Vorlesung
Logik in der Informatik

Wintersemester

Prof. Dr. Nicole Schweikardt

Lehrstuhl Logik in der Informatik
Institut für Informatik
Humboldt-Universität zu Berlin

Kapitel 1:

Einleitung

Abschnitt 1.1:
Von der Bibel bis zu den Simpsons

Logik

- altgriechisch „logos“: Vernunft
- die Lehre des vernünftigen Schlussfolgerns
- Teilgebiet u.a. der Disziplinen Philosophie, Mathematik und Informatik
- zentrale Frage:

  Wie kann man Aussagen miteinander verknüpfen, und auf welche Weise kann man formal Schlüsse ziehen und Beweise durchführen?

Das Lügnerparadoxon von Epimenides

Brief des Paulus an Titus 1:12-13:

> *Es hat einer von ihnen gesagt, ihr eigener Prophet:*
> *Die Kreter sind immer Lügner, böse Tiere und faule Bäuche.*

Angenommen, die Aussage des Propheten ist wahr.

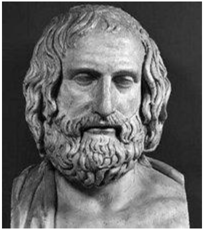
Da der Prophet selbst Kreter ist, lügt er also immer (und ist ein böses Tier und ein fauler Bauch). Dann hat er aber insbesondere in dem Satz „*Die Kreter sind immer Lügner, böse Tiere und faule Bäuche*“ gelogen. D.h. die Aussage des Propheten ist *nicht* wahr. Dies ist ein Widerspruch!

Angenommen, die Aussage des Propheten ist falsch.

Dann gibt es Kreter, die nicht immer Lügner, böse Tiere und faule Bäuche sind. Dies stellt keinen Widerspruch dar.

Insgesamt wissen wir also, dass der Prophet in seiner obigen Aussage nicht die Wahrheit gesagt hat.

Protagoras und sein Student Euthalus vor Gericht



Protagoras (490 – 420 v.Chr.)

Quelle: `http://www.greatthoughtstreasury.com/author/protagoras`

Euthalus studierte die Kunst der Argumentation beim Meister Protagoras, um Anwalt zu werden.
Er vereinbart mit Protagoras, die Gebühren für den Unterricht zu bezahlen, sobald er seinen ersten Prozess gewonnen hat.

Aber dann zögert Euthalus seine Anwaltstätigkeit immer weiter hinaus, und schließlich beschließt Protagoras, seine Gebühren einzuklagen.
Euthalus verteidigt sich selbst ...

Protagoras denkt:

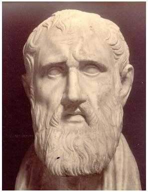
Wenn ich den Prozess gewinne, muss Euthalus gemäß Gerichtsbeschluss zahlen.
Wenn ich den Prozess verliere, muss Euthalus gemäß unserer Vereinbarung zahlen, da er dann seinen ersten Prozess gewonnen hat.

Euthalus denkt:

Wenn ich den Prozess gewinne, muss ich gemäß Gerichtsbeschluss nicht zahlen.
Wenn ich den Prozess verliere, muss ich gemäß unserer Vereinbarung nicht zahlen.

Achilles und die Schildkröte

Achilles und die Schildkröte laufen ein Wettrennen. Achilles gewährt der Schildkröte einen Vorsprung. Zenon behauptet, dass Achilles die Schildkröte niemals einholen kann.



Zenon von Elea (490 – 425 v.Chr.) *Quelle:* `http://aefucr.blogspot.de/2008/04/resolucin-de-la-paradoja-de-zenn-de.html`

Zenons Begründung: Zu dem Zeitpunkt, an dem Achilles den Startpunkt der Schildkröte erreicht, ist die Schildkröte schon ein Stück weiter. Etwas später erreicht Achilles diesen Punkt, aber die Schildkröte ist schon etwas weiter. Wenn Achilles diesen Punkt erreicht, ist die Schildkröte wieder etwas weiter. So kann Achilles zwar immer näher an die Schildkröte herankommen, sie aber niemals einholen.

Auflösung durch die Infinitesimalrechnung:

Gottfried Wilhelm von Leibniz (1646 – 1716)
und Isaac Newton (1643 – 1727)
Quelle: `http://www-history.mcs.st-and.ac.uk/PictDisplay/Leibniz.html`
und *Quelle:* `http://de.wikipedia.org/wiki/Isaac_Newton`

Der Barbier von Sonnenthal

Im Städtchen Sonnenthal (in dem bekanntlich viele seltsame Dinge passieren) wohnt ein Barbier, der genau diejenigen männlichen Einwohner von Sonnenthal rasiert, die sich nicht selbst rasieren.

Frage: Rasiert der Barbier sich selbst?

