



Einführung in die Theoretische Informatik

Martin Avanzini Christian Dalvit Jamie Hochrainer

Georg Moser Johannes Niederhauser Jonas Schöpf

<https://tcs-informatik.uibk.ac.at>



Organisation

Zeitplan

Woche 1	8. Oktober	Woche 8	3. Dezember
Woche 2	22. Oktober	Woche 9	10. Dezember
Woche 3	29. Oktober	Woche 10	17. Dezember
Woche 4	5. November	Woche 11	14. Jänner
Woche 5	12. November	Woche 12	21. Jänner
Woche 6	19. November	Woche 13	28. Jänner
Woche 7	26. November	SL Klausur	4. Feber
		1te Klausur	11. Februar
		2te Klausur	11. März

Zeit und Ort

Vorlesung Freitag, 10:15–12:00, HS A Georg Moser

Tutorium Donnerstag, 12:15–13:00, HS A Christian Dalvit

- 1 Skriptum
bei Studia (10te überarbeitete Auflage)



Online-Lehrmittel

- 2 **Skriptum** ist bei Studia verfügbar und wird ab übernächster Woche innerhalb des Universitätsnetzes verfügbar sein
- 3 Version mit Lösung zum Semesterende
- 4 **Folien, Hausaufgaben, Selbsttests** sind auf OLAT abrufbar
- 5 Folien sind (üblicherweise) **vor** der Vorlesung online
- 6 **Ausgewählte** Lösungen werden verfügbar gemacht, **nachdem** sie in den SL-Gruppen besprochen wurden

Zeit und Ort der Studienorientierungslehrveranstaltungen (SL)

Gruppe 1	Freitag, 13:15–14:00, HS C	Georg Moser
Gruppe 2	Freitag, 12:15–13:00, HSB 7	Jonas Schöpf
Gruppe 3	Freitag, 13:15–14:00, HSB 7	Jonas Schöpf
Gruppe 4	Freitag, 14:15–15:00, eLecture	Martin Avanzini
Gruppe 5	Freitag, 13:15–14:00, eLecture	Martin Avanzini
Gruppe 6	Freitag, 13:15–14:00, HSB 6	Johannes Niederhauser
Gruppe 7	Freitag, 14:15–15:00, HSB 6	Johannes Niederhauser
Gruppe 8	Freitag, 13:15–14:00, HS E	Jamie Hochrainer
Gruppe 9	Freitag, 12:15–13:00, HS E	Jamie Hochrainer

Termine

- 1 Die SL beginnt am **22. Oktober** und endet am 28. Jänner; keine Ausnahmen für Nachmeldungen
- 2 **SL Klausur am 4. Februar**

Prüfungsmodus in Vorlesung & SL

SL

Aus formalen Gründen, besteht in der SL keine Anwesenheitspflicht. Wir empfehlen aber **dringend** die SL regelmäßig zu besuchen!

Klausuren

- 1** Die erste und zweite Vorlesungsprüfung findet im Februar, bzw. März statt; eine Anmeldung (per Ifu:online) ist zwingend erforderlich.
- 2** Sofern möglich ist die Vorlesungsprüfung in **Präsenz**.
- 3** Die SL Klausur findet (in Präsenz) in der entsprechenden SL Gruppe statt; eine Anmeldung ist nicht erforderlich.
- 4** Die Klausurvorbereitungen (für VO+SL) finden im Tutorium/Vorlesung statt
- 5** Zur Abgrenzung von der VO Klausur wird sich die SL Klausur auf die Bereiche der LVA beschränken, die nicht in der Vorlesungsklausur behandelt werden.

Organisatorisches

ARSNova Session

1 QR Code

2 <https://arsnova.uibk.ac.at/mobile/#id/77974190>



Sämtliche weitere organisatorische Informationen, siehe OLAT



Theoretische Informatik

Begriffsdefinition

Die Theoretische Informatik beschäftigt sich mit der Abstraktion, Modellbildung und grundlegenden Fragestellungen, die mit der Struktur, Verarbeitung, Übertragung und Wiedergabe von Informationen in Zusammenhang stehen.

Ihre Inhalte sind Automatentheorie, Theorie der formalen Sprachen, Berechenbarkeits- und Komplexitätstheorie, aber auch Logik und formale Semantik sowie die Informations-, Algorithmen- und Datenbanktheorie.

<http://de.wikipedia.org/> 2021

- 1** Automatentheorie
- 2** Theorie der formalen Sprachen
- 3** Berechenbarkeits- und Komplexitätstheorie
- 4** Logik und formale Semantik
- 5** Informations-, Algorithmen- und Datenbanktheorie

Handbook of Theoretical Computer Science



+



= 2293 Seiten, 4 Kilogramm

Inhaltsverzeichnis Band „Algorithms and Complexity“

Machine models and simulations, A catalog of complexity classes, Machine-independent complexity theory, Kolmogorov complexity and its applications, Algorithms for finding patterns in strings, Data structures, Computational geometry, Algorithmic motion planning in robotics, Average-case analysis of algorithms and data structures, Graph algorithms, Cryptography, Algebraic complexity theory, Algorithms in number theory, The complexity of finite functions, Communication networks, VLSI theory, Parallel algorithms for shared-memory machines, General purpose parallel architectures

Inhaltsverzeichnis Band „Formal Models and Semantics“

Finite automata, Context-free languages, Formal languages and power series, Automata on infinite objects, Graph rewriting: an algebraic and logic approach, Rewrite systems, Functional programming and lambda calculus, Type systems for programming languages, Recursive applicative program schemes, Logic programming, Denotational semantics, Semantic domains, Algebraic specification, Logics of programs, Methods and logics for proving programs, Temporal and modal logic, Elements of relational database theory, Distributed computing: models and methods, Operational and algebraic semantics of concurrent processes



Turing Maschinen,
Berechenbarkeitstheorie,
Alan Turing

—

1930er

Formale Sprachen,
Automatentheorie

Zuse Z3, ENIAC

1940er



Grammatiken, Grundlagen
des Compilerbaus,
Noam Chomsky

UNIVAC, Transistoren
statt Röhren

1950er



P vs. NP
Komplexitätstheorie,
Stephen Cook

Minicomputer,
integrierte
Schaltkreise

1960er

Inhalte der Lehrveranstaltung

Einführung in die Logik

Syntax & Semantik der Aussagenlogik, Kalkül des natürlichen Schließens, Konjunktive und Disjunktive Normalformen

Einführung in die Algebra

algebraische Strukturen, Boolesche Algebra

Einführung in die Theorie der Formalen Sprachen

Grammatiken und Formale Sprachen, Reguläre Sprachen, Kontextfreie Sprachen, Chomsky-Hierarchie, Anwendungen von formalen Sprachen

Einführung in die Berechenbarkeitstheorie und Komplexitätstheorie

Algorithmisch unlösbare Probleme, Turing Maschinen, Registermaschinen, Komplexitätstheorie

Einführung in die Programmverifikation

Prinzipien der Analyse von Programmen, Verifikation nach Hoare

Bachelor Informatik: die ersten zwei Jahre

1. Semester	Einführung in die Programmierung	Einf. Theoretische Informatik	Rechnerarchitektur
	Funktionale Programmierung	Lineare Algebra	
2. Semester	Programmiermethodik	Algorithmen und Datenstrukturen	Angewandte Mathematik
			Betriebssysteme
3. Semester	Softwarearchitektur	Datenbanksysteme	Diskrete Strukturen
		Daten und Wahrscheinlichkeiten	Rechnernetze und Internettechnik
4. Semester	Software Engineering	Maschinelles Lernen	Logik
	Parallele Programmierung		Einführung in das wissensch. Arbeiten



Einführung in die Logik

Logisches Schließen im Allgemeinen

Beispiel

- We arrive at the following paradox in a globalised world: when nationalists pursue more formal sovereignty they achieve less real sovereignty of the people. They want to take back control and they end up with less control.
- That's what the UK will end up with. [...]
- Yet this paradox also has a corollary: when countries in Europe renounce formal sovereignty this leads to more real sovereignty for the people of Europe.

Paul De Grauwe, 2017¹

¹Siehe <https://blogs.lse.ac.uk/brexit/2017/10/06/the-catalan-crisis-and-brexit-stem-from-the-same-kind-of-nationalism/>.

Results of Brexit



Beispiel

Yet this paradox also has a **corollary**: when countries in Europe renounce formal sovereignty this leads to more real sovereignty for the people of Europe.

Bemerkung

- In der Informatik und im Allgemeinen in den Natur- und Ingenieurwissenschaften muss ein „Korollar“ logisch folgen.
- Dazu haben wir in der Logik eine eigene **formale Sprache** für Folgerungen und allgemeiner **gültige** Schlüsse eingeführt.

Beispiel (Fortsetzung)

In dieser Sprache, stellt sich das sogenannte „Korollar“, wie folgt dar (und ist logisch falsch):

$$\begin{aligned} \text{„more formal sovereignty“} &\rightarrow \text{„less real sovereignty“} \models \\ \models \neg \text{„more formal sovereignty“} &\rightarrow \neg \text{„less real sovereignty“} \end{aligned}$$

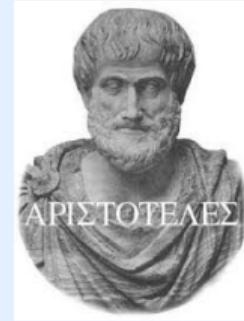


Grundlagen der Logik

Aristoteles sagte

Topik I 1, 100a25-27

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



Beispiel

Sokrates ist ein Mensch	}	Prämisse ①
Alle Menschen sind sterblich	}	Prämisse ②
Somit ist Sokrates sterblich	}	Konklusion

Definition

- Schlussfiguren dieser Art heißen **Syllogismen**
- Syllogismen wurden bereits im antiken Griechenland untersucht, Grundlage der modernen Logik

Fakt

*Nicht die Wahrheit der Prämissen, oder der Konklusion, sondern die Wahrheit der **Schlussfigur** ist entscheidend*

Beispiele für Arten von Syllogismen

Alle Griechen sind Menschen

AAA - modus barbara

Alle Menschen sind sterblich

Somit sind alle Griechen sterblich

Alle Professoren sind ernst

AOO - modus baroco

Einige Dozenten sind nicht ernst

Somit sind einige Dozenten keine Professoren

Typisierung der Relationen

A Alle S sind P E Keine S sind P

I Einige S sind P O Einige S sind nicht P

Modus Ponens

Beispiel

Wenn das Kind schreit, hat es Hunger

Das Kind schreit

Also, hat das Kind Hunger

Fakt

Korrektheit dieser Schlussfigur ist unabhängig von den konkreten Aussagen

Definition (*Modus Ponens*)

Wenn A , dann B

A gilt

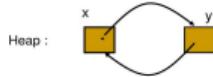
Also, gilt B

Logik in der Informatik



Separation Logic

Formula : $x \rightarrowtail y = y \rightarrowtail x$



A screenshot of a Microsoft Excel spreadsheet. The menu bar at the top includes 'FORMULAS', 'DATA', 'REVIEW', 'VIEW', 'Connections', 'Properties', 'Set', 'Filter', 'Advanced', 'Text to Columns', 'Flash Fill', 'Remove Duplicates', 'Validation', 'Data Tools', and 'Analysis'. The 'Flash Fill' button is highlighted. The main area shows a table with columns 'C', 'D', and 'E'. Cell C1 contains the formula '=B2\$155-1212'. Cell E1 contains the text 'Flash Fill Examples'. Below the table, there is a 'Flash Fill Examples' section with a grid showing how the formula will be applied across multiple rows. The data in the table is as follows:

Die Bedeutung der Logik in der Informatik ist um einiges größer als die Bedeutung der Logik in der Mathematik (oder Philosophie, Linguistik, ...)



Einführung in die Theoretische Informatik

Martin Avanzini Christian Dalvit Jamie Hochrainer

Georg Moser Johannes Niederhauser Jonas Schöpf

<https://tcs-informatik.uibk.ac.at>



Zusammenfassung

Zusammenfassung der letzten LVA

Beispiel

Wenn das Kind schreit, hat es Hunger

Das Kind schreit

Also, hat das Kind Hunger

Fakt

Korrektheit dieser Schlussfigur ist unabhängig von den konkreten Aussagen

Definition (*Modus Ponens*)

Wenn A , dann B

A gilt

Also, gilt B

Einführung in die Logik

Syntax & Semantik der Aussagenlogik, Kalkül des natürlichen Schließens, Konjunktive und Disjunktive Normalformen

Einführung in die Algebra

algebraische Strukturen, Boolesche Algebra

Einführung in die Theorie der Formalen Sprachen

Grammatiken und Formale Sprachen, Reguläre Sprachen, Kontextfreie Sprachen, Chomsky-Hierarchie, Anwendungen von formalen Sprachen

Einführung in die Berechenbarkeitstheorie und Komplexitätstheorie

Algorithmisch unlösbare Probleme, Turing Maschinen, Registermaschinen, Komplexitätstheorie

Einführung in die Programmverifikation

Prinzipien der Analyse von Programmen, Verifikation nach Hoare

Syntax der Aussagenlogik

Definition

Sei **AT** eine Menge von **atomaren Formeln** (oder **Atomen**), deren Elemente mit p, q, r, \dots bezeichnet werden

Definition

Wahrheitswertsymbole:

True False

Junktoren:

¬ ∧ ∨ →

Aussagenlogische Formeln

Definition

Die **Formeln** der Aussagenlogik sind induktiv definiert:

- 1 Eine atomare Formel p ist eine **Formel**,
- 2 ein Wahrheitswertsymbol (True, False) ist eine **Formel**, und
- 3 wenn A und B **Formeln** sind, dann sind auch die Folgenden, **Formeln**:

$$\neg A \quad (A \wedge B) \quad (A \vee B) \quad (A \rightarrow B)$$

Beispiel

Der folgende Ausdruck A ist eine Formel

$$((p \rightarrow \neg q) \rightarrow (\neg q \rightarrow \neg p))$$

Präzedenzen

Konvention

Wir verwenden die folgende Präzedenz:

$\neg > \vee, \wedge > \rightarrow$ \rightarrow ist rechts-assoziativ: $p \rightarrow (q \rightarrow r)$

Beispiel

$\neg p \wedge q \rightarrow r \vee s$ statt $((\neg p \wedge q) \rightarrow (r \vee s))$

Wahrheitswertbelegung

Definition

- 1 T und F bezeichnen die beiden betrachteten **Wahrheitswerte**
- 2 **Belegung v**: AT $\rightarrow \{T, F\}$ assoziiert Atome mit Wahrheitswerten

Beispiel

Betrachte die Atome p, q und r, sowie die folgende Belegung:

$$v(a) := \begin{cases} T & a = p \\ F & a = q \\ F & a = r \end{cases}$$

Wir schreiben auch $v(p) = T$, $v(q) = F$, $v(r) = F$

- 1 Atome sind Platzhalter für konkrete Aussagen
- 2 Junktoren sind formale Zeichen, die Aussagen verbinden

 Negation  Konjunktion  Disjunktion  Implikation

- 3 Die Bedeutung wird durch Wahrheitstafeln definiert

\neg	T	F	\wedge	T	F	\vee	T	F	\rightarrow	T	F
T	F	T	T	F	T	T	T	T	T	T	F
F	T	F	F	F	F	F	T	F	F	T	T

Beispiel

Der allgemeine Aussage „Wenn p , dann q “ kann nun konzise ausgedrückt werden:

$$p \rightarrow q$$

Semantik der Aussagenlogik

Definition

Erweiterung der Belegung v zu einem **Wahrheitswert** für Formeln:

$$\bar{v}(p) = v(p) \quad \bar{v}(\text{True}) = T \quad \bar{v}(\text{False}) = F$$

$$\bar{v}(\neg A) = \begin{cases} T & \bar{v}(A) = F \\ F & \bar{v}(A) = T \end{cases}$$

$$\bar{v}(A \wedge B) = \begin{cases} T & \bar{v}(A) = \bar{v}(B) = T \\ F & \text{sonst} \end{cases}$$

$$\bar{v}(A \vee B) = \begin{cases} F & \bar{v}(A) = \bar{v}(B) = F \\ T & \text{sonst} \end{cases}$$

$$\bar{v}(A \rightarrow B) = \begin{cases} T & \bar{v}(A) = F \text{ oder } \bar{v}(B) = T \\ F & \text{sonst} \end{cases}$$

Wahrheitstabelle

Beispiel

Sei $v(p) = T, v(q) = F$, dann $\bar{v}(A) = \bar{v}((p \rightarrow \neg q) \rightarrow (\neg q \rightarrow \neg p)) = F$

Definition

Sei A eine Formel; die **Wahrheitstabelle von A** listet alle relevanten Belegungen v zusammen mit dem Wahrheitswert $\bar{v}(A)$ auf

Beispiel

Betrachte die Formel:

$$(p \rightarrow \neg q) \rightarrow (\neg q \rightarrow \neg p)$$

Wir stellen die folgende Wahrheitstabelle auf:

p	q	$(p \rightarrow \neg q)$	$(\neg q \rightarrow \neg p)$	$(p \rightarrow \neg q) \rightarrow (\neg q \rightarrow \neg p)$
T	T	F	T	T
T	F	T	F	F
F	T	T	T	T
F	F	T	T	T

Definition

sei A eine Formel

- 1 wenn Belegung v existiert, sodass $\bar{v}(A) = T$, heißt A erfüllbar
- 2 wenn keine solche Belegung existiert, heißt A unerfüllbar
- 3 wenn für alle Belegungen v , $\bar{v}(A) = T$, heißt A gültig oder Tautologie

Definition

Die Konsequenzrelation $\{A_1, \dots, A_n\} \models B$ gilt, gdw. für alle Belegungen v :

$$\bar{v}(A_1) = T, \dots, \bar{v}(A_n) = T \text{ impliziert } \bar{v}(B) = T$$

- Wir schreiben $\models A$ statt $\emptyset \models A$; außerdem schreiben wir $A_1, \dots, A_n \models B$ statt $\{A_1, \dots, A_n\} \models B$
- Gilt $\emptyset \models A$ dann ist A eine Tautologie

Satz

Eine Formel A ist eine Tautologie gdw. $\neg A$ unerfüllbar

Beweis.

1 Wir zeigen die Richtung von links nach rechts:

- angenommen $\bar{v}(A) = T$, für alle Belegungen v
- also $\bar{v}(\neg A) = F$, für alle Belegungen v
- somit ist $\neg A$ unerfüllbar

2 Wir zeigen die Richtung von rechts nach links:

- angenommen $\neg A$ ist unerfüllbar
- $\bar{v}(\neg A) = F$, für alle Belegungen v
- also $\bar{v}(A) = T$, für alle Belegungen v und somit gültig

Definition (Äquivalenz)

$A \equiv B$, wenn $A \models B$ und $B \models A$ gilt

Satz

$A \equiv B$ gilt gdw. $(A \rightarrow B) \wedge (B \rightarrow A)$ eine Tautologie

Beweis.

Wir zeigen die Richtung von links nach rechts:

- $(A \rightarrow B) \wedge (B \rightarrow A)$ gültig gdw. $(A \rightarrow B)$ gültig und $(B \rightarrow A)$ gültig
- Angenommen $A \models B$; dann gilt für alle Belegungen v:

$$\bar{v}(A) = T \text{ impliziert } \bar{v}(B) = T$$

- $\bar{v}(A \rightarrow B) = T$ für alle v
- $(A \rightarrow B)$ ist gültig
- ähnlich folgt aus $B \models A$, dass $(B \rightarrow A)$ gültig



Assoziativität und Kommutativität von Junktoren

- Konjunktion und Disjunktion sind assoziativ und kommutativ
- Wir unterscheiden nicht zwischen:

$$(A \wedge B) \wedge C$$

$$A \wedge B$$

$$A \wedge (B \wedge C)$$

$$B \wedge A$$

$$A \wedge B \wedge C$$

Definition

1 $\bigwedge_{i=1}^n A_i = A_1 \wedge \cdots \wedge A_n \quad n \geq 1$

2 $\bigwedge_{i=1}^0 A_i = \text{True}$

3 $\bigvee_{i=1}^n A_i = A_1 \vee \cdots \vee A_n \quad n \geq 1$

4 $\bigvee_{i=1}^0 A_i = \text{False}$

Äquivalenzen I

Lemma (Elementare Äquivalenzen)

$$\begin{array}{llll} \neg\neg A \equiv A & A \vee \text{True} \equiv \text{True} & A \wedge \text{True} \equiv A & A \rightarrow \text{True} \equiv \text{True} \\ A \vee \text{False} \equiv A & A \wedge \text{False} \equiv \text{False} & A \rightarrow \text{False} \equiv \neg A & \\ A \vee A \equiv A & A \wedge A \equiv A & \text{True} \rightarrow A \equiv A & \\ A \vee \neg A \equiv \text{True} & A \wedge \neg A \equiv \text{False} & \text{False} \rightarrow A \equiv \text{True} & \\ & & A \rightarrow A \equiv \text{True} & \end{array}$$

Lemma (Distributivgesetze und Andere)

$$\begin{array}{ll} A \rightarrow B \equiv \neg A \vee B & \neg(A \rightarrow B) \equiv A \wedge \neg B \\ A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C) & A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C) \end{array}$$

Äquivalenzen II

Lemma (Absorptionsgesetze)

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

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

Lemma (Gesetze von de Morgan)

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

Gleiches durch Gleiches Ersetzen

Definition

Eine **Teilformel** A einer Formel B ist ein Teilausdruck von B , der wiederum eine Formel ist

Satz

- 1** A, B Formeln und E, F Teilformeln von A, B
- 2** Gelte $E \equiv F$
- 3** B ist das Resultat der Ersetzung von E durch F in A

Dann gilt $A \equiv B$

Beispiel

Wir betrachten die folgende Äquivalenz

$$p \rightarrow q \equiv \neg p \vee q$$

mit der folgenden Formel

$$(p \rightarrow q) \wedge r$$

Nun gilt

$$\underline{(p \rightarrow q) \wedge r} \equiv \underline{(\neg p \vee q)} \wedge r$$



Einführung in die Theoretische Informatik

Martin Avanzini Christian Dalvit Jamie Hochrainer

Georg Moser Johannes Niederhauser Jonas Schöpf

<https://tcs-informatik.uibk.ac.at>



Zusammenfassung

Zusammenfassung der letzten LVA

Definition

Die **Formeln** der Aussagenlogik sind induktiv definiert:

- 1 Eine atomare Formel p ist eine **Formel**,
- 2 ein Wahrheitswertsymbol (True, False) ist eine **Formel**, und
- 3 wenn A und B **Formeln** sind, dann sind

$$\neg A \quad (A \wedge B) \quad (A \vee B) \quad (A \rightarrow B)$$

auch **Formeln**

Definitionen

- Erweiterung der Belegung v zu einem **Wahrheitswert** \bar{v} für Formeln
- $A \equiv B$, wenn $A \models B$ und $B \models A$ gilt

Einführung in die Logik

Syntax & Semantik der Aussagenlogik, Kalkül des natürlichen Schließens, Konjunktive und Disjunktive Normalformen

Einführung in die Algebra

algebraische Strukturen, Boolesche Algebra

Einführung in die Theorie der Formalen Sprachen

Grammatiken und Formale Sprachen, Reguläre Sprachen, Kontextfreie Sprachen, Chomsky-Hierarchie, Anwendungen von formalen Sprachen

Einführung in die Berechenbarkeitstheorie und Komplexitätstheorie

Algorithmisch unlösbare Probleme, Turing Maschinen, Registermaschinen, Komplexitätstheorie

