen sinnvollen Schlüssel. (unser Text besteht aus 47)

Damit wäre der Angreifer sehr lange beschäftigt, alles händisch durchzutesten. Er kann aber die Häufigkeitsanalyse durchführen, womit bereits einige Buchstaben eindeutig zugewiesen werden können.