Angenommen, der Barbier rasiert sich selbst.

Da er ein männlicher Einwohner von Sonnenthal ist, der sich selbst rasiert, wird er *nicht* vom Barbier rasiert. Aber er selbst ist der Barbier. Dies ist ein Widerspruch!

Angenommen, der Barbier rasiert sich nicht selbst.

Da er in Sonnenthal wohnt und dort alle Einwohner rasiert, die sich nicht selbst rasieren, muss er sich rasieren. Dies ist ein Widerspruch!

Die Anfänge der formalen Logik

Aristoteles' Syllogismen

Die folgende Schlussweise ist aus rein formalen Gründen korrekt.

Annahme 1:	Alle Menschen sind sterblich.
Annahme 2:	Sokrates ist ein Mensch.
Folgerung:	Also ist Sokrates sterblich.

Diese Art von Schluss und eine Reihe verwandter Schlussweisen nennt man Syllogismen.

Annahme 1:	Alle A sind B.
Annahme 2:	C ist ein A.
Folgerung:	Also ist C B.

Beispiele

Annahme 1:	Alle Borg sind assimiliert worden.
Annahme 2:	Seven of Nine ist eine Borg.
Folgerung:	Also ist Seven of Nine assimiliert worden.

Annahme 1:	Alle Substitutionschiffren sind anfällig gegen Brute-Force-Angriffe.
Annahme 2:	Der Julius-Cäsar-Chiffre ist ein Substitutionschiffre.
Folgerung:	Also ist der Julius-Cäsar-Chiffre anfällig gegen Brute-Force-Angriffe.

Aristoteles (384 - 322 v.Chr.)
Quelle: `http://de.wikipedia.org/wiki/Aristoteles`

Ein komplizierterer formaler Schluss

Annahme 1:	<u>Es gibt keine</u> Schweine, <u>die</u> fliegen können.
Annahme 2:	<u>Alle</u> Schweine <u>sind</u> gefräßige Tiere.
Annahme 3:	<u>Es gibt</u> Schweine.
Folgerung:	<u>Also gibt es</u> gefräßige Tiere, <u>die nicht</u> fliegen können.

Die Form des Schlusses ist:

Annahme 1:	<u>Es gibt keine</u> A, <u>die</u> B (<u>sind</u>).
Annahme 2:	<u>Alle</u> A <u>sind</u> C.
Annahme 3:	<u>Es gibt</u> A.
Folgerung:	<u>Also gibt es</u> C, <u>die nicht</u> B (<u>sind</u>).

Charles Lutwidge Dodgson a.k.a. Lewis Carroll (1838 – 1898)
Quelle: `http://en.wikiquote.org/wiki/Lewis_Carroll`

"Contrariwise," continued Tweedledee, "if it was so, it might be; and if it were so, it would be; but as it isn't, it ain't. That's logic."

aus: *Alice in Wonderland*

Nicht jeder formale Schluss ist korrekt

Annahme 1:	Es gibt Vögel, die fliegen können.
Annahme 2:	Es gibt keine fliegenden (Tiere), die Klavier spielen können.
Folgerung:	Also gibt es keine Vögel, die Klavier spielen können.

Kein korrekter Schluss, auch wenn in diesem Fall die Folgerung wahr ist.

Der folgende, offensichtlich falsche, Schluss hat dieselbe Form:

Annahme 1:	Es gibt Menschen, die stumm sind.
Annahme 2:	Es gibt keine stummen (Lebewesen), die sprechen können.
Folgerung:	Also gibt es keine Menschen, die sprechen können.

Aber wie merkt man es?

Man kann einen falschen Schluss entlarven, indem man einen formal gleichen Schluss findet, der klar falsch ist.

Annahme 1:	Erbeeren schmecken gut.
Annahme 2:	Schlagsahne schmeckt gut.
Folgerung:	Also schmecken Erdbeeren mit Schlagsahne gut.

Aber:

Annahme 1:	Pizza schmeckt gut.
Annahme 2:	Schlagsahne schmeckt gut.
Folgerung:	Also schmeckt Pizza mit Schlagsahne gut.

Wasons Auswahlaufgabe (Wason's selection task)

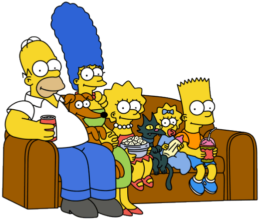
Uns stehen vier Karten der folgenden Art zur Verfügung:
Auf jeder Karte steht auf der Vorderseite eine Ziffer zwischen 0 und 9. Die Rückseite jeder Karte ist komplett rot oder komplett blau.
Wir sehen Folgendes:

Jemand hat folgende **Hypothese** aufgestellt:

> Wenn auf der Vorderseite eine gerade Zahl steht,
> dann ist die Rückseite rot.