Einführung in die Programmverifikation

Prinzipien der Analyse von Programmen, Verifikation nach Hoare



Methode von Quine

Methode von Quine

Lemma

A eine Formel und p ein Atom in A

1 A ist eine Tautologie gdw.

$A\{p \mapsto \text{True}\}$ ist Tautologie und $A\{p \mapsto \text{False}\}$ ist Tautologie

2 A ist unerfüllbar gdw.

$A\{p \mapsto \text{True}\}$ unerfüllbar und $A\{p \mapsto \text{False}\}$ unerfüllbar

Beispiel (Wahrheitstabellen oder logische Äquivalenzen)

Wir betrachten die Formel F

$$F := [(p \wedge q \rightarrow r) \wedge (p \rightarrow q)] \rightarrow (p \rightarrow r)$$

Es ist nicht schwer einzusehen, dass F eine Tautologie ist

Beispiel (Methode von Quine)

Die Methode liefert die folgenden Anforderungen

- 1 $(\text{True} \wedge q \rightarrow r) \wedge (\text{True} \rightarrow q) \rightarrow (\text{True} \rightarrow r) =: G$ ist Tautologie
- 2 $(\text{False} \wedge q \rightarrow r) \wedge (\text{False} \rightarrow q) \rightarrow (\text{False} \rightarrow r)$ ist Tautologie

Anforderungen in Baumform:

$$\begin{array}{c} ((p \wedge q \rightarrow r) \wedge (p \rightarrow q)) \rightarrow (p \rightarrow r) \\ \searrow \{p \mapsto \text{False}\} \\ (\text{False} \wedge q \rightarrow r) \wedge (\text{False} \rightarrow q) \rightarrow (\text{False} \rightarrow r) \\ \equiv \\ (\text{False} \rightarrow r) \wedge (\text{False} \rightarrow q) \rightarrow \text{True} \\ \equiv \\ \text{True} \wedge \text{True} \rightarrow \text{True} \\ \equiv \\ \text{True} \end{array}$$

$\{p \mapsto \text{True}\}$

G

Übrige Anforderungen

- 3 G ist Tautologie

Beispiel (Fortsetzung)

$$\overbrace{(\text{True} \wedge q \rightarrow r) \wedge (\text{True} \rightarrow q) \rightarrow (\text{True} \rightarrow r)}^{\text{G}}$$
$$\equiv$$
$$(q \rightarrow r) \wedge q \rightarrow r$$
$$\begin{array}{c} \{q \mapsto \text{True}\} \quad \{q \mapsto \text{False}\} \\ (\text{True} \rightarrow r) \wedge \text{True} \rightarrow r \quad (\text{False} \rightarrow r) \wedge \text{False} \rightarrow r \\ \equiv \quad \equiv \\ r \wedge \text{True} \rightarrow r \quad \text{True} \wedge \text{False} \rightarrow r \\ \equiv \quad \equiv \\ r \rightarrow r \quad \text{False} \rightarrow r \\ \equiv \quad \equiv \\ \text{True} \quad \text{True} \end{array}$$

Es gibt keine weiteren Anforderungen mehr, also ist F eine Tautologie



Natürliches Schließen

Inferenzregeln: Konjunktion

Definition

	<i>Einführung</i>	<i>Elimination</i>
\wedge	$\frac{A \quad B}{A \wedge B} \wedge : i$	$\frac{A \wedge B}{A} \wedge : e \quad \frac{A \wedge B}{B} \wedge : e$

Beispiel

Wir betrachten die folgenden Inferenzen, die wir etwa in informellen Beweisen verwendet haben:

$$\frac{p \rightarrow q \quad q \rightarrow p}{(p \rightarrow q) \wedge (q \rightarrow p)} \wedge : i \quad \frac{(p \rightarrow q) \wedge (q \rightarrow p)}{p \rightarrow q} \wedge : e \quad \frac{(p \rightarrow q) \wedge (q \rightarrow p)}{q \rightarrow p} \wedge : e$$

Inferenzregeln: Disjunktion

Definition

	<i>Einführung</i>	<i>Elimination</i>
\vee	$\frac{A}{A \vee B} \text{ v: i} \quad \frac{B}{A \vee B} \text{ v: i}$	$\frac{A \vee B}{C} \quad \frac{\begin{array}{ c c } \hline A & B \\ \vdots & \vdots \\ C & C \\ \hline \end{array}}{C} \text{ v: e}$

Beispiel

$$\frac{p}{p \vee \neg p} \text{ v: i} \quad \frac{\neg p}{p \vee \neg p} \text{ v: i} \quad \frac{\text{True} \vee \text{False}}{\text{True}} \quad \frac{\begin{array}{|c|c|} \hline \text{True} & \text{False} \\ \text{True} & \text{True} \\ \hline \end{array}}{\text{True}} \text{ v: e}$$

Inferenzregeln: Implikation

Definition

	<i>Einführung</i>	<i>Elimination</i>
→	$\frac{A \quad \vdots \quad B}{A \rightarrow B}$ →: i	$\frac{A \quad A \rightarrow B}{B}$ →: e

Beispiel

$$\frac{\begin{array}{c} \text{False} \\ \text{True} \end{array}}{\text{False} \rightarrow \text{True}} \rightarrow: i$$

$$\frac{p \quad p \rightarrow q}{q} \rightarrow: e$$

Inferenzregeln: Negation et al.

Definition

	<i>Einführung</i>	<i>Elimination</i>
\neg	$\frac{A \quad \vdots \quad \text{False}}{\neg A} \neg : i$	$\frac{A \quad \neg A}{\text{False}} \neg : e$
False		$\frac{\text{False}}{A} \text{ False} : e$
$\neg\neg$		$\frac{\neg\neg A}{A} \neg\neg : e$

Natürliches Schließen (alle Inferenzregeln)

	<i>Einführung</i>	<i>Elimination</i>
\wedge	$\frac{A \quad B}{A \wedge B} \wedge: i$	$\frac{A \wedge B}{A} \wedge: e \quad \frac{A \wedge B}{B} \wedge: e$
\vee	$\frac{A}{A \vee B} \vee: i \quad \frac{B}{A \vee B} \vee: i$	$\frac{A \vee B}{\begin{array}{ c c } \hline A & B \\ \vdots & \vdots \\ C & C \\ \hline \end{array}} \vee: e$
\rightarrow	$\frac{\begin{array}{ c } \hline A \\ \vdots \\ B \\ \hline \end{array}}{A \rightarrow B} \rightarrow: i$	$\frac{A \quad A \rightarrow B}{B} \rightarrow: e$

	<i>Einführung</i>	<i>Elimination</i>
\neg	$\boxed{\begin{array}{c} A \\ \vdots \\ \text{False} \end{array}} \frac{}{\neg A} \neg : i$	$\frac{A \quad \neg A}{\text{False}} \neg : e$
False		$\frac{\text{False}}{A} \text{ False: e}$
$\neg\neg$		$\frac{\neg\neg A}{A} \neg\neg : e$

Definition

Der Kalkül NK des **natürlichen Schließens** besteht aus den gerade betrachteten Beweisregeln.

Beweisbarkeitsrelation

Definition

Sei \mathcal{G} eine endliche Menge von Formeln und F eine Formel.

- Ein **Beweis von F aus \mathcal{G}** ist eine Folge von Formeln A_1, \dots, A_n mit $A_n = F$, sodass gilt: $A_i \in \mathcal{G}$ oder A_i folgt durch Anwendung einer der Regeln in NK.
- Eine Formel F heißt **beweisbar** aus den Annahmen \mathcal{G} , wenn es einen Beweis von F aus \mathcal{G} gibt.
- Ein Beweis wird oft auch als **Ableitung**, **Herleitung** oder **Deduktion** bezeichnet.

Definition

- 1 Die **Beweisbarkeitsrelation** $A_1, \dots, A_n \vdash B$ gilt, gdw. B aus A_1, \dots, A_n beweisbar ist.
- 2 Wir schreiben $\vdash A$ statt $\emptyset \vdash A$ und nennen A in diesem Fall **beweisbar**.

Beispiel

Wir betrachten die folgende Tautologie

$$a \rightarrow (b \rightarrow (c \rightarrow (d \rightarrow (e \rightarrow c))))))$$

1	a	Prämisse
2	b	Prämisse
3	c	Prämisse
4	d	Prämisse
5	e	Prämisse
6	c	3
7	$e \rightarrow c$	5, 6, \rightarrow : i
8	$d \rightarrow e \rightarrow c$	4, 7, \rightarrow : i
9	$c \rightarrow d \rightarrow e \rightarrow c$	3, 8, \rightarrow : i
10	$b \rightarrow c \rightarrow d \rightarrow e \rightarrow c$	2, 9, \rightarrow : i
11	$a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow c$	1, 10, \rightarrow : i

Versuchen Sie als Übung die Tautologie mit der Methode von Quine nachzuweisen.

Korrektheit und Vollständigkeit

Satz

Der Kalkül NK ist **korrekt** und **vollständig** für die Aussagenlogik:

$$A_1, \dots, A_n \models B \text{ gdw. } \Leftrightarrow A_1, \dots, A_n \vdash B$$



Fakt

Basierend auf einem korrekten und vollständigen Beweissystem können wir versuchen das Beweisen zu automatisieren \Rightarrow **SAT/SMT solvers**

Satz (Deduktionstheorem)

Gelte $A \vdash B$, dann existiert ein Beweis von $A \rightarrow B$, der A nicht als Prämisse hat

Beweis des Deduktionstheorems.

- Nach Annahme gilt $A_1, \dots, A_i, \dots, A_n \vdash B$.
- OBdA. können wir annehmen, dass $n = i$.
- Also gibt es einen Beweis in NK der folgenden Gestalt:

1	A_1	Prämissen
:	:	
$n - 1$	A_{n-1}	Prämissen
n	A_n	Prämissen
:	:	
k	B	

- Nun fügen wir diesem Beweis eine Anwendung der \rightarrow : i Regel auf die Formeln A_n und B hinzu, wodurch aus der Prämissen A_n eine (lokale) Annahme wird.

Beweis (Fortsetzung).

- Wir erhalten einen Beweis von $A_n \rightarrow B$:

1	A_1	Prämissen
\vdots	\vdots	
$n - 1$	A_{n-1}	Prämissen
n	A_n	Annahme
\vdots	\vdots	
k	B	
$k + 1$	$A_n \rightarrow B$	$\rightarrow : i$

- Somit ist die Prämissen A_n aus der Liste der Annahmen eliminiert und wir haben im Allgemeinen die Korrektheit des folgenden Sequents nachgewiesen:

$$A_1, \dots, A_{i-1}, A_{i+1}, \dots, A_n \vdash A_i \rightarrow B .$$



Beispiel

Wir betrachten die formale Ableitung der Formel

$$\neg\neg p \rightarrow p$$

1	$\neg\neg p$	Prämissen
2	p	$\neg\neg : e$
3	$\neg\neg p \rightarrow p$	1, 2, $\rightarrow : i$

Folgerung

- Die Formel $\neg\neg p \rightarrow p$ ist eine Tautologie.
- Die Formel $\neg\neg p \rightarrow p$ ist formal beweisbar.
- Wegen Korrektheit und Vollständigkeit gilt die folgende Äquivalenz:

$$\neg\neg p \models p \text{ gdw. } \neg\neg p \vdash p$$



Einführung in die Theoretische Informatik

Martin Avanzini Christian Dalvit Jamie Hochrainer

Georg Moser Johannes Niederhauser Jonas Schöpf

<https://tcs-informatik.uibk.ac.at>



Zusammenfassung

Zusammenfassung der letzten LVA

	<i>Einführung</i>	<i>Elimination</i>
\wedge	$\frac{A \quad B}{A \wedge B} \wedge: i$	$\frac{A \wedge B}{A} \wedge: e \quad \frac{A \wedge B}{B} \wedge: e$
\vee	$\frac{A}{A \vee B} \vee: i \quad \frac{B}{A \vee B} \vee: i$	$\frac{A \vee B}{\begin{array}{ c c } \hline A & B \\ \vdots & \vdots \\ C & C \\ \hline \end{array}} \vee: e$
\rightarrow	$\frac{\begin{array}{ c } \hline A \\ \vdots \\ B \\ \hline \end{array}}{A \rightarrow B} \rightarrow: i$	$\frac{A \quad A \rightarrow B}{B} \rightarrow: e$

	<i>Einführung</i>	<i>Elimination</i>
\neg	$\boxed{\begin{array}{c} A \\ \vdots \\ \text{False} \end{array}}$ $\frac{}{\neg A}$	$\neg : i$ $\frac{A \quad \neg A}{\text{False}}$ $\neg : e$
False		$\frac{\text{False}}{A}$ $\text{False} : e$
$\neg\neg$		$\frac{\neg\neg A}{A}$ $\neg\neg : e$

Definition

Der Kalkül NK des **natürlichen Schließens** besteht aus den gerade betrachteten Beweisregeln.

Einführung in die Logik

Syntax & Semantik der Aussagenlogik, Kalkül des natürlichen Schließens, Konjunktive und Disjunktive Normalformen

Einführung in die Algebra

algebraische Strukturen, Boolesche Algebra

Einführung in die Theorie der Formalen Sprachen

Grammatiken und Formale Sprachen, Reguläre Sprachen, Kontextfreie Sprachen, Chomsky-Hierarchie, Anwendungen von formalen Sprachen

Einführung in die Berechenbarkeitstheorie und Komplexitätstheorie

Algorithmisch unlösbare Probleme, Turing Maschinen, Registermaschinen, Komplexitätstheorie

Einführung in die Programmverifikation

Prinzipien der Analyse von Programmen, Verifikation nach Hoare

Beispiel (Wiederholung)

Wir betrachten die Ableitung der Formel $\neg\neg p \rightarrow p$

1	$\neg\neg p$	Prämissen
2	p	$\neg\neg : e$
3	$\neg\neg p \rightarrow p$	1, 2, $\rightarrow : i$

Beispiel

Wir betrachten die Ableitung der Umkehrung

$$p \rightarrow \neg\neg p$$

1	p	Prämissen
2	$\neg p$	Prämissen
3	False	1, 2, $\neg : e$
4	$\neg\neg p$	2, 3, $\neg : i$

Beispiel (Abgeleitete Regel $\neg\neg$: i)

Mit der selben Ableitung erhalten wir die folgende (**abgeleitete**) Inferenzregel:

$$\frac{A}{\neg\neg A} \quad \neg\neg : i$$

NB: Wir schreiben Inferenzregeln immer mit den Metavariablen für Formeln $A, B, C \dots$

Beispiel

Wir betrachten noch eine weitere abgeleitete Inferenzregel, nämlich den **Widerspruchsbeweis (WB)**:

$$\frac{\begin{array}{c} \neg A \\ \vdots \\ \text{False} \end{array}}{A} \quad \text{WB}$$

Beispiel (Abgeleitete Regel WB)

Die Ableitung der Regel WB gelingt wie folgt:

1	$\neg A \rightarrow \text{False}$	Prämissen, $\rightarrow: i$
2	$\neg A$	Prämissen
3	False	1, 2, $\rightarrow: e$
4	$\neg\neg A$	2,3, $\neg: i$
5	A	4, $\neg\neg: e$

Beispiel

Nun wollen wir noch $p \vee q \vdash q \vee p$ zeigen:

1	$p \vee q$	Prämissen
2	p	Prämissen
3	$q \vee p$	2, $\vee: i$
4	q	Prämissen
5	$q \vee p$	4, $\vee: i$
6	$q \vee p$	1,2-3,4-5, $\vee: e$

Diskurs: Axiome für die Aussagenlogik nach Frege und Łukasiewicz

- Der Kalkül NK des natürlichen Schließens ist (beileibe) nicht der einzige korrekte und vollständige Kalkül für die Aussagenlogik.

Definition

Axiome für die Aussagenlogik nach Frege und Łukasiewicz

- (1) $A \rightarrow (B \rightarrow A)$
- (2) $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
- (3) $(\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A)$

Satz

Das Axiomensystem nach Frege und Łukasiewicz mit Inferenzregel Modus Ponens ist korrekt und vollständig für die Aussagenlogik.



Konjunktive und Disjunktive Normalformen

Definition

Eine **Wahrheitsfunktion** $f: \{\text{T}, \text{F}\}^n \rightarrow \{\text{T}, \text{F}\}$ ist eine Funktion, die n Wahrheitswerten einen Wahrheitswert zuordnet
(vgl. [Rechnerarchitektur](#))

Definition

Sei $f: \{\text{T}, \text{F}\}^n \rightarrow \{\text{T}, \text{F}\}$ eine Wahrheitsfunktion; wir definieren:

$$\text{TV}(f) := \{(s_1, \dots, s_n) \mid f(s_1, \dots, s_n) = \text{T}\}$$

Definition (Konjunktive und Disjunktive Normalform)

- 1 Ein **Literal** ist ein Atom p oder die Negation eines Atoms $\neg p$
- 2 Formel A ist in **disjunktiver Normalform (DNF)**, wenn A eine Disjunktion von Konjunktionen von Literalen
- 3 Formel A ist in **konjunktiver Normalform (KNF)**, wenn A eine Konjunktion von Disjunktionen von Literalen

- $f: \{\text{T}, \text{F}\}^n \rightarrow \{\text{T}, \text{F}\}$ eine Wahrheitsfunktion
 $\text{TV}(f) \neq \emptyset, \text{TV}(f) \neq \{\text{T}, \text{F}\}^n$
- p_1, \dots, p_n atomare Formeln
- Sei DNF D definiert als:

$$D := \bigvee_{(s_1, \dots, s_n) \in \text{TV}(f)} \bigwedge_{i=1}^n A_i$$

wobei $A_i = p_i$, wenn $s_i = \text{T}$ und $A_i = \neg p_i$ sonst

- Sei KNF K definiert als:

$$K := \bigwedge_{(s_1, \dots, s_n) \notin \text{TV}(f)} \bigvee_{j=1}^n B_j$$

wobei $B_j = \neg p_j$, wenn $s_j = \text{T}$ und $B_j = p_j$ sonst

- Die Wahrheitstabellen von D und K entsprechen der Wahrheitsfunktion f

Satz

- 1 Jede Wahrheitsfunktion kann als DNF oder KNF ausgedrückt werden
- 2 Jede Formel mit n Atomen induziert eine Wahrheitsfunktion in n Variablen

Beweis.

1 Es fehlen die Fälle, wo die Wahrheitsfunktion trivial ist:

- $\text{TV}(f) = \emptyset$
- $\text{TV}(f) = \{\text{T}, \text{F}\}^n$

2 Setze $D = K := \bigwedge_{i=1}^n (p_i \wedge \neg p_i)$ im ersten Fall

3 Setze $D = K := \bigvee_{i=1}^n (p_i \vee \neg p_i)$ im zweiten Fall



Folgerung

Für jede Formel A existiert eine DNF D und eine KNF K , sodass $A \equiv D \equiv K$ gilt.

Beispiel

Die folgende Operation (\oplus) wird XOR genannt:

p	q	$p \oplus q$
F	F	F
F	T	T
T	F	T
T	T	F

Wir erstellen die KNF.

$$TV(\oplus) = \{(F, T), (T, F)\}$$

p_1	p_2	$p_1 \oplus p_2$	Disjunktion	KNF
F	F	F	$p_1 \vee p_2$	
T	T	F	$\neg p_1 \vee \neg p_2$	$(p_1 \vee p_2) \wedge (\neg p_1 \vee \neg p_2)$



Algebraische Strukturen

Definition (Algebra)

Eine **Algebra** $\mathcal{A} = \langle A_1, \dots, A_n; o_1, \dots, o_m \rangle$ besteht aus

- 1 **Träger** (oder **Trägermengen**) A_1, \dots, A_n
- 2 **Operationen** o_1, \dots, o_m auf den Trägern

Nullstellige Operationen werden auch **Konstanten** genannt; wir fixieren eine unendliche Menge von **Variablen** x_1, x_2, \dots und für jede Operation o_i der Algebra \mathcal{A} ein Symbol o_i der gleichen Stelligkeit

Definition (Algebraische Ausdrücke)

Wir definieren die **algebraischen Ausdrücke** einer Algebra \mathcal{A} induktiv:

- 1 Konstanten und Variablen sind algebraische Ausdrücke.
- 2 Wenn A_1, \dots, A_n algebraische Ausdrücke, \circ eine Operation, dann ist $\circ(A_1, \dots, A_n)$ ein algebraischer Ausdruck

Definition

Seien A und B algebraische Ausdrücke

- A und B sind äquivalent, wenn \forall Instanzen A' und B' gilt: $A' = B'$
- Wenn A äquivalent zu B ist, schreiben wir kurz $A \approx B$

Definition

Wenn die Träger von \mathcal{A} endlich sind, dann nennen wir \mathcal{A} endlich

Beispiel

Sei $A = \{a, b, c, d\}$ und \circ durch folgende Operationstabelle definiert:

\circ	a	b	c	d
a	a	b	c	d
b	b	c	d	a
c	c	d	a	c
d	d	a	b	c

Nullelement, neutrales Element, Inverses

Definition

Sei \circ eine binäre Operation auf A

- Wenn $0 \in A$ existiert, sodass für alle $a \in A$

$$a \circ 0 = 0 \circ a = 0$$

dann heißt 0 **Nullelement** für \circ

- Wenn $1 \in A$ existiert, sodass für alle $a \in A$

$$a \circ 1 = 1 \circ a = a$$

dann heißt 1 **Einselement (neutrales Element)** für \circ

- Sei 1 das neutrale Element für \circ und für $a \in A$, existiert $b \in A$, sodass

$$a \circ b = b \circ a = 1$$

Dann heißt b das **Inverse (Komplement)** von a

Halbgruppen, Monoide und Gruppen

Definition

Eine Algebra $\mathcal{A} = \langle A; \circ \rangle$ heißt

- **Halbgruppe**, wenn \circ assoziativ
- **Monoid**, wenn $\mathcal{A} = \langle A; \circ, 1 \rangle$ eine Halbgruppe mit Einselement 1 für \circ
- **Gruppe**, wenn \mathcal{A} ein Monoid ist und jedes Element ein Inverses hat

Eine Halbgruppe, ein Monoid oder eine Gruppe heißt **kommutativ**, wenn \circ kommutativ

Beispiel

Die im vorigen Beispiel definierte Algebra \mathcal{A} hat folgende Eigenschaften:

- 1 a ist das neutrale Element von \circ
- 2 Jedes Element besitzt ein Inverses
- 3 \circ ist nicht kommutativ

Eigenschaft des neutralen Elements

Lemma

Jede binäre Operation hat maximal ein neutrales Element

Beweis.

- 1 Sei \circ eine binäre Operation auf der Menge A
- 2 Angenommen e und u sind neutrale Elemente für \circ
- 3 Wir zeigen, dass $e = u$

$$\begin{aligned} e &= e \circ u && \text{da } u \text{ Einselement} \\ &= u && \text{da } e \text{ Einselement} \end{aligned}$$

Eigenschaft des Inversen

Lemma

Wenn $\mathcal{A} = \langle A; \circ, 1 \rangle$ ein Monoid ist, dann ist das Inverse eindeutig

Beweis.

Sei $a \in A$ und seien b, c Inverse von a . Wir zeigen $b = c$:

$$\begin{aligned} b &= b \circ 1 && 1 \text{ ist neutrales Element} \\ &= b \circ (a \circ c) && c \text{ ist Komplement von } a \\ &= (b \circ a) \circ c && \text{Assoziativitat von } \circ \\ &= 1 \circ c && b \text{ ist Komplement von } a \\ &= c && 1 \text{ ist neutrales Element} \end{aligned}$$

Ringe und Körper

Definition (Ring)

Eine Algebra $\mathcal{A} = \langle A; +, \cdot, 0, 1 \rangle$ heißt **Ring**, wenn

- 1** $\langle A; +, 0 \rangle$ eine kommutative Gruppe
- 2** $\langle A; \cdot, 1 \rangle$ ein Monoid
- 3** \cdot distributiert über $+$ (von links und von rechts),
das heißt für alle $a, b, c \in A$ gilt:

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c) \quad (b + c) \cdot a = (b \cdot a) + (c \cdot a)$$

Definition (Körper)

Eine Algebra $\mathcal{A} = \langle A; +, \cdot, 0, 1 \rangle$ heißt **Körper**, wenn

- 1** \mathcal{A} ein Ring
- 2** $\langle A \setminus \{0\}; \cdot, 1 \rangle$ eine kommutative Gruppe



Einführung in die Theoretische Informatik

Martin Avanzini Christian Dalvit Jamie Hochrainer

Georg Moser Johannes Niederhauser Jonas Schöpf

<https://tcs-informatik.uibk.ac.at>



Zusammenfassung

Zusammenfassung der letzten LVA

Folgerung

Für jede Formel A existiert eine DNF D und eine KNF K , sodass $A \equiv D \equiv K$ gilt.

Definition (Algebra)

Eine **Algebra** $\mathcal{A} = \langle A_1, \dots, A_n; \circ_1, \dots, \circ_m \rangle$ besteht aus

- 1 Träger (oder Trägermengen) A_1, \dots, A_n
- 2 Operationen \circ_1, \dots, \circ_m auf den Trägern

Lemma

Jede binäre Operation hat maximal ein neutrales Element

Lemma

Wenn $\mathcal{A} = \langle A; \circ, 1 \rangle$ ein Monoid ist, dann ist das Inverse eindeutig

Einführung in die Logik

Syntax & Semantik der Aussagenlogik, Kalkül des natürlichen Schließens, Konjunktive und Disjunktive Normalformen

Einführung in die Algebra

algebraische Strukturen, Boolesche Algebra

Einführung in die Theorie der Formalen Sprachen

Grammatiken und Formale Sprachen, Reguläre Sprachen, Kontextfreie Sprachen, Chomsky-Hierarchie, Anwendungen von formalen Sprachen

Einführung in die Berechenbarkeitstheorie und Komplexitätstheorie

Algorithmisch unlösbare Probleme, Turing Maschinen, Registermaschinen, Komplexitätstheorie

Einführung in die Programmverifikation

Prinzipien der Analyse von Programmen, Verifikation nach Hoare

Definition (Boolesche Algebra)

Eine Algebra $\mathcal{B} = \langle B; +, \cdot, \sim, 0, 1 \rangle$ heißt **Boolesche Algebra** wenn gilt:

1 $\langle B; +, 0 \rangle$ und $\langle B; \cdot, 1 \rangle$ sind kommutative Monoide

2 Die Operationen $+$ und \cdot distribuieren übereinander. Es gilt also für alle $a, b, c \in B$:

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c) \quad a + (b \cdot c) = (a + b) \cdot (a + c)$$

3 Für alle $a \in B$ gilt

$$a + \sim(a) = 1 \quad a \cdot \sim(a) = 0$$

Das Element $\sim(a)$ heißt das **Komplement** oder die **Negation** von a

Konventionen

- Wir lassen \cdot oft weg und schreiben ab statt $a \cdot b$
- Wir verwenden die folgende Präzedenz: \sim bindet stärker als $+$ und \cdot .
- Die Definition ist eine Verallgemeinerung der Definition in Rechnerarchitektur

Definition (Boolescher Ausdruck)

Sei eine unendliche Menge von Variablen x_1, x_2, \dots gegeben; diese Variablen heißen **Boolesche Variablen**

Wir definieren **Boolesche Ausdrücke** induktiv:

- 1** 0, 1 und Variablen sind Boolesche Ausdrücke
- 2** Wenn E und F Boolesche Ausdrücke sind, dann sind

$$\sim(E) \quad (E \cdot F) \quad (E + F)$$

Boolesche Ausdrücke A und B heißen **äquivalent** ($A \approx B$), wenn für alle Booleschen Algebren, in allen Instanzen A' und B' gilt: $A' = B'$.

Beispiel (vgl Rechnerarchitektur)

Die folgenden Ausdrücke sind Boolesche Ausdrücke:

$$x_1 \quad x_2 \quad x_1 + x_2 \quad x_1 \cdot x_2 \quad x_1 \cdot (x_1 + x_2) \quad x_1(x_1 + x_2) \quad x_1 \sim (x_1 + x_2)$$

Mengenalgebra

Sei M eine Menge; $\mathcal{P}(M)$ bezeichnet die **Potenzmenge** von M , also

$$\mathcal{P}(M) := \{N \mid N \subseteq M\}$$

Definition

Wir betrachten die Algebra

$$\langle \mathcal{P}(M); \cup, \cap, \sim, \emptyset, M \rangle$$

- 1** \cup die Mengenvereinigung
- 2** \cap die Schnittmenge
- 3** \sim die Komplementärmenge

Diese Algebra nennt man **Mengenalgebra**.

Lemma

Die Mengenalgebra ist eine Boolesche Algebra

Binäre Algebra

Definition

Sei $\mathbb{B} := \{0, 1\}$, wobei $0, 1 \in \mathbb{N}$. Wir betrachten die Algebra

$$\langle \mathbb{B}; +, \cdot, \sim, 0, 1 \rangle$$

wobei die Operationen $+, \cdot, \sim$ wie folgt definiert:

\cdot	1	0	$+$	1	0	\sim	
1	1	0	1	1	1	1	0
0	0	0	0	1	0	0	1

Diese Algebra nennt man **binäre Algebra** oder Boolesche Algebra im **engeren Sinn** (Rechnerarchitektur)

Lemma

Die binäre Algebra ist eine Boolesche Algebra

Algebra der Aussagenlogik

Sei Frm die Menge der aussagenlogischen Formeln

Definition

Wir betrachten die Algebra \mathcal{Frm}

$$\langle \text{Frm}; \vee, \wedge, \neg, \text{False}, \text{True} \rangle$$

Wobei die Zeichen wie in der Aussagenlogik interpretiert werden und Gleichheit von Booleschen Ausdrücken logische Äquivalenz bedeutet

Lemma

Die Algebra \mathcal{Frm} ist eine Boolesche Algebra

Algebra des Kartesischen Produkts und der Schaltfunktionen

Definition

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

$$\langle \mathbb{B}^n; +, \cdot, \sim, (0, \dots, 0), (1, \dots, 1) \rangle$$

- 1** $(a_1, \dots, a_n) + (b_1, \dots, b_n) = (a_1 + b_1, \dots, a_n + b_n)$
- 2** $(a_1, \dots, a_n) \cdot (b_1, \dots, b_n) = (a_1 \cdot b_1, \dots, a_n \cdot b_n)$
- 3** $\sim((a_1, \dots, a_n)) = (\sim(a_1), \dots, \sim(a_n))$

Lemma

Die oben definierte Algebra ist eine Boolesche Algebra

Definition

Sei Abb die Menge der Abbildungen von \mathbb{B}^n nach \mathbb{B}^m wir betrachten

$$\langle \text{Abb}; +, \cdot, \sim, (\mathbf{0}, \dots, \mathbf{0}), (\mathbf{1}, \dots, \mathbf{1}) \rangle$$

- 1** $(\mathbf{0}, \dots, \mathbf{0}): (a_1, \dots, a_n) \mapsto (0, \dots, 0)$
- 2** $(\mathbf{1}, \dots, \mathbf{1}): (a_1, \dots, a_n) \mapsto (1, \dots, 1)$
- 3** $(f + g)(a_1, \dots, a_n) = f(a_1, \dots, a_n) + g(a_1, \dots, a_n)$
- 4** $(f \cdot g)(a_1, \dots, a_n) = f(a_1, \dots, a_n) \cdot g(a_1, \dots, a_n)$
- 5** $\sim(f)(a_1, \dots, a_n) = \sim(f(a_1, \dots, a_n))$

Diese Algebra nennt man **Algebra der Schaltfunktionen** oder *n*-stelligen Booleschen Funktionen

Lemma

Die Algebra der Schaltfunktionen ist eine Boolesche Algebra

Lemma

Die Mengenalgebra ist eine Boolesche Algebra

Beweis.

Wir müssen zeigen, dass

1 $\langle \mathcal{P}(M); \cup, \emptyset \rangle$ sowie $\langle \mathcal{P}(M); \cap, M \rangle$ kommutative Monoide sind

2 seien $A, B, C \subseteq M$, dann gilt $A, B, C \subseteq M \rightarrow A, B, C$ beibehalten in M .

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \quad A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

3 für alle $A \subseteq M$ gilt

$$A \cup \sim(A) = M \quad A \cap \sim(A) = \emptyset$$

Wir beginnen mit den **Gesetzen zum Komplement**; dazu beschränken wir uns auf $A \cap \sim(A) = \emptyset$, der Beweis für $A \cup \sim(A) = M$ ist ganz ähnlich

$$A \cap \sim(A) = A \cap \{x \in M \mid x \notin A\} = \{x \in M \mid x \in A \text{ und } x \notin A\} = \emptyset$$

Beweis (Fortsetzung).

Wir müssen also noch zeigen, dass

- 1 $\langle \mathcal{P}(M); \cup, \emptyset \rangle$ sowie $\langle \mathcal{P}(M); \cap, M \rangle$ kommutative Monoide sind
- 2 seien $A, B, C \subseteq M$, dann gilt

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \quad A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

Die Korrektheit der **Distributivgesetze** folgt leicht aus den Definitionen der Mengenoperationen (nachrechnen!)

Zeigen wir nun also, dass $\langle \mathcal{P}(M); \cup, \emptyset \rangle$ ein kommunitative Monoid ist; dazu zeigen wir

- \cup ist assoziativ : $A \cup (B \cup C) = (A \cup B) \cup C$
- \emptyset ist das neutrale Element für \cup auf $\mathcal{P}(M)$: $A \cup \emptyset = \emptyset \cup A = A$

Beide Gleichungen folgen aus der Definition der Vereinigung.

Ebenso zeigt man, dass $\langle \mathcal{P}(M); \cap, M \rangle$ ein kommunitative Monoid ist. ■

Gesetze Boolescher Algebren

die noch nicht in Rechnerarchitektur behandelt wurden

Lemma ①

Für alle $a, b \in B$ gilt die **Eindeutigkeit des Komplements**:

Wenn $a + b = 1$ und $ab = 0$, dann $b = \sim(a)$

Beweis.

Gelte $a + b = 1$ und $ab = 0$

$$\begin{aligned} b &= b1 = b(a + \sim(a)) \\ &= ba + b \cdot \sim(a) = 0 + b \cdot \sim(a) && \text{da } ba = ab = 0 \\ &= a \cdot \sim(a) + b \cdot \sim(a) = (a + b) \cdot \sim(a) \\ &= 1 \cdot \sim(a) && \text{da } a + b = 1 \\ &= \sim(a) \end{aligned}$$

Lemma

Für alle $a \in B$ gilt das **Involutionsgesetz**:

$$\sim(\sim(a)) = a$$

Beweis.

Nach Definition einer Booleschen Algebra und Kommutativität von + beziehungsweise · gilt:

1 $\sim(a) + a = 1$

2 $\sim(a) \cdot a = 0$

Mit Lemma ① folgt, dass a das Komplement von $\sim(a)$ ist



Lemma

Für alle $a, b \in B$ gelten die **Gesetze von de Morgan**:

$$\sim(a + b) = \sim(a) \cdot \sim(b) \quad \sim(a \cdot b) = \sim(a) + \sim(b)$$

Idempotenz und Absorption

bereits in Rechnerarchitektur behandelt

Lemma

Für alle $a \in B$ gelten die **Idempotenzgesetze**:

$$a \cdot a = a \quad a + a = a$$

und die folgenden Gesetze für 0 und 1 (**Substitution**):

$$0 \cdot a = 0 \quad 1 + a = 1$$

Lemma

Für alle $a, b \in B$ gelten die **Absorptionsgesetze**:

$$a + ab = a \quad a(a + b) = a$$

$$a + \sim(a) \cdot b = a + b \quad a(\sim(a) + b) = ab$$

Erstes Gesetz von de Morgan.

- Wir zeigen $(a + b) + (\sim(a) \cdot \sim(b)) = 1$:

$$\begin{aligned}(a + b) + (\sim(a) \cdot \sim(b)) &= (a + b + \sim(a))(a + b + \sim(b)) \\&= (a + \sim(a) + b)(a + b + \sim(b)) \\&= (1 + b)(a + 1) \\&= 1 \cdot 1 = 1\end{aligned}$$

- Wir zeigen $(a + b) \cdot (\sim(a) \cdot \sim(b)) = 0$:

$$\begin{aligned}(a + b) \cdot \sim(a) \cdot \sim(b) &= a \cdot \sim(a) \cdot \sim(b) + b \cdot \sim(a) \cdot \sim(b) \\&= a \cdot \sim(a) \cdot \sim(b) + \sim(a) \cdot b \cdot \sim(b) \\&= 0 \cdot \sim(b) + \sim(a) \cdot 0 \\&= 0 + 0 = 0\end{aligned}$$

- Die Voraussetzungen von Lemma ① sind gezeigt
- Somit ist $\sim(a) \cdot \sim(b)$ das Komplement von $a + b$, wzzw.

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 $f: \mathbb{B}^n \rightarrow \mathbb{B}$ wie folgt:

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

Dann heißt f die **Boolesche Funktion** zum Ausdruck F

Beispiel (Boolesche Algebra) $\mathcal{Frm} = \langle \text{Frm}; \vee, \wedge, \neg, \text{False}, \text{True} \rangle$

Sei $F = x_1 \wedge \neg(x_2 \vee x_1)$, dann ist $f: \mathbb{B}^2 \rightarrow \mathbb{B}$
die Boolesche Funktion zu F

Sei $G = x_1 \wedge x_2 \wedge \neg x_2$, dann ist $g: \mathbb{B}^2 \rightarrow \mathbb{B}$
die Boolesche 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

Definition

- 1 Sei $f: \mathbb{B}^n \rightarrow \mathbb{B}$ eine Boolesche Funktion
- 2 Sei F ein Boolescher Ausdruck, dessen Boolesche Funktion gleich f

Dann nennen wir F den **Booleschen Ausdruck** von f

Satz (Darstellungssatz von Stone)

Jede Boolesche Algebra ist isomorph zu einer Mengenalgebra

Bemerkung

- Isomorphie bedeutet, dass die Operationen auf den Algebren ident sind.
- Der Darstellungssatz von Stone bedeutet also, dass jede Gleichheit in **einer** Mengenalgebra eine Gleichheit für **alle** Booleschen Algebren ist.
- Anders ausgedrückt stellen Mengenalgebren die eindeutige Darstellung von Booleschen Algebren dar.

Folgerung aus dem Darstellungssatz von Stone

Folgerung

- 1 Seien A, B Boolesche Ausdrücke
- 2 Seien f, g ihre Booleschen Funktionen

Dann sind A und B **äquivalent** ($A \approx B$), wenn $f = g$ in der Algebra der Booleschen Funktionen gilt

Beweisskizze

- Äquivalenzen von Boolesche Ausdrücke gelten (per Definition) für alle Booleschen Algebren.
- Um diese Äquivalenzen zu überprüfen genügt (nach der Definition) die Verifikation in einer bestimmten Algebra, nämlich der Algebra der Booleschen Funktionen; das folgt aus dem Darstellungssatz von Stone



Einführung in die Theoretische Informatik

Martin Avanzini Christian Dalvit Jamie Hochrainer

Georg Moser Johannes Niederhauser Jonas Schöpf

<https://tcs-informatik.uibk.ac.at>



Zusammenfassung

Erinnerung: Natürliches Schließen

	<i>Einführung</i>	<i>Elimination</i>
\neg	$\begin{array}{c} A \\ \vdots \\ \text{False} \end{array} \frac{}{\neg A} \neg : i$	$\frac{A \quad \neg A}{\text{False}} \neg : e$
False		$\frac{\text{False}}{A} \text{ False: e}$
$\neg\neg$		$\frac{\neg\neg A}{A} \neg\neg : e$

Satz

Der Kalkül NK ist **korrekt** und **vollständig** für die Aussagenlogik:

$$A_1, \dots, A_n \models B \quad gdw. \quad A_1, \dots, A_n \vdash B$$

Zusammenfassung der letzten LVA

Definition (Boolesche Algebra)

Eine Algebra $\mathcal{B} = \langle B; +, \cdot, \sim, 0, 1 \rangle$ heißt **Boolesche Algebra** wenn gilt:

- 1 $\langle B; +, 0 \rangle$ und $\langle B; \cdot, 1 \rangle$ sind kommutative Monoide
- 2 Die Operationen $+$ und \cdot distribuieren übereinander. Es gilt also für alle $a, b, c \in B$:

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c) \quad a + (b \cdot c) = (a + b) \cdot (a + c)$$

- 3 Für alle $a \in B$ gilt

$$a + \sim(a) = 1 \quad a \cdot \sim(a) = 0$$

Das Element $\sim(a)$ heißt das **Komplement** oder die **Negation** von a

Satz (Darstellungssatz von Stone)

Jede Boolesche Algebra \mathcal{B} ist isomorph zu einer Mengenalgebra $\langle \mathcal{P}(M); \cup, \cap, \sim, \emptyset, M \rangle$, wobei M geeignet aus Elementen von \mathcal{B} gewählt wird.

Einführung in die Logik

Syntax & Semantik der Aussagenlogik, Kalkül des natürlichen Schließens, Konjunktive und Disjunktive Normalformen

Einführung in die Algebra

algebraische Strukturen, Boolesche Algebra

Einführung in die Theorie der Formalen Sprachen

Grammatiken und Formale Sprachen, Reguläre Sprachen, Kontextfreie Sprachen, Chomsky-Hierarchie, Anwendungen von formalen Sprachen

Einführung in die Berechenbarkeitstheorie und Komplexitätstheorie

Algorithmisch unlösbare Probleme, Turing Maschinen, Registermaschinen, Komplexitätstheorie

Einführung in die Programmverifikation

Prinzipien der Analyse von Programmen, Verifikation nach Hoare



Boolesche Algebren

Isomorphie

vgl. Lineare Algebra

Definition

Seien $\mathcal{A} = \langle A; \circ_1, \dots, \circ_m \rangle$, $\mathcal{B} = \langle B; \odot_1, \dots, \odot_m \rangle$ Algebren, dann heißt eine Abbildung $\varphi: A \rightarrow B$ ein **Isomorphismus** zwischen \mathcal{A} und \mathcal{B} , wenn gilt

- φ ist bijektiv
- für alle Operationen \circ_i von \mathcal{A} (\circ_i n -stellig) gilt:

$$\varphi(\circ_i(a_1, \dots, a_n)) = \odot_i(\varphi(a_1), \dots, \varphi(a_n)),$$

für alle $a_1, \dots, a_n \in A$.

Definition

Eine Algebra $\mathcal{A} = \langle A; \circ_1, \dots, \circ_m \rangle$ heißt **isomorph** zur Algebra $\mathcal{B} = \langle B; \odot_1, \dots, \odot_m \rangle$, wenn ein Isomorphismus $\varphi: A \rightarrow B$ existiert. Wir schreiben $\mathcal{A} \cong \mathcal{B}$.

Beispiel

- Betrachte die Monoide $\langle \{a, b\}, + \rangle$ und $\langle \{0, 1\}, \cdot \rangle$, wobei die Operationen $+$ bzw. \cdot durch die folgenden Operationstafeln definiert sind:

+ \ a \ b	0 \ 0 \ 1
a \ \ a \ b	0 \ \ 0 \ 1
b \ \ b \ a	1 \ \ 1 \ 0

- Dann ist die Abbildung $\varphi: \{a, b\} \rightarrow \{0, 1\}$ mit

$$\varphi(a) := 0 \quad \varphi(b) := 1 ,$$

ein Isomorphismus

Bemerkung

Wir interessieren uns besonders für Isomorphismen zwischen Booleschen Algebren und können damit den Darstellungsatz von Stone exakt definieren.

Beispiel

- Sei $\langle \mathbb{B}; +, \cdot, \sim, 0, 1 \rangle$ die binäre Algebra
- Sei $\langle \mathcal{P}(M); \cup, \cap, \sim, \emptyset, M \rangle$ die Mengenalgebra, mit der Menge $M := \{a\}$. Wir können diese Mengenalgebra einfacher wie folgt schreiben:

$$\langle \{\emptyset, \{a\}\}; \cup, \cap, \sim, \emptyset, \{a\} \rangle$$

- Sei $\varphi: \mathbb{B} \rightarrow \{\emptyset, \{a\}\}$ wie folgt definiert

$$\varphi(0) := \emptyset \quad \varphi(1) := \{a\}$$

- Dann ist φ eine bijektive Funktion und außerdem sogar ein Isomorphismus:

\cup	$\{a\}$	\emptyset	\cap	$\{a\}$	\emptyset	\sim	$\{a\}$	\emptyset
$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$	\emptyset	\emptyset	$\{a\}$	\emptyset	\emptyset
\emptyset	$\{a\}$	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	$\{a\}$	$\{a\}$

Partielle Ordnungen und Boolesche Algebren

Definition

Eine **partielle Ordnung** auf einer Menge $M \neq \emptyset$ ist eine Menge von geordneten Paaren $(a, b) \in M \times M$, geschrieben $a \leq b$, sodass gilt

- $a \leq a$, für alle $a \in M$ Reflexivität
- $a \leq b$ und $b \leq c$ impliziert $a \leq c$, für alle $a, b, c \in M$ Transitivität
- $a \leq b$ und $b \leq a$ impliziert $a = b$, für alle $a, b \in M$ Antisymmetrie

Fakt

- Sei $\mathcal{B} = \langle B; +, \cdot, \sim, 0, 1 \rangle$ eine Boolesche Algebra und definiere Relation \leq auf B :
$$a \leq b \Leftrightarrow a \cdot b = a$$
- \leq ist eine partielle Ordnung

Beweis des Darstellungsatz von Stone (I)

- Sei $\mathcal{B} = \langle B; +, \cdot, \sim, 0, 1 \rangle$ eine (endliche) Boolesche Algebra;
- sei \leq , die von \mathcal{B} induzierte partielle Ordnung.

Definition

- Sei $a \in B \setminus \{0\}$.
- Wenn $0 \leq a$ und kein $a' \in B \setminus \{0\}$, $a \neq a'$ existiert, sodass $0 \leq a' \leq a$, dann nennen wir a ein **Atom**.

Kürzer: die Atome sind die oberen Nachbarn von 0 in B .

Beispiel

Sei $\langle \mathbb{B}; +, \cdot, \sim, 0, 1 \rangle$ die binäre Algebra, dann ist $1 \in \mathbb{B}$ ein Atom. Es gibt kein Element $\neq 0$ in \mathbb{B} gibt, das größer als 0 und (echt) kleiner als 1 ist.

Beweis des Darstellungsatz von Stone (II)

Lemma

Zu jedem $b \in B \setminus \{0\}$ gibt es mindestens ein Atom $a \in B$ mit $a \leq b$. ■

Konstruktion

- 1 Sei $M := \{a \in B \mid a \text{ ein Atom in } \mathcal{B}\}$.
- 2 Wir betrachten nun die Mengenalgebra $\langle \mathcal{P}(M); \cup, \cap, \sim, \emptyset, M \rangle$.
- 3 Für jedes $b \in B$, definiere $A(b) := \{a \in B \mid a \text{ ein Atom in } \mathcal{B} \text{ und } a \leq b\}$.
- 4 Schließlich definieren wir die Abbildung $\varphi: B \rightarrow \mathcal{P}(M)$, sodass $\varphi(b) := A(b)$.