Welche Karte(n) müssen Sie umdrehen, um zu überprüfen, ob die Hypothese stimmt?

Und was sagen die Simpsons?



Quelle: `http://en.wikipedia.org/wiki/Simpson_family`

Homer:	Not a bear in sight. The Bear Patrol must be working like a charm.
Lisa:	That's specious reasoning, Dad.
Homer:	Thank you, dear.
Lisa:	By your logic I could claim that this rock keeps tigers away.
Homer:	Oh, how does it work?
Lisa:	It doesn't work.
Homer:	Uh-huh.
Lisa:	It's just a stupid rock.
Homer:	Uh-huh.
Lisa:	But I don't see any tigers around, do you? *(Pause)*
Homer:	Lisa, I want to buy your rock.

[Lisa refuses at first, then takes the exchange]

Abschnitt 1.2:

Logik in der Informatik

Die Rolle der Logik in der Informatik

Halpern, Harper, Immerman, Kolaitis, Vardi, Vianu (2001):

> *Concepts and methods of logic occupy a central place in computer science, insomuch that logic has been called "the calculus of computer science".*

aus: *On the unusual effectiveness of logic in computer science*, Bulletin of Symbolic Logic 7(2): 213-236 (2001)

Anwendungsbereiche der Logik in der Informatik

- Repräsentation von Wissen (z.B. im Bereich der künstlichen Intelligenz) [siehe Kapitel 2 und 3]
- Grundlage für Datenbank-Anfragesprachen [siehe Kapitel 3]
- Bestandteil von Programmiersprachen (z.B. um Bedingungen in IF-Anweisungen zu formulieren) [siehe Kapitel 2]
- automatische Generierung von Beweisen (so genannte *Theorembeweiser*) [siehe Kapitel 4]
- Verifikation von
  - Schaltkreisen (*Ziel:* beweise, dass ein Schaltkreis bzw. Chip „richtig" funktioniert)
  - Programmen (*Ziel:* beweise, dass ein Programm gewisse wünschenswerte Eigenschaften hat)
  - Protokollen (*Ziel:* beweise, dass die Kommunikation zwischen zwei „Agenten", die nach einem gewissen Protokoll abläuft, „sicher" ist — etwa gegen Abhören oder Manipulation durch dritte; Anwendungsbeispiel: Internet-Banking)
- Logik-Programmierung [siehe folgende Folien und Kapitel 5]

Einführung in die Logik-Programmierung

„Was“ statt „Wie“ am Beispiel von Tiramisu

Tiramisu — Deklarativ

Aus Eigelb, Mascarpone und in Likör und Kaffee getränkten Biskuits hergestellte cremige Süßspeise

(aus: DUDEN, Fremdwörterbuch, 6. Auflage)

Tiramisu — Imperativ

1/4 l Milch mit 2 EL Kakao und 2 EL Zucker aufkochen. 1/4 l starken Kaffee und 4 EL Amaretto dazugeben.

5 Eigelb mit 75 g Zucker weißschaumig rühren, dann 500 g Mascarpone dazumischen.

ca 200 g Löffelbiskuit.

Eine Lage Löffelbiskuit in eine Auflaufform legen, mit der Flüssigkeit tränken und mit der Creme überziehen. Dann wieder Löffelbiskuit darauflegen, mit der restlichen Flüssigkeit tränken und mit der restlichen Creme überziehen.

Über Nacht im Kühlschrank durchziehen lassen und vor dem Servieren mit Kakao bestäuben.

(aus: Gisela Schweikardt, handschriftliche Kochrezepte)

Der große Traum der Informatik

Imperative Vorgehensweise:

Beschreibung, wie das gewünschte Ergebnis erzeugt wird . . . . „Wie“

Deklarative Vorgehensweise:

Beschreibung der Eigenschaften des gewünschten Ergebnisses . . . . „Was“

Traum der Informatik:

Möglichst wenig „wie“, möglichst viel „was“

D.h.: Automatische Generierung eines Ergebnisses aus seiner Spezifikation

Realität:

Datenbanken: Deklarative Anfragesprache ist Industriestandard (SQL)

Software-Entwicklung: Generierungs-Tools

Programmiersprachen: Logik-Programmierung, insbes. Prolog

ABER: Imperativer Ansatz überwiegt in der Praxis

Logik-Programmierung

- Logik-Programmierung bezeichnet die Idee, Logik direkt als Programmiersprache zu verwenden.

- Logik-Programmierung (in Sprachen wie Prolog) und die verwandte funktionale Programmierung (in Sprachen wie LISP, ML, Haskell) sind deklarativ,
  im Gegensatz zur imperativen Programmierung (in Sprachen wie Java, C, Perl).

- Die Idee er deklarativen Programmierung besteht darin, dem Computer lediglich sein Wissen über das Anwendungsszenario und sein Ziel mitzuteilen und dann die Lösung des Problems dem Computer zu überlassen.