Lemma

Die Abbildung φ ist ein Isomorphismus von $\langle B; +, \cdot, \sim, 0, 1 \rangle$ auf $\langle \mathcal{P}(M); \cup, \cap, \sim, \emptyset, M \rangle$. ■



Formale Sprachen

Definition (Alphabet)

Ein **Alphabet** Σ ist eine endliche, nicht leere Menge von Symbolen

Beispiel

- $\Sigma = \{0, 1\}$ ist das **binäre** Alphabet
- $\Sigma = \{a, b, \dots, z\}$, die Menge aller Kleinbuchstaben
- die Menge der (druckbaren) ASCII-Zeichen

Definition (Wort)

- Eine **Zeichenreihe** (ein **Wort**, ein **String**) ist eine endliche Folge von Symbolen über einem Alphabet Σ
- Die **leere Zeichenreihe** wird mit ϵ bezeichnet

Beispiel

Die Symbolkette 01101 ist eine Zeichenreihe über dem Alphabet $\{0, 1\}$

Konvention

- Buchstaben werden mit a, b, c, \dots bezeichnet
- Wörter werden mit x, y, z, \dots bezeichnet
- $\epsilon \notin \Sigma$

Definition (Wortlänge)

- Die **Länge** eines Wortes w ist die Anzahl der Positionen in w
- Die Länge von w wird auch mit $|w|$ bezeichnet
- Das Leerwort ϵ hat die Länge 0

Definition (Σ^k , Σ^+ , Σ^*)

- Definiere Σ^k als die Menge
der Wörter der Länge k , deren Symbole aus Σ stammen Σ^k
- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \dots$ Σ^+
- $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$ Σ^*

Beispiel

Sei $\Sigma = \{0, 1\}$. Dann ist

- $\Sigma^0 = \{\epsilon\}$
- $\Sigma^1 = \{0, 1\}$
- $\Sigma^2 = \{00, 01, 10, 11\}$
- $\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

Definition

Seien x, y Wörter über Σ , wir schreiben $x \cdot y$ für die **Konkatenation** von x und y

$$\epsilon \cdot x = x$$

$$(ax) \cdot y = a(x \cdot y)$$

Hier gilt $a \in \Sigma$.

Beispiel

- Sei $x = 01101$, $y = 110$, $z = 10101$
- Dann ist $x \cdot y = 01101110$ und $y \cdot x = 11001101$

Lemma

- Konkatenation ist assoziativ und besitzt das Leerwort ϵ als neutrales Element
- Wir lassen \cdot oft weg und schreiben xy statt $x \cdot y$
- Die Algebra $\langle \Sigma^*; \cdot, \epsilon \rangle$ ist ein Monoid; das **Wortmonoid**

Formale Sprachen

Definition

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

Beispiel

- Die Sprache aller Wörter, die aus n 0en gefolgt von n 1er bestehen, wobei $n \geq 0$:
$$\{\epsilon, 01, 0011, 000111, \dots\}$$
- Die Menge der Wörter, die jeweils die selbe Anzahl 0en und 1er enthalten:
$$\{\epsilon, 01, 10, 0011, 0101, \dots\}$$
- Σ^* ist eine Sprache, \emptyset —die leere Sprache—is eine Sprache, $\{\epsilon\}$ ist eine Sprache.
Beachte $\{\epsilon\} \neq \emptyset$

Definition

Seien L, M formale Sprachen über dem Alphabet Σ

- Die **Vereinigung** von L und M ist wie folgt definiert

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

- Wir definieren das **Komplement von L :**

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

- Der **Durchschnitt** von L und M ist wie folgt definiert:

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

- Das **Produkt** (oder **Verkettung**) von L und M ist definiert als:

$$LM = \{xy \mid x \in L, y \in M\}$$

Lemma

Seien L, L_1, L_2, L_3 formale Sprachen, dann gilt

$$(L_1 L_2) L_3 = L_1 (L_2 L_3) \quad L\{\epsilon\} = \{\epsilon\}L = L$$

Definition

Sei L eine formale Sprache und $k \in \mathbb{N}$

Die **k -te Potenz** von L definiert als:

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

Definition

Der **Kleene-Stern** * oder **Abschluss** von L ist wie folgt definiert:

$$L^* = \bigcup_{k \geq 0} L^k = \{x_1 \cdots x_k \mid x_1, \dots, x_k \in L \text{ und } k \geq 0\}$$

Definition

Schließlich definieren wir:

$$L^+ = \bigcup_{k \geq 1} L^k = \{x_1 \cdots x_k \mid x_1, \dots, x_k \in L \text{ und } k > 0\}$$

Beispiel

- Sei $\Sigma = \{0, 1\}$ und betrachte die Sprache L aller Wörter, die aus n 0en gefolgt von n 1er bestehen, wobei $n \geq 0$
- Wir können L konzise in Mengennotation angeben:

$$L = \{0^n 1^n \mid n \geq 0\}$$

- Es gilt $010101 \notin L$, aber $010011 \in L^2$
- Allgemein erhalten wir etwa:

$$L^2 = \{0^n 1^n 0^k 1^k \mid n, k \geq 0\}$$



Einführung in die Theoretische Informatik

Martin Avanzini Christian Dalvit Jamie Hochrainer

Georg Moser Johannes Niederhauser Jonas Schöpf

<https://tcs-informatik.uibk.ac.at>



Zusammenfassung

Zusammenfassung der letzten LVA

Definition (Alphabet)

Ein **Alphabet** Σ ist eine endliche, nicht leere Menge von Symbolen

Definition (Wort)

- Eine **Zeichenreihe** (ein **Wort**, ein **String**) ist eine endliche Folge von Symbolen über einem Alphabet Σ
- Die **leere Zeichenreihe** wird mit ϵ bezeichnet

Definition

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

Bemerkung

Die Algebra $\langle \Sigma^*; \cdot, \epsilon \rangle$ ist ein Monoid; das **Wortmonoid**

Einführung in die Logik

Syntax & Semantik der Aussagenlogik, Formales Beweisen, Konjunktive und Disjunktive Normalformen

Einführung in die Algebra

Algebraische Strukturen, Boolesche Algebra, Universelle Algebra

Einführung in die Theorie der Formalen Sprachen

Grammatiken und Formale Sprachen, Chomsky-Hierarchie, Reguläre Sprachen, Kontextfreie Sprachen, Anwendungen von formalen Sprachen

Einführung in die Berechenbarkeitstheorie und Komplexitätstheorie

Algorithmisch unlösbare Probleme, Turing Maschinen, Registermaschinen, Komplexitätstheorie

Einführung in die Programmverifikation

Prinzipien der Analyse von Programmen, Verifikation nach Hoare



Grammatiken und Formale Sprachen

Grammatiken und Formale Sprachen

Beispiel

$S \rightarrow \text{Pronomen Nomen Verb Adjektiv}$

$\text{Nomen} \rightarrow \text{Lehrveranstaltungsleiter}$

$\text{Nomen} \rightarrow \text{Vortragender}$

$\text{Pronomen} \rightarrow \text{Unser} \mid \text{Mein}$

$\text{Verb} \rightarrow \text{ist}$

$\text{Adjektiv} \rightarrow \text{lästig} \mid \text{nett} \mid \text{streng} \mid \text{monoton} \mid \text{anspruchsvoll}$

Es gilt

Unser Nomen

Verb ist

$S \xrightarrow{*} \text{Unser Lehrveranstaltungsleiter ist anspruchsvoll}$

Definition

Eine Grammatik G ist ein Quadrupel $G = (V, \Sigma, R, S)$, wobei

- 1 V eine endliche Menge von Variablen (oder Nichtterminale)
- 2 Σ ein Alphabet, die Terminale, $V \cap \Sigma = \emptyset$
- 3 R eine endliche Menge von Regeln
- 4 $S \in V$ das Startsymbol von G

Eine Regel ist ein Paar $P \rightarrow Q$ von Wörtern, sodass $P, Q \in (V \cup \Sigma)^*$ und in P mindestens eine Variable vorkommt

P nennen wir auch die Prämissen und Q die Konklusion der Regel

Konvention

- Variablen werden groß geschrieben, Terminale klein
- Statt $P \rightarrow Q_1, P \rightarrow Q_2, P \rightarrow Q_3$ schreiben wir $P \rightarrow Q_1 | Q_2 | Q_3$

Sei $G = (V, \Sigma, R, S)$ eine Grammatik und seien $x, y \in (V \cup \Sigma)^*$

Definition

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

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

- 2 In diesem Fall schreiben wir kurz $x \xrightarrow[G]{} y$

- 3 Wenn G aus dem Kontext folgt schreiben wir $x \xrightarrow{} y$

variable
↓

Definition (Ableitbar)

Wir sagen y ist aus x in G **ableitbar**, wenn $k \in \mathbb{N}$ und $w_0, w_1, \dots, w_k \in (V \cup \Sigma)^*$ gibt, sodass

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

Wir schreiben $x \xrightarrow[G]{}^* y$, beziehungsweise $x \xrightarrow{}^* y$

Sprache einer Grammatik

Definition

- Die vom Startsymbol S ableitbaren Wörter heißen **Satzformen**
- Elemente von Σ^* heißen **Terminalwörter**
- Satzformen, die Terminalwörter sind, heißen **Sätze**

Definition (Sprache einer Grammatik)

Die Menge aller Sätze

$$L(G) = \{x \in \Sigma^* \mid S \xrightarrow[G]{*} x\}$$

heißt die von der Grammatik G **erzeugte Sprache**

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

Klassen von Grammatiken

Definition (rechtslinear)

Grammatik $G = (V, \Sigma, R, S)$ heißt **rechtslinear**, wenn für alle Regeln $P \rightarrow Q$ gilt:

- 1 $P \in V$
- 2 $Q \in \Sigma^* \cup \Sigma^+V$

Beispiel

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

$$\begin{array}{l} B \Rightarrow 1 \\ B \Rightarrow 0B \end{array} \Rightarrow 0 \underset{1 \in L(G_1)}{\underset{\text{B}}{\underset{\text{B}}{\Rightarrow}}} 0 \underset{1 \in L(G_1)}{\underset{\text{B}}{\underset{\text{B}}{\Rightarrow}}} 00 \underset{1 \in L(G_1)}{\underset{\text{B}}{\underset{\text{B}}{\Rightarrow}}} 001 \in L(G_1) \quad B \rightarrow 0 \mid 1 \mid 0B \mid 1B$$

- Es gilt:

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

Definition (kontextfrei)

Grammatik $G = (V, \Sigma, R, S)$ heißt **kontextfrei**, wenn für alle Regeln $P \rightarrow Q$ gilt:

- 1 $P \in V$
- 2 $Q \in (V \cup \Sigma)^*$

Beispiel

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

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

- Es gilt:

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

Definition (kontextsensitiv)

Grammatik $G = (V, \Sigma, R, S)$ heißt **kontextsensitiv**, wenn für alle Regeln $P \rightarrow Q$ gilt:

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

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

- 2 oder $P = S$ und $Q = \epsilon$

Wenn $S \rightarrow \epsilon \in G$, dann kommt S nicht in einer Konklusion vor

Beispiel

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

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

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

Definition (beschränkt)

Grammatik $G = (V, \Sigma, R, S)$ heißt **beschränkt**, wenn für alle Regeln $P \rightarrow Q$ gilt:

- 1 entweder $|P| \leq |Q|$ oder
- 2 $P = S$ und $Q = \epsilon$

Wenn $S \rightarrow \epsilon \in G$, dann kommt S nicht in einer Konklusion vor

Beispiel

Die Grammatik $G_4 = (\{S, X, Y, T\}, \{a\}, R, S)$ sei wie folgt definiert:

$$\begin{array}{ll} S \rightarrow YT \mid a \mid aa & Xa \rightarrow aaX \\ Y \rightarrow XY \mid aa & XaT \rightarrow aaT \\ & XaaT \rightarrow aaaa \end{array}$$

$$L(G_4) = \{a^{2^n} \mid n \geq 0\}$$

Definition

Eine formale Sprache L heißt

- regulär (vom Typ 3)
wenn \exists rechtslineare Grammatik G , $L = L(G)$
- kontextfrei (vom Typ 2)
wenn \exists kontextfreie Grammatik G , $L = L(G)$
- kontextsensitiv (vom Typ 1)
wenn \exists kontextsensitive Grammatik G , $L = L(G)$

Bemerkung

- formale Sprache $L \subseteq \Sigma^*$
- Grammatik ist endliche Beschreibung von L
- Art der Beschreibung bestimmt Typ der Sprache

Definition

Eine formale Sprache L heißt

- **beschränkt** wenn \exists beschränkte Grammatik G , $L = L(G)$
- **rekursiv aufzählbar** (vom **Typ 0**)
wenn \exists Grammatik G , $L = L(G)$

Satz (Chomsky-Hierarchie)

Es gelten die folgenden Inklusionen

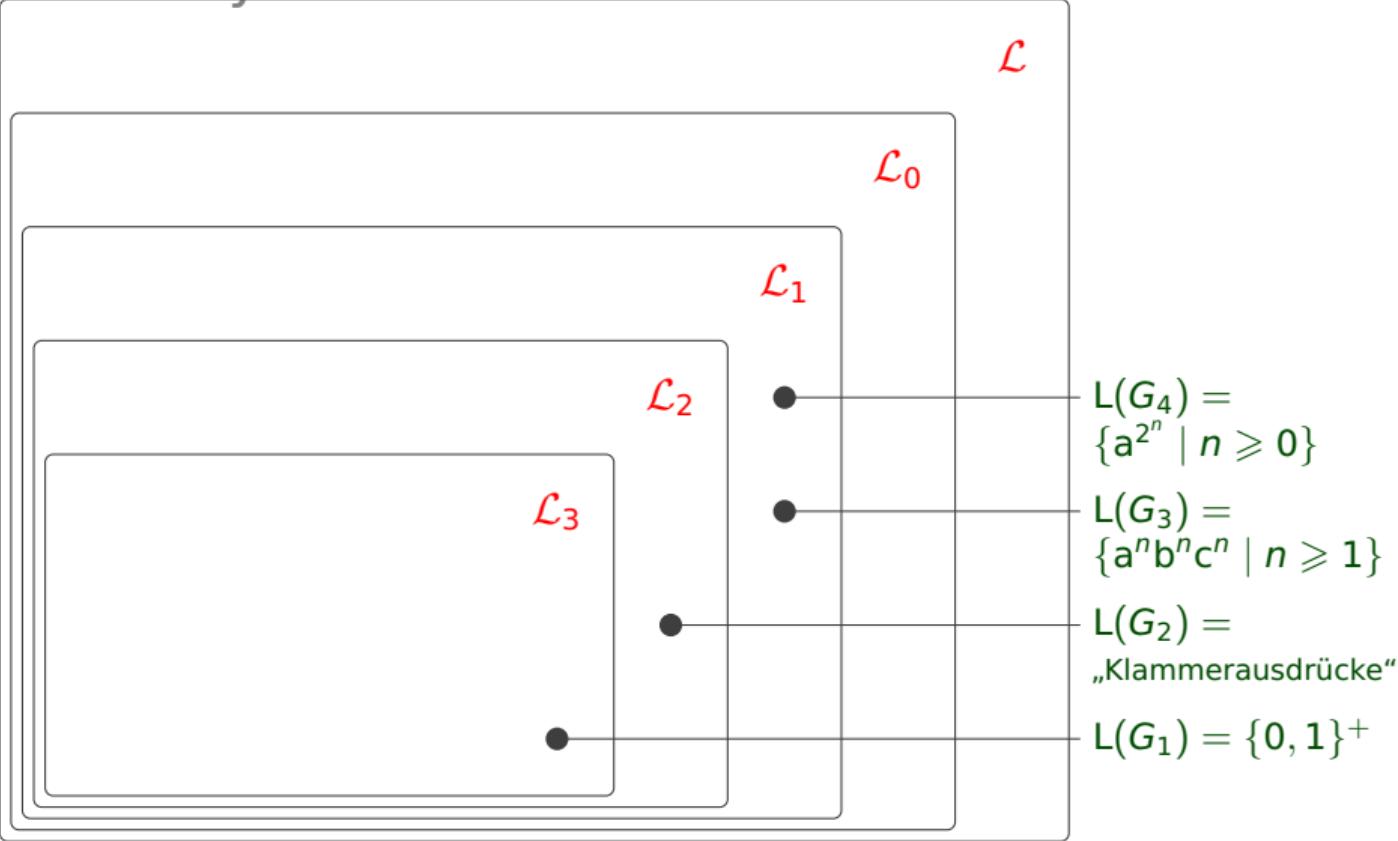
$$\mathcal{L}_3 \subsetneq \mathcal{L}_2 \subsetneq \mathcal{L}_1 \subsetneq \mathcal{L}_0 \subsetneq \mathcal{L}$$

- \mathcal{L}_i die Klasse der Sprachen von Typ i
- \mathcal{L} Klasse der formalen Sprachen

Satz

Eine Sprache L ist kontextsensitiv gdw. L beschränkt ist

Chomsky-Hierarchie



Feedback zum Typ einer Formalen Sprache

Wie sollte man denn die Sprachen beschreiben? In natürlicher Sprache ist es unexakt. Und wenn man auf eine Menge der Sprache kommt: Wie kann man dann sicher sein, dass es sich bei der erzeugten Sprache tatsächlich um diese Menge handelt? Müsste man das beweisen? Wenn ja: Wie?

Zudem ist es mir schwer gefallen, „leichtere“ Grammatiken für gegebene Sprachen zu finden. Auch hier: Wie kann man sicher sein, dass eine gegebene Grammatik tatsächlich die gesamte Sprache erzeugt? Und wie kann man sicher sein, dass die gefundene Grammatik tatsächlich die „minimalste“ ist?



Einführung in die Theoretische Informatik

Martin Avanzini Christian Dalvit Jamie Hochrainer

Georg Moser Johannes Niederhauser Jonas Schöpf

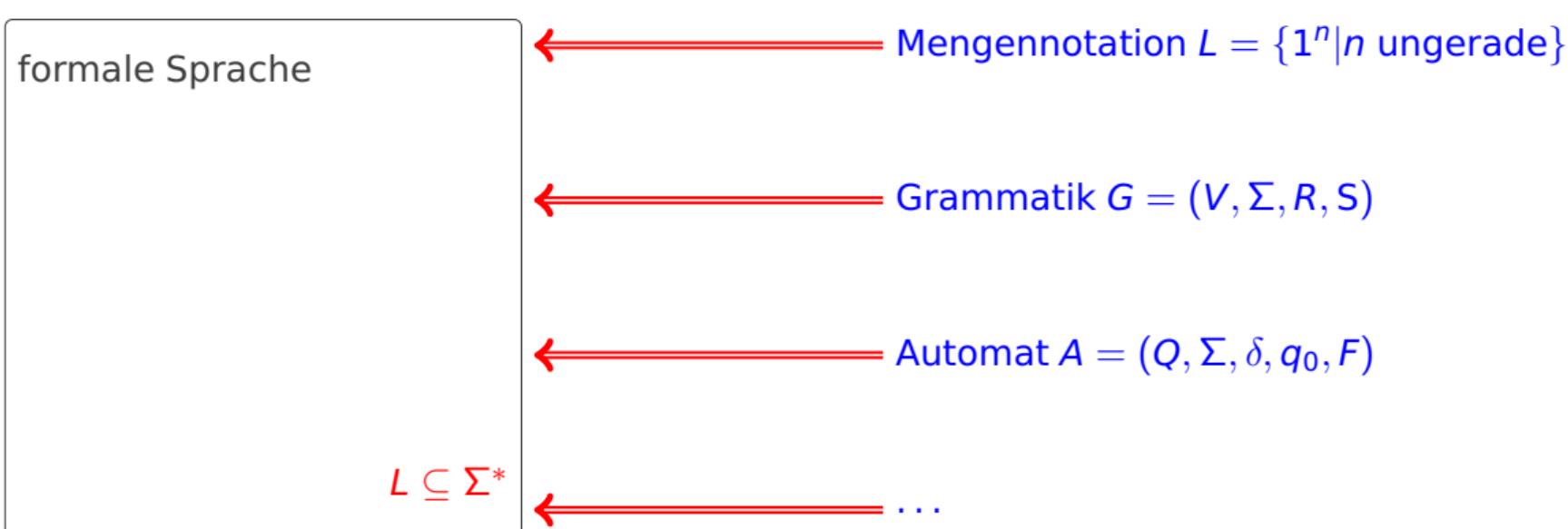
<https://tcs-informatik.uibk.ac.at>



Zusammenfassung

Zusammenfassung der letzten LVA

Formale Sprachen und Beschreibungsmöglichkeiten



Einführung in die Logik

Syntax & Semantik der Aussagenlogik, Formales Beweisen, Konjunktive und Disjunktive Normalformen

Einführung in die Algebra

Algebraische Strukturen, Boolesche Algebra, Universelle Algebra

Einführung in die Theorie der Formalen Sprachen

Grammatiken und Formale Sprachen, Chomsky-Hierarchie, **Reguläre Sprachen**,
Kontextfreie Sprachen, Anwendungen von formalen Sprachen

Einführung in die Berechenbarkeitstheorie und Komplexitätstheorie

Algorithmisch unlösbare Probleme, Turing Maschinen, Registermaschinen,
Komplexitätstheorie

Einführung in die Programmverifikation

Prinzipien der Analyse von Programmen, Verifikation nach Hoare



Reguläre Sprachen

Endliche Automaten



Anwendungen von Endlichen Automaten

- Software zum Entwurf und Testen von digitalen Schaltkreisen (Mealy/Moore Automaten)
- Softwarebausteine eines Compilers, etwa in der lexikalischen Analyse:
 - 1 lexikalische Scanner („Lexer“) wird mit endlichen Automaten implementiert
 - 2 Der lexikalische Scanner dient zur Aufteilung des Eingabetextes in logische Einheiten, wie Bezeichner oder Schlüsselwörter
- Software zum Durchsuchen umfangreicher Texte
- Software zur Verifizierung aller Arten von Systemen, die eine endliche Anzahl verschiedener Zustände besitzen
- Softwarebausteine eines Computerspiels:
 - 1 Kontrolle von Spielfiguren kann mit Hilfe eines endlichen Automaten implementiert werden
 - 2 erlaubt eine bessere Modularisierung des Codes

Beispiel

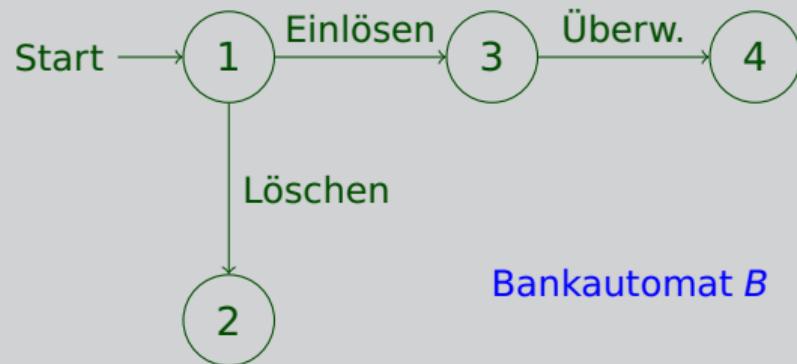
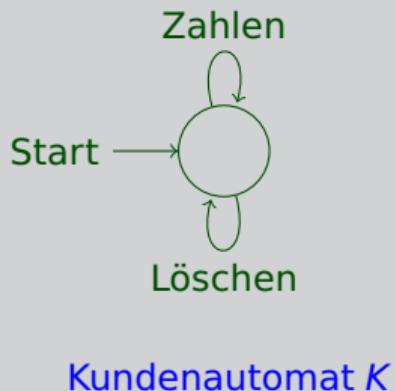
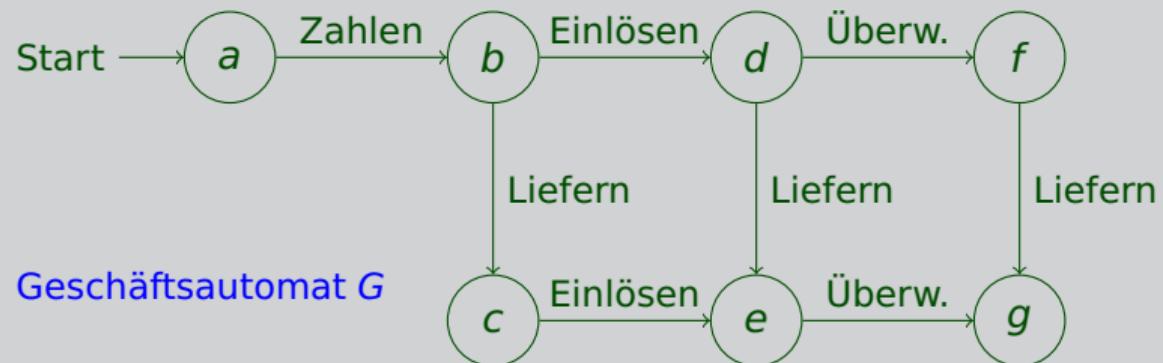
Wir untersuchen Protokolle, die den Gebrauch elektronischen „Geldes“ ermöglichen; dabei handeln: der **Kunde**, die **Bank** und das **Geschäft**:

- Der Kunde kann **zahlen**
- Der Kunde kann das Geld **löschen**
- Das Geschäft kann dem Kunden Waren **zusenden**
- Das Geschäft kann Geld **einlösen**
- Die Bank kann Geld **überweisen**

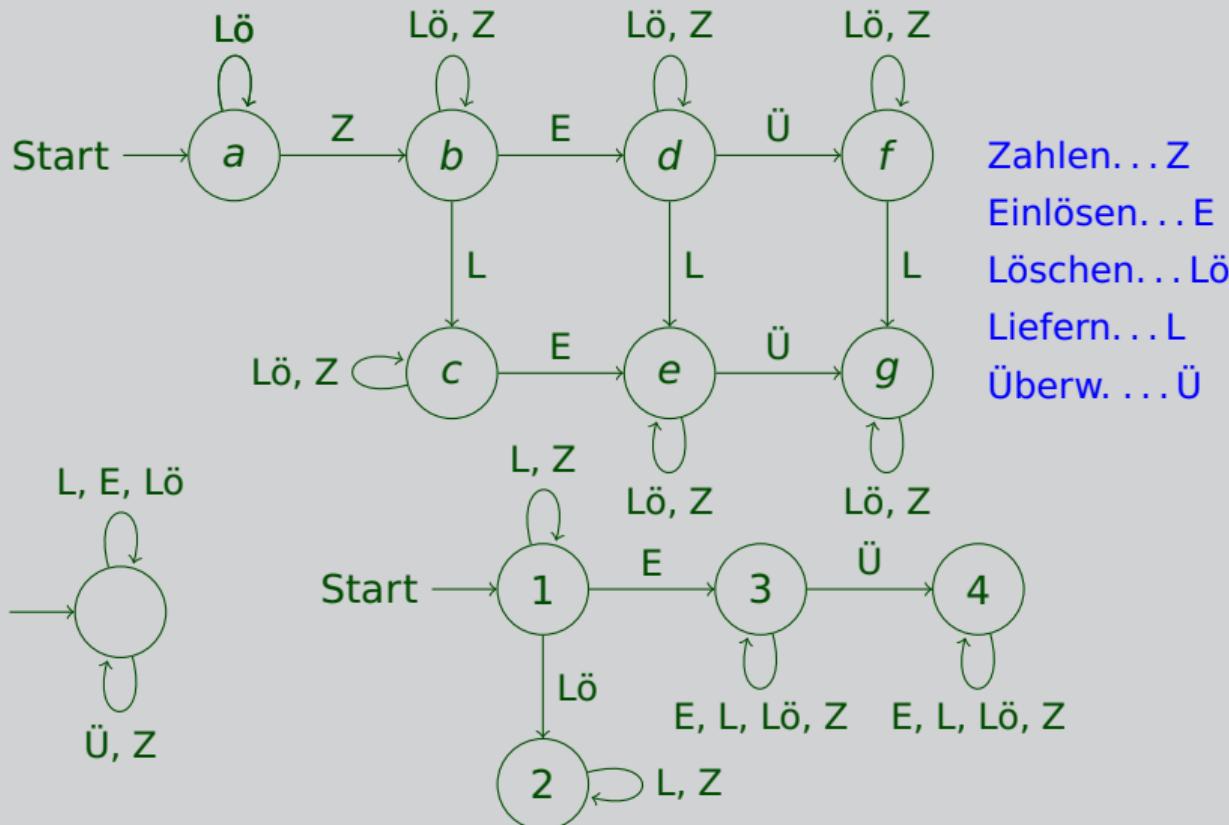
Wir treffen die folgenden Grundannahmen:

- Der Kunde ist **unverantwortlich**
- Das Geschäft ist **verantwortlich**, aber **gutgläubig**
- Die Bank ist **strikt**

Beispiel (Fortsetzung)



Beispiel (Fortsetzung)



Beispiel (Fortsetzung)

Wir definieren den Produktautomaten $B \times G$ aus B und G :

- 1 Die Zustände dieses Automaten sind:

$$(i, x) \quad \text{wobei } i \in \{1, 2, 3, 4\}, x \in \{a, b, c, d, e, f, g\}$$

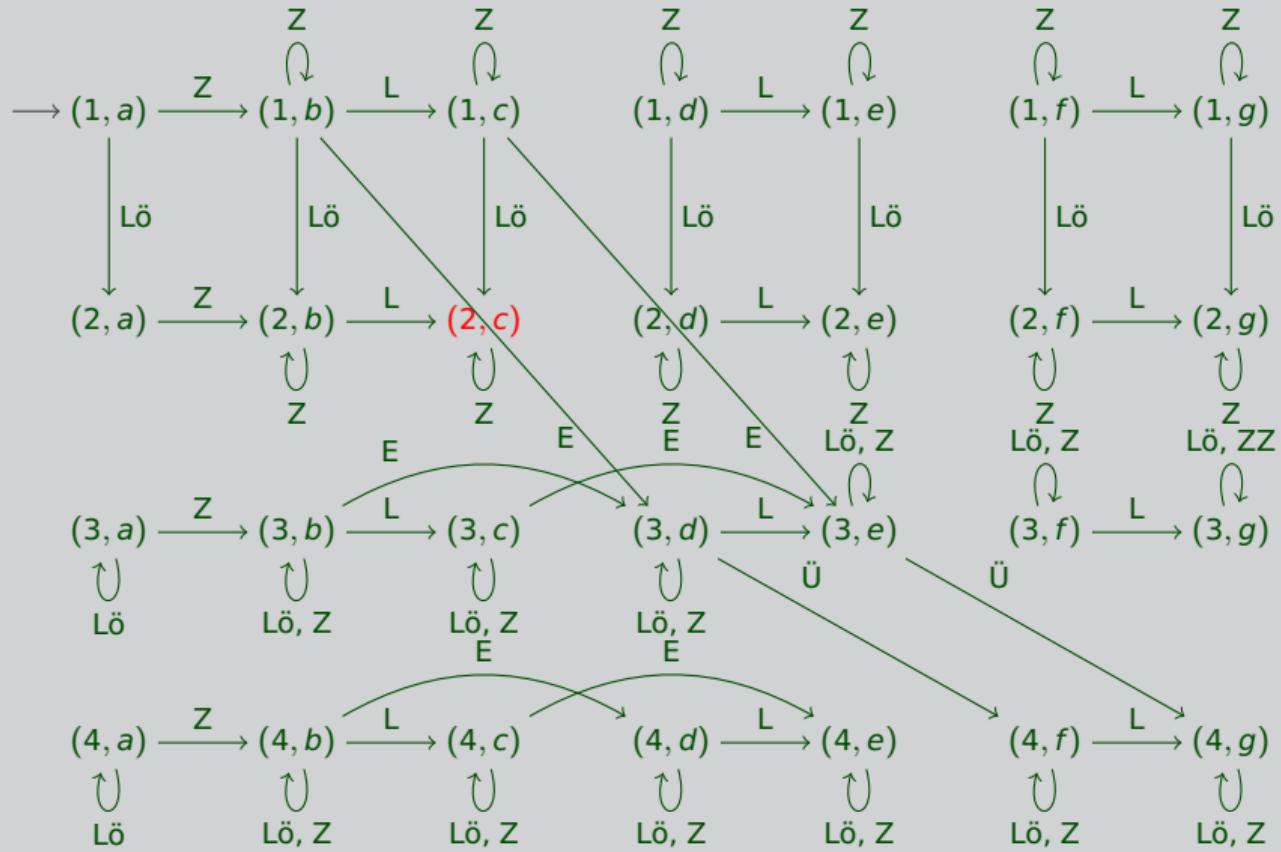
- 2 Die Übergänge werden durch paralleles Ausführen von B und G definiert:

$$\text{wenn } i \xrightarrow{\text{Aktion}} i' \text{ und } x \xrightarrow{\text{Aktion}} x' \text{ dann } (i, x) \xrightarrow{\text{Aktion}} (i', x')$$

Betrachte B und G :

- in B gilt, dass aus Zustand 1 durch die Aktion „Einlösen“ Zustand 3 wird, konzise: $1 \xrightarrow{\text{Einlösen}} 3$
- in G gilt, dass aus Zustand b mit Aktion „Einlösen“ d wird Konzise: $b \xrightarrow{\text{Einlösen}} d$
- also gilt in $B \times G$, dass aus Zustand $(1, b)$ mit Aktion „Einlösen“ Zustand $(3, d)$ wird, konzise: $(1, b) \xrightarrow{\text{Einlösen}} (3, d)$

Beispiel (Fortsetzung)



Beispiel (Fortsetzung)

Als Schlussfolgerung ergibt sich, dass das Protokoll **nicht sicher** ist:

Der Automat $B \times G$ kann in den Zustand $(2, c)$ gelangen, in welchem die Waren geschickt wurden und trotzdem nie eine Überweisung an das Geschäft erfolgen wird

Definition (Deterministischer endlicher Automat (kurz: DEA))

Ein **DEA** ist ein 5-Tupel $A = (Q, \Sigma, \delta, q_0, F)$ sodass

- 1 Q eine endliche Menge von **Zuständen**
- 2 Σ eine endliche Menge von **Eingabesymbole**, (Σ wird auch **Eingabealphabet** genannt)
- 3 $\delta: Q \times \Sigma \rightarrow Q$ die **Übergangsfunktion**
- 4 $q_0 \in Q$ der **Startzustand**
- 5 $F \subseteq Q$ eine endliche Menge von **akzeptierenden Zuständen**

δ muss für alle möglichen Argumente definiert sein; vgl. auch **Mealy-Moore Automaten**

Zustandstabelle

	$a_1 \in \Sigma$	$a_2 \in \Sigma$	\dots
$q_1 \in Q$	$\delta(q_1, a_1)$	$\delta(q_1, a_2)$	\dots
$q_2 \in Q$	$\delta(q_2, a_1)$		
\vdots	\vdots		

Zustandsgraph

Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein DEA, der **Zustandsgraph** ist definiert, sodass

- 1 die Ecken die Zustände sind,
- 2 für Zustände $p, q \in Q$ sind die Kanten von p nach q alle Tripel

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

Konvention

Zu jeder Kante (p, a, q) schreibt man die Eingabe a . Den Startzustand markiert man mit einem Pfeil; die akzeptierenden Zustände werden mit einem doppelten Kreis gekennzeichnet.

Definition (erweiterte Übergangsfunktion)

Sei δ eine Übergangsfunktion, wir definieren die **erweiterte Übergangsfunktion**
 $\widehat{\delta}: Q \times \Sigma^* \rightarrow Q$ induktiv:

$$\widehat{\delta}(q, \epsilon) := q$$

$$\widehat{\delta}(q, xa) := \delta(\widehat{\delta}(q, x), a) \quad x \in \Sigma^*, a \in \Sigma$$

Definition

Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein DEA; die **Sprache $L(A)$** von A :

$$L(A) := \{x \in \Sigma^* \mid \widehat{\delta}(q_0, x) \in F\}$$

Satz

Sei A ein DEA, dann ist $L(A)$ regulär und umgekehrt existiert zu jeder regulären Sprache L ein DEA A , sodass $L = L(A)$

Wiederholung: Vollständige Induktion

Vollständige Induktion

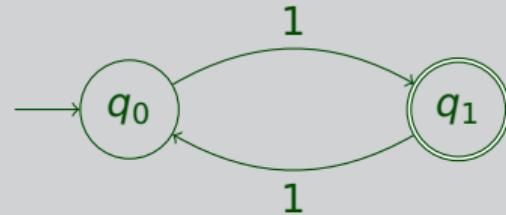
- 1 Sei m eine fest gewählte natürliche Zahl, z.B. $m = 0$ oder $m = 1$
- 2 Eine Aussage $S(n)$ soll für alle natürlichen Zahlen $n \geq m$ gezeigt werden
- 3 In diesem Fall gehen wir wie folgt vor:
 - **Induktionsbasis:** Wir zeigen, dass S für den Basiswert m gilt.
 - **Induktionsschritt:** Wir zeigen, dass für alle $n \geq m$ aus $S(n)$ auch $S(n + 1)$ folgt.
- 4 Dann gilt $S(n)$ für alle $n \geq m$

Formal & Eingeschränkt

$$(S(m) \wedge \forall n \geq m (S(n) \rightarrow S(n + 1))) \rightarrow (\forall n \geq m S(n))$$

Beispiel

Betrachte den Automaten B :



Satz

Die von Automat B akzeptierte Sprache ist gegeben durch $L(B) = \{1^n | n \text{ ungerade}\}$

Beweis.

Beweis mittels verschränkter Induktion der folgenden beiden Aussagen:

$S_1(n)$ Der Automat A ist nach n -maligem Lesen von 1 im Zustand $q_0 \Leftrightarrow n$ gerade.

$S_2(n)$ Der Automat A ist nach n -maligem Lesen von 1 im Zustand $q_1 \Leftrightarrow n$ ungerade.

Basis.

- Nach 0-maligem Lesen ist A im Zustand q_0 gdw 0 gerade. ✓
- Nach 0-maligem Lesen ist A im Zustand q_1 gdw 0 ungerade. ✓

Induktionshypothese. Wir können die Aussagen $S_1(n)$ und $S_2(n)$ annehmen

- Nach n -maligem Lesen ist A im Zustand q_0 gdw n gerade.
- Nach n -maligem Lesen ist A im Zustand q_1 gdw n ungerade.

Induktionsschritt. Nun zeigen wir $S_1(n+1)$ und $S_2(n+1)$ (wobei wir $S_1(n)$ und $S_2(n)$ annehmen)

- $S_1(n+1)$: Nach $n+1$ -maligem Lesen ist A im Zustand q_0 gdw $n+1$ gerade.
 A ist nach $n+1$ -maligem Lesen von 1 im Zustand $q_0 \Leftrightarrow$
 A ist nach n -maligem Lesen von 1 im Zustand $q_1 \Leftrightarrow n$ ungerade \Leftrightarrow
 $n+1$ gerade ✓
- $S_2(n+1)$: Nach $n+1$ -maligem Lesen ist A im Zustand $q_1 \Leftrightarrow n+1$ ungerade. ✓



Kontextfreie Sprachen

Beispiel

Induktive Definition von Palindromen über $\Sigma := \{0, 1\}$:

- 1 $\epsilon, 0, 1$ sind Palindrome
- 2 Wenn x ein Palindrom ist, dann sind auch

$$0x0 \quad 1x1$$

Palindrome

Wir betrachten die folgenden Regeln R :

$$P \rightarrow \epsilon \mid 0 \mid 1$$

$$P \rightarrow 0P0 \mid 1P1$$

Die KFG $G_1 = (\{P\}, \Sigma, R, P)$ beschreibt die Sprache der Palindrome:

$L(G_1)$ ist genau die Menge der Palindrome über dem Alphabet Σ

Lemma (für KFG)

Gelte $A \Rightarrow X_1X_2\dots X_n \stackrel{*}{\Rightarrow} x$, wobei $X_i \in V \cup \Sigma$, $x \in \Sigma^*$; dann können wir x in die Stücke x_1, x_2, \dots, x_n brechen, sodass für alle i : $X_i \stackrel{*}{\Rightarrow} x_i$

Hilfsüberlegung

- Wir betrachten: $X_1X_2\dots X_n \stackrel{*}{\Rightarrow} x$
- Sei $i < j$, dann sind in x alle aus X_i abgeleiteten Satzformen links von den aus X_j abgeleiteten zu finden



Beweis (des Lemmas).

- 1 Wenn $X_i \in \Sigma$, dann $X_i = x_i$ und offensichtlich $X_i \stackrel{*}{\Rightarrow} x_i$
- 2 Wenn $X_i \in V$, dann erhalten wir $X_i \stackrel{*}{\Rightarrow} x_i$ indem
 - Expansionen von X_1, \dots, X_{i-1} eliminiert
 - Expansionen von X_{i+1}, \dots, X_n eliminiert
 - überflüssige Schritte weggelassen werden



Lemma

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

1 sei $A \xrightarrow{*} x$ eine Ableitung in G

2 seien $u, v \in (V \cup \Sigma)^*$

Dann ist auch $uAv \xrightarrow{*} uxv$ eine Ableitung in G

Beweis.

Angenommen es existieren Wörter $w_1, \dots, w_{k-1} \in (V \cup \Sigma)^*$, sodass

$$A \Rightarrow w_1 \Rightarrow \dots \Rightarrow w_{k-1} \Rightarrow x$$

Wir argumentieren informell:

- Wir betrachten den Schritt $w_i \Rightarrow w_{i+1}$ für $i \in \{1, \dots, k-2\}$
- Nach Definition gilt auch $uw_iv \Rightarrow uw_{i+1}v$
- Die anderen Fälle werden gleich behandelt



Korrektheit der Grammatik.

Es ist zu zeigen, dass $x \in L(G)$ gdw. x ein Palindrom ist

Wir zeigen nur: $x \in L(G)$ impliziert x ist ein Palindrom. Dazu verwenden wir Induktion nach der Länge ℓ der Ableitung $P \xrightarrow{*} x$

1 Basis $\ell = 1$: Also gilt einer der folgenden 3 Fälle:

- $x = \epsilon$
- $x = 0$
- $x = 1$

In allen Fällen: x ist Palindrom

2 Schritt $\ell > 1$: Also hat die Ableitung eine der folgenden Gestalten:

- $P \Rightarrow 0P0 \xrightarrow{*} 0y0 = x$
- $P \Rightarrow 1P1 \xrightarrow{*} 1y1 = x$

wobei $y \in \Sigma^*$

Somit gilt $P \xrightarrow{*} y$, woraus mit Induktionshypothese folgt, y ist ein Palindrom; damit ist aber auch x ein Palindrom



Einführung in die Theoretische Informatik

Martin Avanzini Christian Dalvit Jamie Hochrainer

Georg Moser Johannes Niederhauser Jonas Schöpf

<https://tcs-informatik.uibk.ac.at>



Zusammenfassung

Definition (Deterministischer endlicher Automat (kurz: DEA))

Ein **DEA** ist ein 5-Tupel $A = (Q, \Sigma, \delta, q_0, F)$ sodass

- 1 Q eine endliche Menge von **Zuständen**
- 2 Σ eine endliche Menge von **Eingabesymbole**
- 3 $\delta: Q \times \Sigma \rightarrow Q$ die **Übergangsfunktion**
- 4 $q_0 \in Q$ der **Startzustand**
- 5 $F \subseteq Q$ eine endliche Menge von **akzeptierenden Zuständen**

Zu beachten: δ muss für alle möglichen Argumente definiert sein

Satz

Sei A ein **DEA**, dann ist $L(A)$ regulär und umgekehrt existiert zu jeder regulären Sprache L ein **DEA** A , sodass $L = L(A)$

Einführung in die Logik

Syntax & Semantik der Aussagenlogik, Formales Beweisen, Konjunktive und Disjunktive Normalformen

Einführung in die Algebra

Algebraische Strukturen, Boolesche Algebra, Universelle Algebra

Einführung in die Theorie der Formalen Sprachen

Grammatiken und Formale Sprachen, Chomsky-Hierarchie, Reguläre Sprachen,
Kontextfreie Sprachen, Anwendungen von formalen Sprachen

Einführung in die Berechenbarkeitstheorie und Komplexitätstheorie

Algorithmisch unlösbare Probleme, Turing Maschinen, Registermaschinen,
Komplexitätstheorie

Einführung in die Programmverifikation

Prinzipien der Analyse von Programmen, Verifikation nach Hoare



Linksableitung, Rechtsableitung und Rekursive Inferenz

Definition

- Eine **Linksableitung** ist eine Ableitung sodass immer die am weitesten links stehende Variable ersetzt wird
- In einer **Rechtsableitung** wird immer die am weitesten rechts stehende Variable ersetzt

$\xrightarrow{\ell}, \xrightarrow{\ell}^*$

$\xrightarrow{r}, \xrightarrow{r}^*$

Beispiel

Wir betrachten KFG $G = (\{S\}, \{(,)\}, R, S)$, wobei R :

$$S \rightarrow \epsilon \mid (S) \mid SS$$

dann gilt $S \xrightarrow{r} SS \xrightarrow{r} S(S) \xrightarrow{r} S() \xrightarrow{r} (S)() \xrightarrow{r} ()()$

Definition (Eindeutigkeit einer Grammatik)

KFG G heißt **eindeutig**, wenn jedes Wort $x \in L(G)$ genau eine **Linksableitung** besitzt, ansonsten **mehrdeutig**

Definition

Sei G eine KFG und sei $A \rightarrow x$ eine Regel in G , sodass $x \in (V \cup \Sigma)^*$. Die **Sprache von A** , bezeichnet als $L(A)$, kann mittels **rekursiver Inferenz** definiert werden:

1 Wenn $x \in \Sigma^*$, dann $x \in L(A)$

2 Andererseits, sei $x = X_1 \cdots X_n$, wobei $X_i \in V \cup \Sigma$

Gelte außerdem $x_i \in L(X_i)$ oder $x_i = X_i \in \Sigma$

Dann gilt $x_1 x_2 \cdots x_n \in L(A)$

Bemerkung

- Die rekursive Inferenz entspricht dem **bottom-up parsing** eines Compilers: zuerst wird der Rumpf einer Regel gelesen, dann wird das Ergebnis der Variable im Kopf übergeben
- Der Parsergenerator yacc (siehe später) generiert einen bottom-up parser

Beispiel

Wir betrachten die KFG $G_1 = (\{E, I\}, \{+, \cdot, (,), a, b, 0, 1\}, R, E)$, wobei R wie folgt:

$$\begin{array}{llll} E \rightarrow I & E \rightarrow E \cdot E & I \rightarrow a & I \rightarrow I0 \\ E \rightarrow E+E & E \rightarrow (E) & I \rightarrow Ib & I \rightarrow I1 \\ & & I \rightarrow Ia & I \rightarrow b \end{array}$$

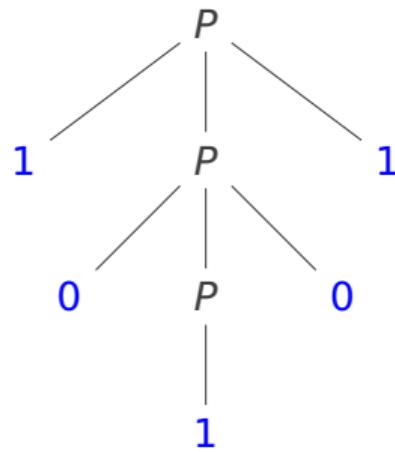
und zeigen $(a+b10) \in L(E)$:

	Wort x	Variable	Regel	Rekursion
1	a	I	$I \rightarrow a$	
2	b	I	$I \rightarrow b$	
3	b1	I	$I \rightarrow I1$	2
4	b10	I	$I \rightarrow I0$	3
5	a	E	$E \rightarrow I$	1
6	b10	E	$E \rightarrow I$	4
7	a+b10	E	$E \rightarrow E+E$	5, 6
8	(a+b10)	E	$E \rightarrow (E)$	7

Wiederholung: Bäume



Das ist ein Baum



Und so sehen Bäume jetzt aus

Definition (Syntaxbaum)

Ein **Syntaxbaum** für KFG $G = (V, \Sigma, R, S)$ ist ein Baum B mit:

- 1 Jeder innere Knoten von B ist eine Variable in V
- 2 Jedes Blatt in B ist entweder:
 - ein Terminal aus Σ
 - ein Nichtterminal aus V , oder
 - ϵ

Im letzten Fall ist das Blatt das einzige Kind seines Vorgängers

- 3 Sei A ein innerer Knoten, X_1, \dots, X_n seine Kinder ($X_i \in V \cup \Sigma$); dann gilt:

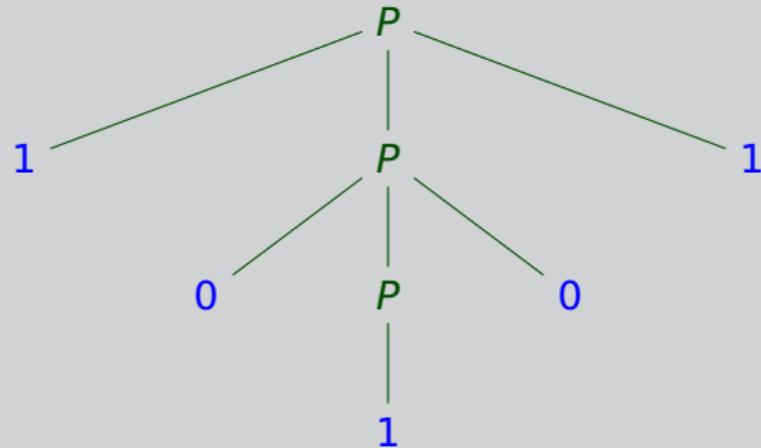
$$A \rightarrow X_1 \cdots X_n \in R$$

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

Beispiel

Wir betrachten die KFG $G = (\{P\}, \Sigma, R, P)$ mit den Regeln $R: P \rightarrow \epsilon \mid 0 \mid 1 \mid 0P0 \mid 1P1$

Dann ist der folgende Baum ein Syntaxbaum für G :

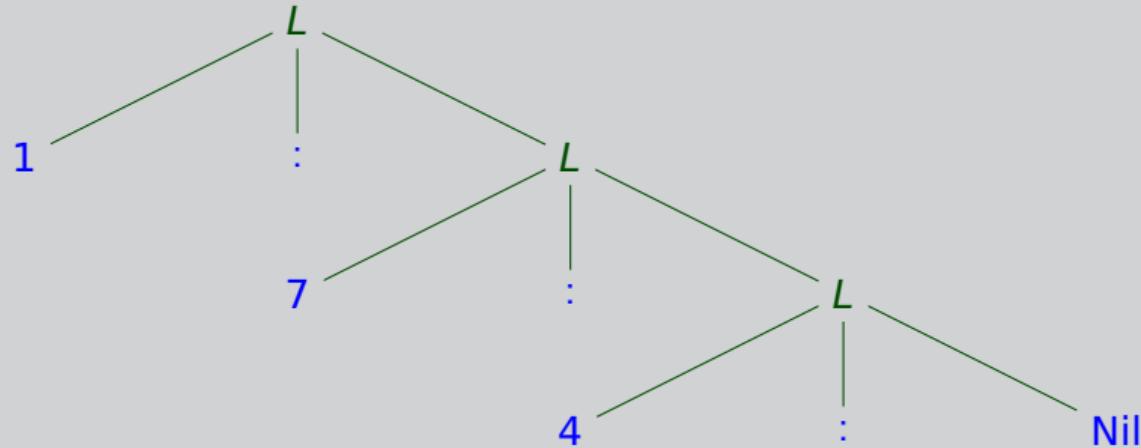


Bemerkung

Vergleiche AST zum Parsen von Haskellausdrücken in Funktionaler Programmierung

Beispiel

Sei $G = (\{L\}, \{0, 1, \dots, 9, \text{Nil}, :\} R, P)$ mit den Regeln $R: L \rightarrow \text{Nil} \mid 0 : L \mid 1 : L \mid \dots \mid 9 : L$



Das entspricht der Liste $1 : 7 : 4 : \text{Nil}$

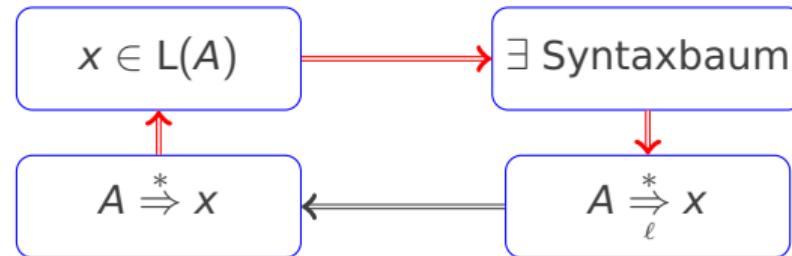
Bemerkung

Eigenschaften von Syntaxbäumen können mit Induktion über die **Höhe des Baumes** gezeigt werden.

Satz

Sei Σ ein Alphabet und sei $x \in \Sigma^*$. Die folgenden Beschreibungen kontextfreier Sprachen sind äquivalent:

- 1 $x \in L(A)$ nach dem rekursiven Inferenzverfahren
- 2 $A \xrightarrow{*} x$
- 3 $A \xrightarrow{\ell}^* x$
- 4 $A \xrightarrow{r}^* x$
- 5 Es existiert ein Syntaxbaum mit Wurzel A und Ergebnis x



Satz

Angenommen $x \in L(A)$ nach dem rekursiven Inferenzverfahren, dann gibt es einen Syntaxbaum B mit Wurzel A und Ergebnis x

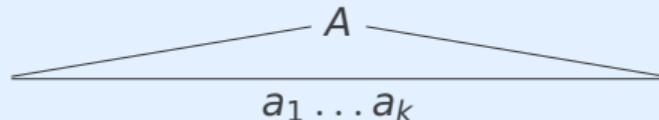
Beweis.

Mit Induktion nach der Anzahl der Rekursionen im Inferenzverfahren

- **Basis** $x \in L(A)$, $x = a_1 \dots a_k$ benutzt genau die Regel

$$A \rightarrow a_1 \dots a_k$$

Wir konstruieren einen Syntaxbaum B mit Wurzel A wie folgt



Beweis (Fortsetzung).

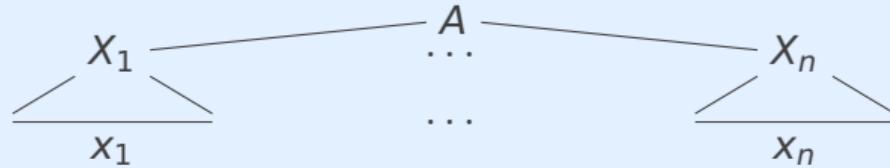
- **Schritt** $x \in L(A)$, $x = x_1 \dots x_n$ benutzt die Regel

$$A \rightarrow X_1 \cdots X_n$$

wobei $X_i \in V \cup \Sigma$ und $\forall i x_i \in L(X_i)$

Nach IH \exists Syntaxbäume B_i gelten mit Wurzeln X_i und Ergebnis x_i

Wir konstruieren einen neuen Syntaxbaum mit Wurzel A :



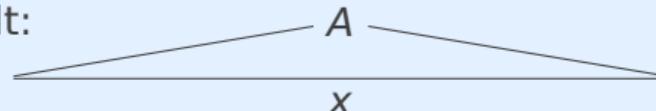
Satz

Sei B ein Syntaxbaum mit Wurzel A und Ergebnis $x \in \Sigma^*$, dann gibt es eine Linksableitung (eine Rechtsableitung) von x aus A

Beweis.

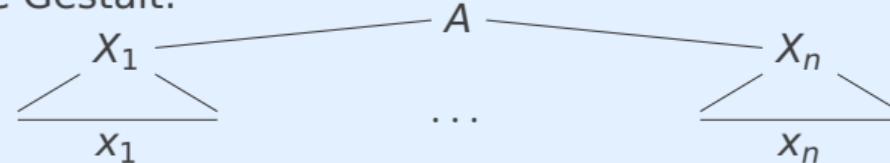
Mit Induktion nach der Höhe des Syntaxbaums B

- **Basis** Habe B die Gestalt:



dann existiert $A \rightarrow x \in R$, also $A \xrightarrow{\ell} x$

- **Schritt** Habe B die Gestalt:



dann $\exists A \rightarrow X_1 \cdots X_n \in R$, also:

$$A \xrightarrow{\ell} X_1 \cdots X_n \xrightarrow{\ell}^* x_1 x_2 \cdots x_n \xrightarrow{\ell}^* x_1 \cdots x_{n-1} x_n$$

Satz

Angenommen $A \xrightarrow{*} x$ mit $x \in \Sigma^*$, dann liefert das rekursive Inferenzverfahren, dass $x \in L(A)$

Beweis.

Mit Induktion nach der Länge ℓ der Ableitung $A \xrightarrow{*} x$:

- **Basis** Sei $\ell = 1$, dann gilt $x \in L(A)$
- **Schritt** Angenommen $\ell = \ell' + 1$

Zunächst können wir die Ableitung $A \xrightarrow{*} x$ wie folgt schreiben:

$$A \Rightarrow X_1 X_2 \cdots X_n \xrightarrow{*} x_1 x_2 \cdots x_n = x$$

Wir verwenden, dass wir die Ableitungen der Sätze x_i aufbrechen können, also gilt:

- Wenn $X_i \in \Sigma$, dann $X_i = x_i$
- Wenn $X_i \in V$, dann gilt $X_i \xrightarrow{*} x_i$ und somit $x_i \in L(X_i)$

Anwendung der Theorie der Kontextfreien Sprachen

- Parsergeneratoren, beziehungsweise im **Compilerbau**
 - 1 Parsergenerator verwandelt die Beschreibung einer Sprache in einen Parser für diese Sprache
 - 2 Parser werden zur syntaktischen Analyse von Programmen verwendet
- Anwendungen im Bereich der **Wissensrepräsentation**, etwa in XML-Dokumenten (siehe Skriptum)

Lexikalische Analyse

- der Eingabestrom aus ASCII-Zeichen wird analysiert
- in terminale Symbole der formalen Sprache zerlegt, die man **Token** nennt

Beispiel

Betrachte die Eingabe eines (simplen) Taschenrechners, dann sind die Token die Ziffern der Rechung, charakterisiert etwa durch den folgenden regulären Ausdruck

([0-9]+|([0-9]*\.[0-9]+))([eE][+-]#[0-9]+)?

mögliche Eingabe: $1.0 + (2.1e1 - 3 * 5) * 0.5$

die Regeln der lexikalischen Analyse werden in der folgenden Form definiert

Muster	Aktion
--------	--------

Beispiel

KFG G_2 für arithmetische Ausdrücke

$$E \rightarrow T \mid +T \mid -T \mid E+T \mid E-T$$

$$T \rightarrow F \mid T \cdot F \mid T/F$$

$$F \rightarrow c \mid v \mid (E) .$$

in G_2 gilt etwa $-c * v \in L(E)$:

	Wort x	Variable	Regel	Rekursion
1	c	F	$F \rightarrow c$	
2	v	F	$F \rightarrow v$	
3	c	T	$T \rightarrow F$	1
4	c · v	T	$T \rightarrow T \cdot F$	3, 2
5	-c · v	E	$E \rightarrow -T$	4

Parsergenerierung in C mit yacc

Regelteil calc.y (gekürzt)

```
expression:    term          { $$ = $1; }
              | '+' term     { $$ = $2; }
              | '-' term     { $$ = -$2; }
              | expression '+' term { $$ = $1 + $3; }
              | expression '-' term { $$ = $1 - $3; }
              ;
;

term:         factor        { $$ = $1; }
             | term '*' factor { $$ = $1 * $3; }
             | term '/' factor { $$ = $1 / $3; }
             ;
;

factor:       NUMBER        { $$ = $1; }
            | '(' expression ')' { $$ = $2; }
            ;
;
```



Demo: Lex & Yacc



Einführung in die Theoretische Informatik

Martin Avanzini Christian Dalvit Jamie Hochrainer

Georg Moser Johannes Niederhauser Jonas Schöpf

<https://tcs-informatik.uibk.ac.at>

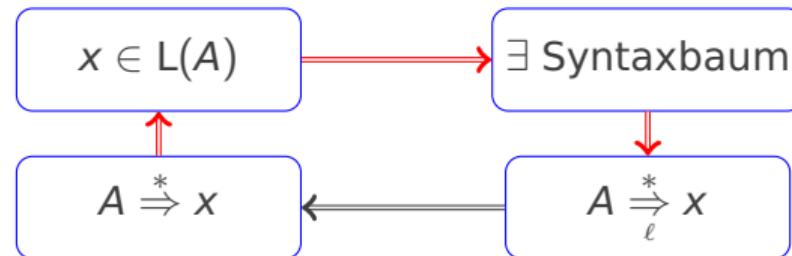


Zusammenfassung

Satz

Sei Σ ein Alphabet und sei $x \in \Sigma^*$. Die folgenden Beschreibungen kontextfreier Sprachen sind äquivalent:

- 1 $x \in L(A)$ nach dem rekursiven Inferenzverfahren
- 2 $A \xrightarrow{*} x$
- 3 $A \xrightarrow{\ell}^* x$
- 4 $A \xrightarrow{r}^* x$
- 5 Es existiert ein Syntaxbaum mit Wurzel A und Ergebnis x



Einführung in die Logik

Syntax & Semantik der Aussagenlogik, Formales Beweisen, Konjunktive und Disjunktive Normalformen

Einführung in die Algebra

Algebraische Strukturen, Boolesche Algebra, Universelle Algebra

Einführung in die Theorie der Formalen Sprachen

Grammatiken und Formale Sprachen, Chomsky-Hierarchie, Reguläre Sprachen, Kontextfreie Sprachen, Anwendungen von formalen Sprachen

Einführung in die Berechenbarkeitstheorie und Komplexitätstheorie

Algorithmisch unlösbare Probleme, Turing Maschinen, Registermaschinen, Komplexitätstheorie

Einführung in die Programmverifikation

Prinzipien der Analyse von Programmen, Verifikation nach Hoare



Berechenbarkeitstheorie

Einführung in die Berechenbarkeitstheorie

Frage

Ist jedes Problem **algorithmisch** lösbar?

Antwort

Nein

Ein einfaches Programm

```
main()
{
    printf("hello, world");
}
```

Beispiel

betrachte Programm F:

```
main()
{
    int n, summe = 3, x, y, z;
    scanf(``%d'', &n);
    while (1) {
        for (x=1; x <= summe-2; x++)
            for (y=1; y <= summe-x-1; y++) {
                z = summe - x - y;
                if (pow(x,n) + pow(y,n) == pow(z,n))
                    printf("hello, world");
            } summe++;
    }
}
```

Das Programm F ist kein „hello, world“-Programm

Beispiel (Fortsetzung)

- Code repräsentiert den großen Fermat; $x^n + y^n = z^n$, für alle n
- widerlegt von Andrew Wiles, 1980er
- In der Simpons Folge “The Wizard of Evergreen Terrace” schreibt Homer die folgende (scheinbare) Lösung an die Tafel:

$$3987^{12} + 4365^{12} = 4472^{12} .$$

Das ist zwar falsch, aber auf einem einfachen Taschenrechner erscheint die Lösung (durch Rundungsfehler) als richtig.

Beispiel

betrachte Programm G

```
main()
{
    int n, x, y, z, test, summe=4;
    while (1) {
        test = 0;
        for (x=2; x <= summe; x++) {
            y = summe - x;
            if (is_prime(x) && is_prime(y)) test = 1;
        }
        if (!test) printf("hello, world");
        summe = summe + 2;
    }
}
```

Das Programm G ist **wahrscheinlich** kein „hello, world“-Programm¹

¹G ist kein „hello, world“-Programm für Zahlen $\leq 10^{17}$

Großer Fermat (1637)

Die Gleichung $a^n + b^n = c^n$ besitzt für $n > 2$ keine Lösung in \mathbb{N}



Vermutung von Goldbach (1742)

Jede gerade Zahl (≥ 2) kann als Summe zweier Primzahlen dargestellt werden



Satz

Es kann niemals ein Testprogramm für „hello, world“-Programme geben

Entscheidbarkeit & Unentscheidbarkeit

Definition (informell)

Ein Problem, das **algorithmisch** nicht lösbar ist, wird **unentscheidbar** genannt;
andernfalls heißt das Problem **entscheidbar**

Definition

Als **Halteproblem** bezeichnen wir das Problem, ob ein beliebiges Programm auf seiner Eingabe hält.

Definition

Postsches Korrespondenzproblem: Gegeben zwei gleich lange Listen von (nicht-leeren) Wörtern w_1, w_2, \dots, w_n und x_1, x_2, \dots, x_n . Gesucht sind Indizes i_1, i_2, \dots, i_m , sodass

$$w_{i_1} w_{i_2} \dots w_{i_m} = x_{i_1} x_{i_2} \dots x_{i_m}$$

Satz

Die folgenden Probleme sind **entscheidbar**:

- das Erfüllbarkeitsproblem der Aussagenlogik (SAT)
- das Wortproblem der Booleschen Algebra
- sei G eine KFG, ist $L(G) = \emptyset$?
- ...

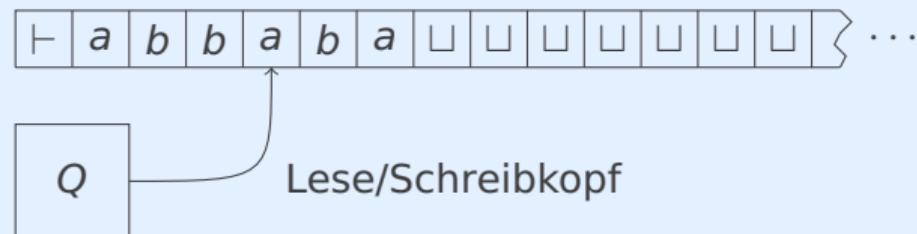
Satz

Die folgenden Probleme sind **unentscheidbar**:

- das Halteproblem
- das Postsche Korrespondenzproblem
- sei G eine KFG über Σ , ist $L(G) = \Sigma^*$?
- ...

Definition (informell)

deterministische, einbändige Turingmaschine (TM):



- Eine TM verwendet ein einseitig unendliches Band als Speicher
 - Zu Beginn der Berechnung steht die Eingabe auf dem Band
 - Der Speicher wird mit einem **Lese/Schreibkopf** gelesen oder beschrieben
 - Das Verhalten der TM wird durch die **endliche Kontrolle** Q kontrolliert

Definition (formal)

eine deterministische, einbändige Turingmaschine (TM) M ist ein 9-Tupel

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

sodass

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

Definition (Übergangsfunktion)

die Gleichung $\delta(p, a) = (q, b, d)$ bedeutet: Wenn die TM M im Zustand p das Symbol a liest, dann

- 1 M ersetzt a durch b auf dem Band
- 2 der Lese/Schreibkopf bewegt sich einen Schritt in die Richtung d
- 3 M wechselt in den Zustand q

Definition (Zusatzbedingungen)

- Der linke Endmarker darf nicht überschrieben werden

$$\forall p \in Q, \exists q \in Q \quad \delta(p, \textcolor{red}{\vdash}) = (q, \textcolor{red}{\vdash}, R)$$

- Wenn die TM akzeptiert/verwirft, bleibt die TM in diesem Zustand

$$\forall b \in \Gamma, \exists c, c' \in \Gamma \text{ und } d, d' \in \{L, R\}: \quad \delta(\textcolor{red}{t}, b) = (t, c, d)$$

$$\delta(\textcolor{red}{r}, b) = (r, c', d')$$

Beispiel

sei $M = (\{s, t, r, q_1, q_2, q_3\}, \{0, 1\}, \{\vdash, \sqcup, 0, 1, X, Y\}, \vdash, \sqcup, \delta, s, t, r)$ mit δ wie folgt

$p \in Q$	$a \in \Gamma$	$\delta(p, a)$	$p \in Q$	$a \in \Gamma$	$\delta(p, a)$
s	\vdash	(s, \vdash, R)	q_2	\vdash	(r, \vdash, R)
s	\sqcup	(r, \sqcup, R)	q_2	\sqcup	(r, \sqcup, R)
s	0	(q_1, X, R)	q_2	0	$(q_2, 0, L)$
s	1	$(r, 1, R)$	q_2	1	$(r, 1, R)$
s	X	(r, X, R)	q_2	X	(s, X, R)
s	Y	(q_3, Y, R)	q_2	Y	(q_2, Y, L)
q_1	\vdash	(r, \vdash, R)	q_3	\vdash	(r, \vdash, R)
q_1	\sqcup	(r, \vdash, R)	q_3	\sqcup	(t, \sqcup, R)
q_1	0	$(q_1, 0, R)$	q_3	0	$(r, 0, R)$
q_1	1	(q_2, Y, L)	q_3	1	$(r, 1, R)$
q_1	X	(r, X, R)	q_3	X	(r, X, R)
q_1	Y	(q_1, Y, R)	q_3	Y	(q_3, Y, R)
t	$*$	$(t, *, R)$	r	$*$	$(r, *, R)$

Konfiguration einer TM

Definition

eine Konfiguration einer TM M ist ein Tripel (p, x, n) , sodass

- 1 $p \in Q$ Zustand,
- 2 $x = y \sqcup^\infty$ Bandinhalt $y \in \Gamma^*$
- 3 $n \in \mathbb{N}$ Position des Lese/Schreibkopfes

Definition

Startkonfiguration bei Eingabe $x \in \Sigma^*$: $(s, \vdash x \sqcup^\infty, 0)$

Beispiel

Sei 0011 die Eingabe der TM M , dann ist die Startkonfiguration gegeben durch $(s, \vdash 0011 \sqcup^\infty, 0)$

Schrittfunktion von TMs

Definition

Relation $\xrightarrow[M]{1}$ ist wie folgt definiert:

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

z' ist String, den wir aus z erhalten, wenn z_n durch b ersetzt

Definition

reflexive, transitive Hülle $\xrightarrow[M]^*$:

1 $\alpha \xrightarrow[M]^0 \alpha$

2 $\alpha \xrightarrow[M]^{n+1} \beta$, wenn $\alpha \xrightarrow[M]^n \gamma \xrightarrow[M]^1 \beta$ für Konfiguration γ

3 $\alpha \xrightarrow[M]^* \beta$, wenn $\exists n \quad \alpha \xrightarrow[M]^n \beta$

Beispiel (Wiederholung)

sei $M = (\{s, t, r, q_1, q_2, q_3\}, \{0, 1\}, \{\vdash, \sqcup, 0, 1, X, Y\}, \vdash, \sqcup, \delta, s, t, r)$ mit δ wie folgt

$p \in Q$	$a \in \Gamma$	$\delta(p, a)$	$p \in Q$	$a \in \Gamma$	$\delta(p, a)$
s	\vdash	(s, \vdash, R)	q_2	\vdash	(r, \vdash, R)
s	\sqcup	(r, \sqcup, R)	q_2	\sqcup	(r, \sqcup, R)
s	0	(q_1, X, R)	q_2	0	$(q_2, 0, L)$
s	1	$(r, 1, R)$	q_2	1	$(r, 1, R)$
s	X	(r, X, R)	q_2	X	(s, X, R)
s	Y	(q_3, Y, R)	q_2	Y	(q_2, Y, L)
q_1	\vdash	(r, \vdash, R)	q_3	\vdash	(r, \vdash, R)
q_1	\sqcup	(r, \vdash, R)	q_3	\sqcup	(t, \sqcup, R)
q_1	0	$(q_1, 0, R)$	q_3	0	$(r, 0, R)$
q_1	1	(q_2, Y, L)	q_3	1	$(r, 1, R)$
q_1	X	(r, X, R)	q_3	X	(r, X, R)
q_1	Y	(q_1, Y, R)	q_3	Y	(q_3, Y, R)
t	$*$	$(t, *, R)$	r	$*$	$(r, *, R)$

Beispiel

Wir betrachten die Schrittfunktion für eine akzeptierende Berechnung von M bei Eingabe 0011:

$$\begin{aligned}(s, \vdash 0011 \sqcup^\infty, 0) &\xrightarrow[M]{1} (s, \vdash 0011 \sqcup^\infty, 1) \xrightarrow[M]{1} (q_1, \vdash X011 \sqcup^\infty, 2) \\&\xrightarrow[M]{1} (q_1, \vdash X011 \sqcup^\infty, 3) \xrightarrow[M]{1} (q_2, \vdash X0Y1 \sqcup^\infty, 2) \\&\xrightarrow[M]{1} (q_2, \vdash X0Y1 \sqcup^\infty, 1) \xrightarrow[M]{1} (s, \vdash X0Y1 \sqcup^\infty, 2) \\&\xrightarrow[M]{1} (q_1, \vdash XXY1 \sqcup^\infty, 3) \xrightarrow[M]{1} (q_1, \vdash XXY1 \sqcup^\infty, 4) \\&\xrightarrow[M]{1} (q_2, \vdash XXYY \sqcup^\infty, 3) \xrightarrow[M]{1} (q_2, \vdash XXYY \sqcup^\infty, 2) \\&\xrightarrow[M]{1} (s, \vdash XXYY \sqcup^\infty, 3) \xrightarrow[M]{1} (q_3, \vdash XXYY \sqcup^\infty, 4) \\&\xrightarrow[M]{1} (q_3, \vdash XXYY \sqcup^\infty, 5) \xrightarrow[M]{1} (t, \vdash XXYY \sqcup \sqcup^\infty, 6)\end{aligned}$$

Definition

eine TM M

- akzeptiert $x \in \Sigma^*$, wenn $\exists y, n:$

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

- verwirft $x \in \Sigma^*$, wenn $\exists y, n:$

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

- hält bei Eingabe x , wenn x akzeptiert oder verworfen
- hält nicht bei Eingabe x , wenn x weder akzeptiert, noch verworfen
- ist total, wenn M auf allen Eingaben hält

Definition

die Sprache einer TM M ist wie folgt definiert:

$$\textcolor{red}{L}(M) := \{x \in \Sigma^* \mid M \text{ akzeptiert } x\}$$

Satz

Sei L eine Sprache, die von einer TM akzeptiert wird. Dann ist L **rekursiv aufzählbar**.

Folgerung

Sei L eine Sprache, die von einer TM akzeptiert wird, dann existiert eine Grammatik G , sodass $L = L(G)$

Definition (Berechenbarkeit mit einer TM)

- Sei $M = (Q, \{\sqcap, \sqcup\}, \{\vdash, \sqcup, \sqcap, \sqcup\}, \vdash, \sqcup, \delta, s, t, r)$
- Eine partielle Funktion $f: \mathbb{N}^k \rightarrow \mathbb{N}$ heißt **M -berechenbar**, wenn:
$$f(n_1, \dots, n_k) = m \quad \text{gdw. } (s, \vdash \sqcap^{n_1} \sqcup \dots \sqcup \sqcap^{n_k} \sqcup^\infty, 0) \xrightarrow[M]{*} (t, \vdash \sqcap^m \sqcup^\infty, n)$$
- Eine partielle Funktion $f: \mathbb{N}^k \rightarrow \mathbb{N}$ heißt **berechenbar mit einer TM**, wenn eine TM M über dem Alphabet $\{\sqcap, \sqcup\}$ existiert, sodass $f M$ -berechenbar



Demo: Turing Machine Simulator



Frohe Feiertage & Guten Rutsch



Einführung in die Theoretische Informatik

Martin Avanzini Christian Dalvit Jamie Hochrainer

Georg Moser Johannes Niederhauser Jonas Schöpf

<https://tcs-informatik.uibk.ac.at>



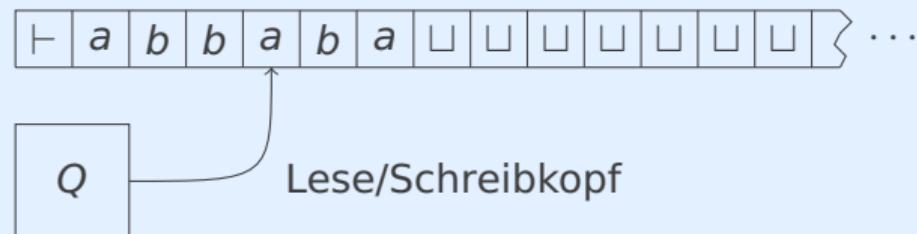
Frohes Neues!



Zusammenfassung

Definition (informell)

deterministische, einbändige Turingmaschine (TM):



- Eine TM verwendet ein einseitig unendliches Band als Speicher
 - Zu Beginn der Berechnung steht die Eingabe auf dem Band
 - Der Speicher wird mit einem **Lese/Schreibkopf** gelesen oder beschrieben
 - Das Verhalten der TM wird durch die **endliche Kontrolle** Q kontrolliert

Unentscheidbarkeit

Satz

Es kann niemals ein Testprogramm für „hello, world“-Programme geben

Definition (informell)

ein Problem, das nicht algorithmisch lösbar ist, heißt **unentscheidbar**

Satz

*die folgenden Probleme sind **unentscheidbar**:*

- 1** *das Halteproblem*
- 2** *das Postsche Korrespondenzproblem*
- 3** *ist eine beliebige kontextfreie Grammatik eindeutig*
- 4** *...*

Einführung in die Logik

Syntax & Semantik der Aussagenlogik, Formales Beweisen, Konjunktive und Disjunktive Normalformen

Einführung in die Algebra

Algebraische Strukturen, Boolesche Algebra, Universelle Algebra

Einführung in die Theorie der Formalen Sprachen

Grammatiken und Formale Sprachen, Chomsky-Hierarchie, Reguläre Sprachen, Kontextfreie Sprachen, Anwendungen von formalen Sprachen

Einführung in die Berechenbarkeitstheorie und Komplexitätstheorie

Algorithmisch unlösbare Probleme, **Turing Maschinen**, **Registermaschinen**, Komplexitätstheorie

Einführung in die Programmverifikation

Prinzipien der Analyse von Programmen, Verifikation nach Hoare

Turing Maschinen Berechenbarkeit

Beispiel

Wir beschreiben informell eine TM M , die zwei Zahlen x_1 und x_2 als Eingabe bekommt; M soll $x_1 - x_2$ berechnen und diese Differenz verdoppeln

Algorithmus

- wir kodieren die Eingabe x_1, x_2 durch Wörter über dem Alphabet $\{\Box\}$ der Länge x_1 bzw. x_2
- die beiden Eingaben werden durch Trennsymbol \Box getrennt
- Subtraktion funktioniert, indem wir sukzessive für alle \Box rechts von \Box ein \Box links mit \Box überschreiben
- Verdopplung funktioniert indem wir für jedes übrige \Box , $\Box\Box$ schreiben

Registermaschinen

Definition

Eine Registermaschine (RM) R ist ein Paar $R = ((x_i)_{1 \leq i \leq n}, P)$ sodass

- 1 $(x_i)_{1 \leq i \leq n}$ eine Sequenz von n Registern x_i , die natürliche Zahlen beinhalten
- 2 P ein Programm

Programme sind endliche Folgen von Befehlen und sind induktiv definiert:

- 1 Für jedes Register x_i sind die folgenden Instruktionen sowohl Befehle wie Programme: $x_i := x_i + 1$ und $x_i := x_i - 1$
- 2 wenn P_1, P_2 Programme sind, dann ist $P_1; P_2$ ein Programm
- 3 wenn P_1 ein Programm und x_i ein Register, dann ist

while $x_i \neq 0$ do P_1 end

sowohl ein Befehl als auch ein Programm

1 Zu Beginn der Berechnung steht die **Eingabe** (als natürliche Zahlen) in den Registern

2 Die Befehle

- $x_i := x_i + 1$
- $x_i := x_i - 1$

bedeuten, dass der Inhalt des Register x_i entweder um 1 erhöht oder vermindert wird

3 $P_1; P_2$ bedeutet, dass zunächst das Programm P_1 und dann das Programm P_2 ausgeführt wird

4 Der Befehl (und das Programm)

while $x_i \neq 0$ **do** P_1 **end**

bedeutet, der Schleifenrumpf P_1 wird ausgeführt, bis die Bedingung $x_i \neq 0$ falsch ist

Beispiel

Sei $R = ((x_i)_{1 \leq i \leq 5}, P)$ eine RM mit dem folgenden Programm:

Zuweisung $x_i := x_j$

Multiplikation

while $x_i \neq 0$ do

$x_i := x_i - 1$

end;

while $x_k \neq 0$ do

$x_k := x_k - 1$

end

while $x_j \neq 0$ do

$x_j := x_j + 1$;

$x_j := x_j - 1$;

$x_k := x_k + 1$

end;

while $x_k \neq 0$ do

$x_j := x_j + 1$;

$x_k := x_k - 1$

end

$x_3 := 0$;

while $x_1 \neq 0$ do

$x_1 := x_1 - 1$;

$x_4 := x_2$;

while $x_2 \neq 0$ do

$x_2 := x_2 - 1$;

$x_3 := x_3 + 1$

end;

$x_2 := x_4$

end

Bei Eingabe $(m, n, 0, 0, 0)$ berechnet R $(0, n, m \times n, n, 0)$

Berechenbarkeit mit einer RM

Definition

- sei $R = ((x_i)_{1 \leq i \leq n}, P)$ eine RM
- eine **partielle** Funktion $f: \mathbb{N}^k \rightarrow \mathbb{N}$, heißt ***R*-berechenbar**, wenn
$$f(n_1, \dots, n_k) = m \quad \text{gdw. } R \text{ startet mit } n_i \text{ in den Registern } x_i \text{ und endet mit } m \text{ im Register } x_{k+1} \text{ (und Eingaben } n_i \text{ in den Registern } x_i\text{)}$$
- Eine partielle Funktion $f: \mathbb{N}^k \rightarrow \mathbb{N}$ heißt **berechenbar auf einer RM**, wenn eine RM R existiert, sodass f R -berechenbar.

Satz

Jede partielle Funktion $f: \mathbb{N}^k \rightarrow \mathbb{N}$, die berechenbar auf einer RM ist, ist auf einer TM berechenbar und umgekehrt

Church-Turing-These („Naturgesetz“ der Informatik)

Jedes algorithmisch lösbarbare Problem ist auch mit Hilfe einer Turingmaschine lösbar.

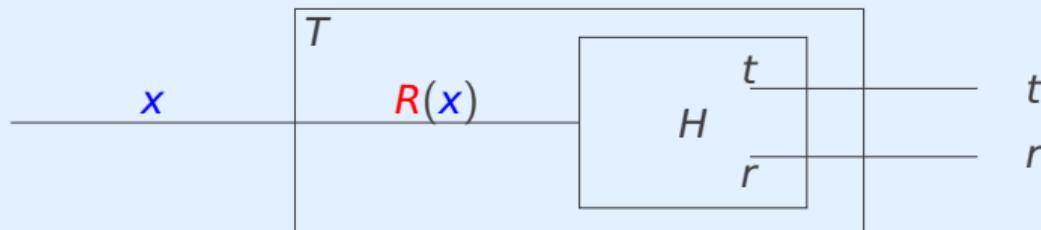
Definition (Turingreduktion)

angenommen

- L, M Sprachen über Σ
- $L \leqslant_T M$ mit $R: \Sigma^* \rightarrow \Sigma^*$
- die Reduktion R wird von TM T berechnet, sodass gilt

$$x \in L \Leftrightarrow R(x) \in M$$

Entscheidbarkeit von L , durch Entscheider H von M



Beispiel

Seien

$$L = \{x \in \{a, b\}^* \mid |x| \text{ ist gerade}\}$$

$$M = \{x \in \{a, b\}^* \mid x \text{ ist ein Palindrom gerader Länge}\}$$

dann gilt $L \leqslant_T M$

Reduktion

Wir geben eine (TM) berechenbare Abbildung $R: \{a, b\}^* \rightarrow \{a, b\}^*$ an, sodass $x \in L \Leftrightarrow R(x) \in M$:

- definiere R' , sodass $R'(a) := a$ und $R'(b) := a$
- definiere R als Erweiterung von R' auf Wörter
- R ist eine Stringfunktion, die ein Wort aus $\{a, b\}^n$ in das Wort a^n umwandelt
- Genau dann wenn n gerade ist, ist a^n ein Palindrom gerader Länge

Tabelle für $x \in L$ gdw $R(x) \in M$

$x \in L$	x	$R(x)$	$R(x) \in M$
✓	ϵ	ϵ	✓
✗	a	a	✗
✗	b	a	✗
✓	aa	aa	✓
✓	ab	aa	✓
✓	ba	aa	✓
✓	bb	aa	✓
✗	aaa	aaa	✗
⋮	⋮	⋮	⋮

wobei

$$L = \{x \in \{a, b\}^* \mid |x| \text{ ist gerade}\} \quad M = \{x \in \{a, b\}^* \mid x \text{ ist ein Palindrom gerader Länge}\}$$

Anwendungen von Reduktionen

Lemma

wenn $L \leqslant_T M$ und M rekursiv, dann ist L rekursiv

Unentscheidbarkeit

Unentscheidbarkeit eines Problems zeigt man mittels Reduktion **von** einem unentscheidbaren Problem (zum Beispiel das Halteproblem) auf das betrachtete Problem

Satz

es kann kein Testprogramm für “hello, world” Programme geben

Beweis.

$\text{HP} \leqslant_T \text{“hello, world” Programme}$

Satz

Sei Σ ein Alphabet und $L \subseteq \Sigma^*$ rekursiv; dann ist $\sim L$ rekursiv

Beweis.

Da L rekursiv ist, gibt es eine totale TM M mit $L = L(M)$. Wir definieren eine TM M' , wobei der akzeptierende und der verwerfende Zustand von M vertauscht werden. Weil M total ist, ist auch M' total. Somit akzeptiert M' ein Wort genau dann, wenn M es verwirft und es folgt $\sim L = L(M')$, d.h. $\sim L$ ist rekursiv. ■

Satz

Jede rekursive Menge ist rekursiv aufzählbar. Andererseits ist nicht jede rekursiv aufzählbare Menge rekursiv.

Beweis.

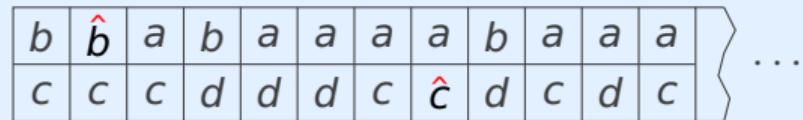
Der erste Teil des Satzes ist eine Konsequenz der Definitionen; der zweite Teil wird in „Diskrete Strukturen“ bewiesen werden. ■

Satz

Wenn L und $\sim L$ rekursiv aufzählbar sind, dann ist L rekursiv.

Beweis.

- \exists TM M_1, M_2 mit $L = L(M_1)$ und $\sim(L) = L(M_2)$
- definiere TM $\textcolor{red}{M'}$, sodass das Band zwei Hälften hat

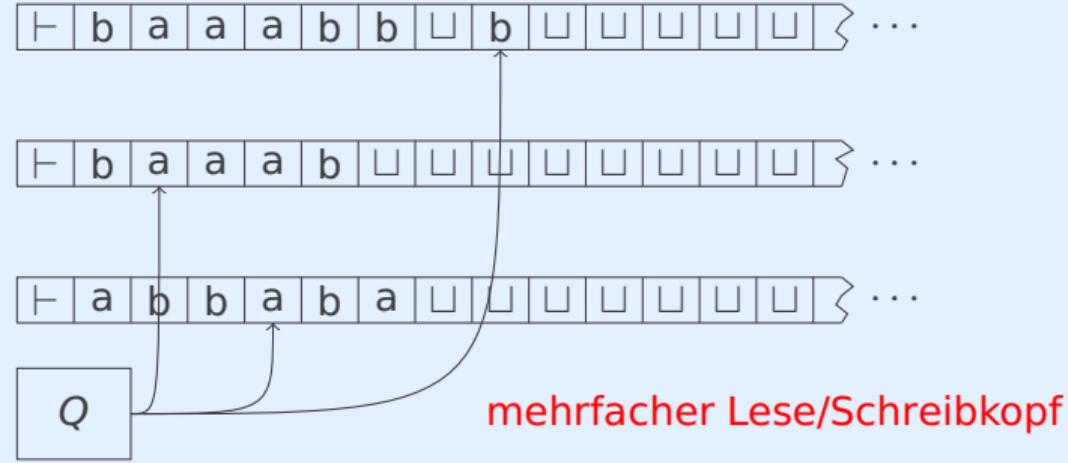


- M_1 wird auf der oberen und M_2 auf der unteren Hälfte simuliert
- wenn $M_1 x$ akzeptiert, $\textcolor{red}{M'}$ akzeptiert x
- wenn $M_2 x$ akzeptiert, $\textcolor{red}{M'}$ verwirft x



Definition (informell)

Erweiterung um mehrere Bänder und Lese/Schreibköpfe:



Definition

$$\delta: Q \times \Gamma^3 \rightarrow Q \times \Gamma^3 \times \{L, R\}^3$$

Satz

Sei M eine k -bändige TM. Dann existiert eine (einbändige) TM M' , sodass $L(M) = L(M')$

Beweisskizze.

- wir simulieren die Bänder übereinander, oBdA sei $k = 2$
- wir erweitern das Alphabet von M'

a
c

\hat{a}
c

a
\hat{c}

\hat{a}
\hat{c}

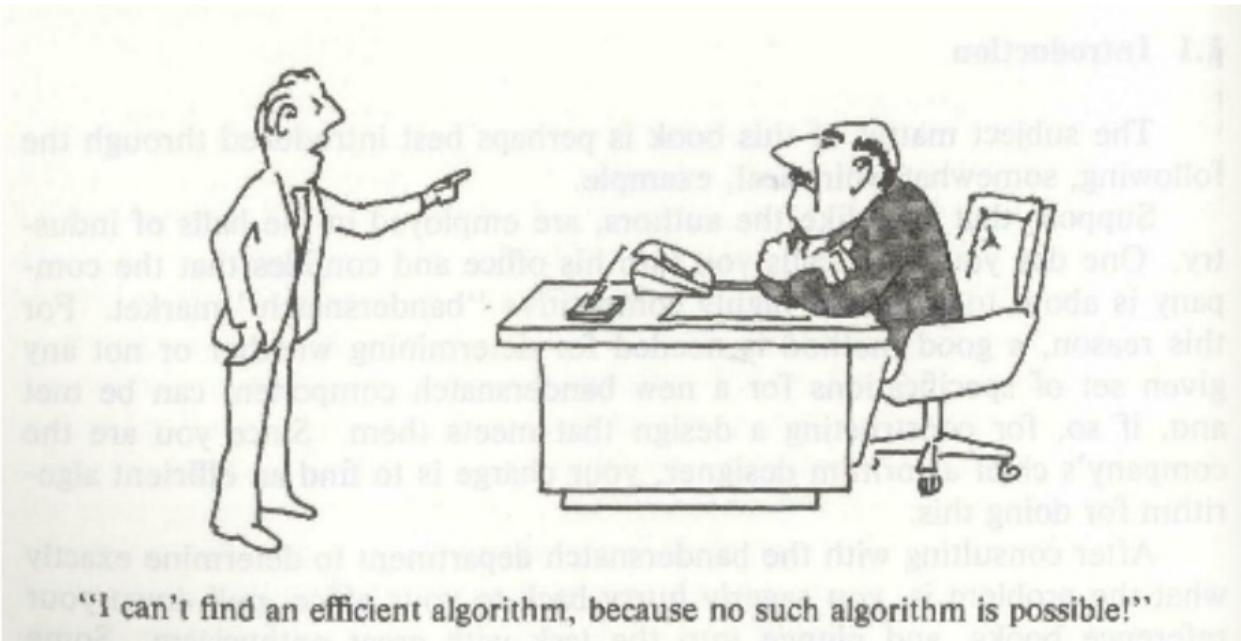
- Band von M' kann folgende Gestalt haben:

b	\hat{b}	a	b	a	a	a	a	b	a	a	a	...
c	c	c	d	d	d	c	\hat{c}	d	c	d	c	

- alle Bänder von M sind nun repräsentiert und die Leseköpfe werden durch die Zusatzmarkierung $\hat{\cdot}$ ausgedrückt



Berechenbarkeitstheorie, graphisch



"I can't find an efficient algorithm, because no such algorithm is possible!"



Einführung in die Theoretische Informatik

Martin Avanzini Christian Dalvit Jamie Hochrainer

Georg Moser Johannes Niederhauser Jonas Schöpf

<https://tcs-informatik.uibk.ac.at>



Zusammenfassung

Satz

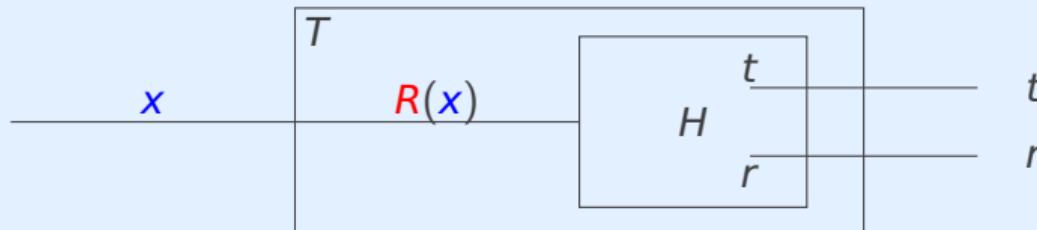
Jede partielle Funktion $f: \mathbb{N}^k \rightarrow \mathbb{N}$, die berechenbar auf einer RM ist, ist auf einer TM berechenbar und umgekehrt

Definition (Turingreduktion)

angenommen

- L, M Sprachen über Σ
- $L \leqslant_T M$ mit $R: \Sigma^* \rightarrow \Sigma^*$
- die Reduktion R wird von TM T berechnet, sodass gilt $x \in L \Leftrightarrow R(x) \in M$

Entscheidbarkeit von L , durch Entscheider H von M



Einführung in die Logik

Syntax & Semantik der Aussagenlogik, Formales Beweisen, Konjunktive und Disjunktive Normalformen

Einführung in die Algebra

Algebraische Strukturen, Boolesche Algebra, Universelle Algebra

Einführung in die Theorie der Formalen Sprachen

Grammatiken und Formale Sprachen, Chomsky-Hierarchie, Reguläre Sprachen, Kontextfreie Sprachen, Anwendungen von formalen Sprachen

Einführung in die Berechenbarkeitstheorie und Komplexitätstheorie

Algorithmisch unlösbare Probleme, Turing Maschinen, Registermaschinen, Komplexitätstheorie

Einführung in die Programmverifikation

Prinzipien der Analyse von Programmen, **Verifikation nach Hoare**



Einführung in die Komplexitätstheorie

Definition

Komplexitätstheorie analysiert Algorithmen und Probleme:

Welche Ressourcen benötigt ein bestimmter Algorithmus oder ein Problem?

Ressourcen

- Speicherplatz
- Rechenzeit
- ...

Problem & Algorithmus

Wir unterscheiden zwischen

- der Komplexität **eines Algorithmus**
- der Komplexität **eines Problems**

Algorithmus von Quine: $2^{c \cdot n}$
(wobei n die maximale binäre Länge der Eingabe)

SAT ist in NP

Laufzeitkomplexität

Definition

sei M eine totale TM

- die **Laufzeitkomplexität** von M ist Funktion $T: \mathbb{N} \rightarrow \mathbb{N}$, wobei T wie folgt definiert
$$T(n) := \max\{m \mid M \text{ hält bei Eingabe } x, |x| = n, \text{ nach } m \text{ Schritten}\}$$
- $T(n)$ bezeichnet die **Laufzeit** von M , wenn n die Länge der Eingabe
- M heißt **T -Zeit-Turingmaschine**

Definition

sei $T: \mathbb{N} \rightarrow \mathbb{N}$ eine numerische Funktion

$$\text{DTIME}(T) := \{L(M) \mid M \text{ ist eine mehrbändige TM mit Laufzeit (ungefähr) in } T\}$$

NB: Formal gilt $\exists c \in \mathbb{R}^+ \exists m \forall n \geq m$ Laufzeit von M bei Eingabe $x \leq c \cdot T(n)$, wobei $|x| = n$.

Die Klasse P und NP

Definition

$$\text{P} := \bigcup_{k \geq 1} \text{DTIME}(n^k)$$

Beispiel

betrachte SAT als Sprache:

$$\text{SAT} = \{F \mid F \text{ Formel mit erfüllbarer Belegung } v\}$$

es gilt $\text{SAT} \in \text{DTIME}(2^n)$, aber es ist nicht bekannt ob $\text{SAT} \in \text{P}$

Definition

- ein **Verifikator** einer Sprache $L \subseteq \Sigma^*$, ist ein Algorithmus V sodass
$$L = \{x \in \Sigma^* \mid \exists c, \text{ sodass } V \text{ akzeptiert Eingabe } (x, c)\}$$
- ein **polytime Verifikator** ist ein Verifikator mit (ungefährer) Laufzeit n^k wobei $|x| = n$
- Wort c wird **Zertifikat** genannt

Definition

NP ist die Klasse der Sprachen, die einen polytime Verifikator haben

Beispiel

- Es gilt $SAT \in NP$.
- Als Zertifikat wählen wir die (erfüllende) Belegung v für F . Für jede Belegung v kann leicht (in polynomieller Zeit) nachgewiesen werden, ob $v(F) = T$.

Reduktionen (in polynomieller Zeit)

Definition

- 1 $\exists k\text{-Band TM } M \text{ mit Eingabealphabet } \Sigma$
- 2 M läuft in polynomieller Zeit
- 3 bei Eingabe $x \in \Sigma^*$, schreibt M , $R(x)$ auf das (erste) Band
dann heißt $R : \Sigma^* \rightarrow \Delta^*$ in polynomieller Zeit berechenbar

Definition

- 1 $\exists R : \Sigma^* \rightarrow \Delta^*$
- 2 R berechenbar in polynomieller Zeit
- 3 für $L \subseteq \Sigma^*$, $M \subseteq \Delta^*$ gilt $x \in L \Leftrightarrow R(x) \in M$
dann ist L in polynomieller Zeit auf M reduzierbar; kurz: $L \leqslant^p M$

Beispiel (Wiederholung)

Seien

$$L = \{x \in \{a, b\}^* \mid |x| \text{ ist gerade}\}$$

$$M = \{x \in \{a, b\}^* \mid x \text{ ist ein Palindrom gerader Länge}\}$$

dann gilt $L \leqslant^p M$

Polynomielle Reduktion

Wir geben eine **polynomiell** berechenbare Abbildung $R: \{a, b\}^* \rightarrow \{a, b\}^*$ an, sodass $x \in L \Leftrightarrow R(x) \in M$:

- definiere R' , sodass $R'(a) := a$ und $R'(b) := a$
- definiere R als Erweiterung von R' auf Wörter
- R ist eine Stringfunktion, die ein Wort aus $\{a, b\}^n$ in das Wort a^n umwandelt
- Genau dann wenn n gerade ist, ist a^n ein Palindrom gerader Länge

Definition

- 1 \mathcal{C} eine beliebige Komplexitätsklasse
- 2 L eine Sprache über Σ
- 3 \forall Sprachen $M \in \mathcal{C}$ gilt: $M \leq^p L$

dann ist $L \leq^p$ -hart für \mathcal{C}

Beispiel

SAT ist \leq^p -hart für NP, dh. jedes Problem in NP ist auf SAT reduzierbar und außerdem ist $SAT \in NP$

Definition

für eine Sprache L , sei

- 1 $L \leq^p$ -hart für \mathcal{C} und
- 2 $L \in \mathcal{C}$

dann ist $L \leq^p$ -vollständig für \mathcal{C} oder (kurz) \mathcal{C} -vollständig



Verifikation nach Hoare

Prädikatenlogik (informell)

- Die Prädikatenlogik ist eine Logik, deren Ausdrucksstärke über die der Aussagenlogik weit hinausgeht
- Die wichtigste Erweiterung sind **Prädikatensymbole** und **Quantoren**
- **Prädikatensymbole** erlauben es uns, über Elemente einer Menge Aussagen zu treffen

Sprache einer Prädikatenlogik

Eine Prädikatenlogik durch eine **Sprache** beschrieben, diese Sprache enthält:

1 Funktionssymbole und Prädikatensymbole; Variablen

2 
Gleichheit Junktoren Quantoren

Beispiel

- Sei 7 eine Konstante und ist_prim ein Prädikatensymbol
- Wir schreiben ist_prim(7), um auszudrücken, dass 7 eine Primzahl

Definition

Ein Ausdruck der mit Hilfe von Variablen und Funktionssymbolen gebildet wird, heißt **Term**

Definition

- 1 Sei P ein Prädikatensymbol
- 2 Seien t_1, \dots, t_n Terme

Dann nennen wir die Ausdrücke $P(t_1, \dots, t_n)$ und $t_1 = t_2$ **Atome** oder **atomare Formel**

Definition (Zusicherungen)

Wir definieren **Zusicherungen** induktiv:

- 1 Atome sind Zusicherungen
- 2 Wenn A und B Zusicherungen sind, dann sind auch die folgenden Ausdrücke, Zusicherungen:
 $\neg A$ $(A \wedge B)$ $(A \vee B)$ $(A \rightarrow B)$

Konvention

Zusicherungen werden **Formeln** genannt

Definition

Interpretationen \mathcal{I} werden verwendet, um den Ausdrücken der Prädikatenlogik eine **Bedeutung** zu geben

Beispiel

- Wir betrachten die Konstante 7 und das Prädikat ist_prim
- Interpretation \mathcal{I} legt fest, dass 7 als die Zahl sieben zu verstehen ist
- \mathcal{I} legt fest, dass das Atom $\text{ist_prim}(n)$ genau dann wahr ist, wenn n eine Primzahl

Beobachtung

- 1 Mit Hilfe von Interpretationen wird der Wahrheitsgehalt von Atomen bestimmt
- 2 Ist die Wahrheit von Atomen in \mathcal{I} definiert, wird die Wahrheit einer beliebigen Formel durch die Bedeutung der Junktoren bestimmt

Beispiel

Die Formel $\text{ist_prim}(x) \wedge x = 7$ bedeutet, dass x die Primzahl 7 ist

Definition

Sei \mathcal{I} eine Interpretation und F eine Formel, wir schreiben $\mathcal{I} \models F$, wenn die Formel F in der Interpretation \mathcal{I} wahr ist

Beispiel

Wenn x die Primzahl 7 ist, dann gilt $\mathcal{I} \models \text{ist_prim}(x) \wedge x = 7$

Definition

Die Konsequenzrelation $A \models B$ gilt, gdw. für alle Interpretationen \mathcal{I} :

$$\mathcal{I} \models A \text{ impliziert } \mathcal{I} \models B$$

Beispiel

Seien $x_1 > 4$ und $x_1 + 1 > 5$ Atome, es gilt:

$$x_1 > 4 \models x_1 + 1 > 5$$

Hoare-Tripel

Definition

- Sei P ein while-Programm (ein Programm einer Registermaschine)
- Seien Q und R Zusicherungen
- Ein **Hoare-Tripel** ist wie folgt definiert:

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

- Q wird **Vorbedingung**
- R wird **Nachbedingung** genannt

Beispiel

Seien $x_1 > 4$, $x_1 > 5$ Zusicherungen und $x_1 := x_1 + 1$ ein Programm,
dann ist $\{x_1 > 4\} x_1 := x_1 + 1 \{x_1 > 5\}$ ein Hoare-Tripel

Definition

- Ein Hoare-Tripel $\{Q\} P \{R\}$ ist **wahr**, wenn
 - 1 Q **vor** der Ausführung von P gilt
 - 2 R **nach** der Ausführung von P gilt
 - 3 unter der Voraussetzung, dass P **terminiert**
- Wenn $\{Q\} P \{R\}$ wahr, dann ist P korrekt in Bezug auf Q und R
- Dann sagen wir auch P ist **partiell korrekt**
- Das Programm P ist **total korrekt**, wenn es partiell korrekt ist und terminiert

Beispiel

Die folgenden Hoare-Tripel

$$\{x_1 > 4\} x_1 := x_1 + 1 \{x_1 > 5\} \quad \{x_2 = 0\} x_2 := x_2 - 1 \{x_2 = 0\}$$

sind wahr und die jeweiligen Programme **total korrekt**

Hoare-Kalkül

Definition

Die Regeln des **Hoare-Kalkül** sind wie folgt definiert:

$$\begin{array}{c} [z] \quad \frac{}{\{Q\{x \mapsto t\}\} x := t \{Q\}} \quad [a] \quad \frac{\{Q'\} P \{R'\}}{\{Q\} P \{R\}} Q \models Q', R' \models R \\ [s] \quad \frac{\{Q\} P_1 \{R\} \quad \{R\} P_2 \{S\}}{\{Q\} P_1; P_2 \{S\}} \quad [w] \quad \frac{\{I \wedge B\} P \{I\}}{\{I\} \text{ while } B \text{ do } P \text{ end } \{I \wedge \neg B\}} \end{array}$$

Ist ein Hoare-Tripel in diesem Kalkül ableitbar, dann ist es wahr

Beispiel

$$\frac{\overline{\{x_1 + 1 > 5\} x_1 := x_1 + 1 \{x_1 > 5\}}}{\{x_1 > 4\} x_1 := x_1 + 1 \{x_1 > 5\}} \frac{[z]}{[a], x_1 > 4 \models x_1 + 1 > 5}$$

Beispiel

Wir betrachten das folgende einfache while-Programm P :

```
while  $x_i \neq 0$  do  
     $x_i := x_i - 1$   
end
```

und zeigen $\{x_i \geq 0\} P \{x_i = 0\}$

$$\frac{\frac{\frac{\{x_i - 1 \geq 0\} x_i := x_i - 1 \{x_i \geq 0\}}{[z]} \{x_i \geq 0 \wedge x_i \neq 0\} x_i := x_i - 1 \{x_i \geq 0\}}{[a]} \{x_i \geq 0\} P \{x_i \geq 0 \wedge x_i = 0\}}{[w]} \{x_i \geq 0\} P \{x_i = 0\} [a]$$

wir verwenden:

- 1 $x_i \geq 0 \wedge x_i = 0 \models x_i = 0$
- 2 die Schleifeninvariante $x_i \geq 0$
- 3 $x_i \geq 0 \wedge x_i \neq 0 \models x_i - 1 \geq 0$



Einführung in die Theoretische Informatik

Martin Avanzini Christian Dalvit Jamie Hochrainer

Georg Moser Johannes Niederhauser Jonas Schöpf

<https://tcs-informatik.uibk.ac.at>



Zusammenfassung

Zusammenfassung der letzten LVA

Definition

- ein **Verifikator** einer Sprache $L \subseteq \Sigma^*$, ist ein Algorithmus V sodass
$$L = \{x \in \Sigma^* \mid \exists c, \text{ sodass } V \text{ akzeptiert Eingabe } (x, c)\}$$
- ein **polytime Verifikator** ist ein Verifikator mit (ungefährer) Laufzeit n^k wobei $|x| = n$

Definition

NP ist die Klasse der Sprachen, die einen polytime Verifikator haben

Example

- Es gilt $SAT \in NP$; außerdem ist SAT NP-hart.
- Also ist SAT NP-vollständig.

Hoare-Kalkül

Definition

Die Regeln des **Hoare-Kalkül** sind wie folgt definiert:

$$\begin{array}{c} [z] \quad \frac{\{Q\{x \mapsto t\}\} \ x := t \ \{Q\}}{\{Q\} \ P \ \{R\}} \quad [a] \quad \frac{\{Q'\} \ P \ \{R'\}}{\{Q\} \ P \ \{R\}} \ Q \models Q', R' \models R \\ [s] \quad \frac{\{Q\} \ P_1 \ \{R\} \quad \{R\} \ P_2 \ \{S\}}{\{Q\} \ P_1; P_2 \ \{S\}} \quad [w] \quad \frac{\{I \wedge B\} \ P \ \{I\}}{\{I\} \text{ while } B \text{ do } P \text{ end } \{I \wedge \neg B\}} \end{array}$$

Ist ein Hoare-Tripel in diesem Kalkül ableitbar, dann ist es wahr

Beispiel

$$\frac{\overline{\{x_1 + 1 > 5\} \ x_1 := x_1 + 1 \ \{x_1 > 5\}}}{\{x_1 > 4\} \ x_1 := x_1 + 1 \ \{x_1 > 5\}} \quad \begin{array}{l} [z] \\ [a], x_1 > 4 \models x_1 + 1 > 5 \end{array}$$



Komplexitätstheorie

Beispiel

Wir betrachten das Rucksackproblem

- gegeben, n verschiedene Gegenstände mit einem bestimmten Gewicht g_i und Wert w_i ($1 \leq i \leq n$)
- gesucht, eine optimale Auswahl der Gegenstände, sodass Wert maximiert und Gewicht minimiert

Beispiel (Fortsetzung)

Das Rucksackproblem als Entscheidungsproblem mit Maximalgewicht G und Minimalwert W

$$\text{KNAPSACK} := \{(g_i)_{1 \leq i \leq n}, (w_i)_{1 \leq i \leq n}, G, W \mid \text{es existiert } I \text{ mit } \sum_{j \in I} g_j \leq G, \sum_{j \in I} w_j \geq W\}$$

Hier gilt $I \subseteq \{i \mid 1 \leq i \leq n\}$. Wir zeigen, dass $\text{KNAPSACK} \in \text{NP}$

Beispiel (Fortsetzung)

- zunächst betrachten wir lösbare und unlösbarer Instanzen
- sei etwa $n = 3$ mit $g_1 = g_2 = g_3 = 1$, $w_1 = w_2 = w_3 = 1$ und Minimalwert $W = 3$, aber Maximalgewicht $G < 3$; dann müssen alle drei Gegenstände in I gewählt werden, damit $\sum_{j \in I} w_j \geq 3$
- anderseits gilt aber dann $\neg(\sum_{j \in I} g_j < 3)$; diese Instanz ist also nicht lösbar
- eine lösbarer Instanz für $n = 3$, wäre etwa $g_1 = 3, g_2 = 2, g_3 = 1$
 $w_1 = w_2 = w_3 = 1$ mit $G = 2$ und $W = 1$
- nun überlegen wir welches Zertifikat wir wählen sollten, damit die Aufgabe leicht (= in polynomieller Zeit) lösbar wird
- gegeben Indexmenge I , ist es offensichtlich einfach zu prüfen, dass gilt:
 - 1 $\sum_{j \in I} g_j \leq G$
 - 2 $\sum_{j \in I} w_j \geq W$
- also wählen wir I als Zertifikat, der polytime Verifier nimmt dann die gegebene Instanz $((g_i)_{1 \leq i \leq n}, (w_i)_{1 \leq i \leq n}, G, W)$ und prüft die Bedingungen

Formale Sprachen und Komplexitätstheorie

- die Chomskyhierarchie beschreibt den Zusammenhang zwischen Klassen von formalen Sprachen (regulär, kontextfrei, ...)
- ähnlich bezeichnen Komplexitäsklassen, Klassen von formalen Sprachen (P, NP, ...)

Satz

es gelten die

- *Die Klasse der regulär Sprachen ist in P enthalten*
- *Die Klasse der kontextfreien Sprachen ist auch in P enthalten*
- *Die Klasse der kontextsensitiven Sprachen ist in EXPTIME enthalten, wobei*

$$\text{EXPTIME} := \bigcup_{k \geq 1} \text{DTIME}(2^{k \cdot n})$$

Komplexitätstheorie. graphisch



"I can't find an efficient algorithm, but neither can all these famous people."



Programmverifikation

Wozu Programmverifikation



- Ariane-5
- Fehler in der Datenkonvertierung
- USD 370 Millionen



- Intel Pentium FDIV-Bug
- Falsche Berechnungen
- USD 475 Millionen



- Gepäckverteilung (Denver)
- Desaster
- USD 560 Millionen



- Blue Screen of Death
- ziemlich lästig

Begutachtung

- Der Code wird von ähnlich qualifizierten Programmierern kontrolliert
- subtile Fehler werden leicht übersehen

Testen

- dynamische Technik, bei der das Programm ausgeführt wird
- Wie wird die richtige Testumgebung geschaffen?

Formal Methoden

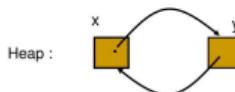
- erlauben die frühe Integration der Verifikation in die Softwareentwicklung
- sind effizienter als andere Methoden (höhere Erkennensrate von Fehlern)
- sind (im Besonderen wenn automatisierbar) schneller anwendbar

Beispiele formalen Softwareverifikation



Separation Logic

Formula : $x \mapsto y * y \mapsto x$



Beispiel (Hoare Kalkül)

Gegeben seien P , Q und R :

$$(P) \quad x_1 := x_1 - 1; \quad x_1 := x_1 - 1; \quad x_1 := x_1 - 1$$

$$(Q) \quad x_1 = 10$$

$$(R) \quad \text{prim}(x_1)$$

- das Prädikatsymbol $\text{prim}(x)$ ist wahr, wenn x eine Primzahl ist
- wir zeigen totale Korrektheit von P

Beispiel (Fortsetzung)

Zunächst betrachten wir die korrekte Ableitung Π_1 für die erste Zuweisung

$$\overline{\{x_1 - 1 = 9\} \quad x_1 := x_1 - 1 \quad \{x_1 = 9\}} \quad [z]$$

in ähnlicher Weise können wir Ableitungen für die zweite und dritte Zuweisung, bezeichnet mit Π_2 , Π_3 definieren

Beispiel (Fortsetzung)

$$\frac{\frac{\frac{\{x_1 - 1 = 9\} \ x_1 := x_1 - 1 \ \{x_1 = 9\}}{\{x_1 = 10\} \ x_1 := x_1 - 1 \ \{x_1 = 9\}} \text{ [a]}}{\{x_1 = 10\} \ x_1 := x_1 - 1; x_1 := x_1 - 1; x_1 := x_1 - 1 \ \{\text{prim}(x_1)\}} \text{ [s]}}$$

wobei Ψ wie folgt definiert ist

$$\frac{\frac{\frac{\{x_1 - 1 = 8\} \ x_1 := x_1 - 1 \ \{x_1 = 8\}}{\{x_1 = 9\} \ x_1 := x_1 - 1 \ \{x_1 = 8\}} \text{ [a]}}{\frac{\{\text{prim}(x_1 - 1)\} \ x_1 := x_1 - 1 \ \{\text{prim}(x_1)\}}{\{x_1 = 8\} \ x_1 := x_1 - 1 \ \{\text{prim}(x_1)\}} \text{ [a]}} \text{ [z]}}{\{x_1 = 9\} \ x_1 := x_1 - 1; x_1 := x_1 - 1 \ \{\text{prim}(x_1)\}} \text{ [s]}}$$

wir verwenden die folgenden Konsequenzrelation bei den Abschwächungen

- $(x_1 = 10) \models (x_1 - 1 = 9)$
- $(x_1 = 9) \models (x_1 - 1 = 8)$
- $(x_1 = 8) \models (\text{prim}(x_1 - 1))$



Prüfungsorganisation und -vorbereitung

Organisation der Vorlesungsklausur, 11. Februar

1te Klausur

- Bitte registrieren Sie sich heute (!) online für die Klausur
- Die Klausur findet in Präsenz von 10:15-11:45 im **Großen Hörsaal** und **HSB 3** statt
- Aufteilung auf die Hörsäle anhand des Familiennamens:
 - Studierende deren Familiennamen mit **A-K** angefängt: Großer Hörsaal
 - Studierende deren Familiennamen **L-Z**: HSB 3
- Studierendenausweis wird **vor** der Klausur kontrolliert, ohne Anmeldung keine Klausur
- Prüfungsstoff ist alles
- Die Prüfung is **open-book**: Unterlagen sind erlaubt
- Alte Klausuren (inkl. Musterlösungen) sind online

2te und 3te Klausur

- Die 2te Klausur findet am 11. März, die 3te Klausur im Juni/Juli statt
- Bitte melden Sie sich dann **rechtzeitig** online für die Klausur an
- Es besteht die Möglichkeit die SL nochmals im Sommersemester zu besuchen, um sich für die 3te Klausur vorzubereiten

10 Multiple-Choice Fragen

- jeweils 5-10 Antwortmöglichkeiten
- die Anzahl der richtigen Antworten ist **nicht** angegeben
- Korrekte Antworten werden positiv, falsche Antwort negativ bewertet
- Die Aufgaben sind randomisiert
- Während der Klausur können Sie sich bei Unklarheiten an die Aufsichtspersonen richten, Fragen zum Stoff werden natürlich nicht beantwortet

Organisation der SL Klausur

SL Klausur am Freitag, den 4. Februar

- SL Klausur findet am **4.2.** entweder in Präsenz in den jeweiligen Gruppen oder online (14:15–15:00) statt
- In jedem Fall dauert die Klausur 45' und ist **open-book**, dh. es dürfen Unterlagen verwendet werden; es ist keine weitere Anmeldung erforderlich
- Es werden drei Multiple-Choice Fragen zu lösen sein mit größerem Umfang als in der Vorlesungsklausur

Zweite SL Klausur

- Die zweite SL Klausur findet am 4. März statt; der genaue Termin wird noch bekanntgegeben
- Sie können die SL auch im Rahmen der LVA im Sommersemester nochmals besuchen (Teil der STEOP)



Prüfungsvorbereitung