  Bei der imperativen Programmierung hingegen gibt man dem Computer die einzelnen Schritte zur Lösung des Problems vor.

Prolog

- Prolog
  - ist die wichtigste logische Programmiersprache,
  - geht zurück auf Kowalski und Colmerauer (Anfang der 1970er Jahre, Marseilles),
  - steht für (franz.) Programmation en logique.
  - Mitte/Ende der 1970er Jahre: effiziente Prolog-Implementierung durch den von Warren (in Edinburgh) entwickelten Prolog-10 Compiler.

- Aus Effizienzgründen werden in Prolog die abstrakten Ideen der logischen Programmierung nicht in Reinform umgesetzt, Prolog hat auch „nichtlogische“ Elemente.

- Prolog ist eine voll entwickelte und mächtige Programmiersprache, die vor allem für symbolische Berechnungsprobleme geeignet ist.

Anwendungen

Die wichtigsten Anwendungsgebiete sind die künstliche Intelligenz und die Computerlinguistik.

Beispiele

Das Interface für natürliche Sprache

- in der International Space Station wurde von der NASA
- beim IBM Watson System, das in 2011 die Jeopardy! Man vs. Machine Challenge gewonnen hat, wurde

in Prolog implementiert.

Mehr Informationen dazu finden sich z.B. unter `https://sicstus.sics.se/customers.html` und `http://www.cs.nmsu.edu/ALP/2011/03/natural-language-processing-with-prolog-in-the-ibm-watson-system/`

Learn Prolog Now!

Im Rahmen der Übungsaufgaben zur Vorlesung werden wir jede Woche eins der 12 Kapitel des Buchs

> „Learn Prolog Now!“ von Patrick Blackburn, Johan Bos und Kristina Striegnitz (Kings College Publications, 2006)
>
> ... auch erhältlich als Online-Kurs unter `http://www.learnprolognow.org`

durcharbeiten.
Als Unterstützung dazu gibt es jede Woche eine 2-stündige Prolog-Übung.

Am Ende des Semesters, in Kapitel 5, werden wir von Prolog abstrahieren und uns die Grundprinzipien der Logik-Programmierung anschauen.

Kapitel 2:

Aussagenlogik

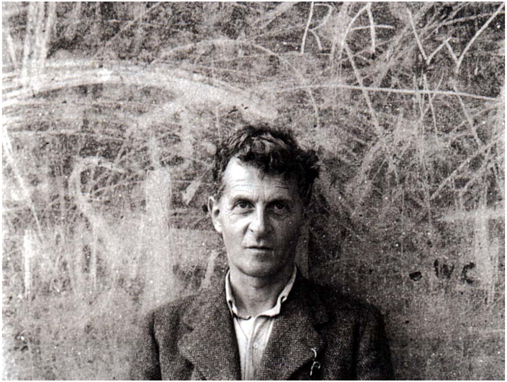
Abschnitt 2.1:

Syntax und Semantik

Aussagen

Die Frage „Was ist eigentlich ein Wort?“ ist analog der „Was ist eine Schachfigur?“ Ludwig Wittgenstein, *Philosophische Untersuchungen*

- Aussagen (im Sinne der Aussagenlogik) sind sprachliche Gebilde, die entweder wahr oder falsch sind.
- Aussagen können mit Junktoren wie nicht, und, oder oder wenn ... dann zu komplexeren Aussagen verknüpft werden.
- Aussagenlogik beschäftigt sich mit allgemeinen Prinzipien des korrekten Argumentierens und Schließens mit Aussagen und Kombinationen von Aussagen.



Ludwig Wittgenstein (1889 – 1951)
Quelle: `http://en.wikipedia.org/wiki/Ludwig_Wittgenstein`

Beispiel 2.1 (Geburtstagsfeier)

Fred möchte mit möglichst vielen seiner Freunde Anne, Bernd, Christine, Dirk und Eva seinen Geburtstag feiern. Er weiß Folgendes:
Wenn Bernd und Anne beide zur Party kommen, dann wird Eva auf keinen Fall kommen. Und Dirk wird auf keinen Fall kommen, wenn Bernd und Eva beide zur Feier kommen. Aber Eva kommt allenfalls dann, wenn Christine und Dirk kommen. Andererseits kommt Christine nur dann, wenn auch Anne kommt. Anne wiederum wird nur dann kommen, wenn auch Bernd oder Christine dabei sind.

Frage: Wie viele Freunde (und welche) werden im besten Fall zur Party kommen?

Das Wissen, das in dem Text wiedergegeben ist, lässt sich in „atomare Aussagen“ zerlegen, die mit Junktoren verknüpft werden können.
Die atomaren Aussagen, um die sich der Text dreht, kürzen wir folgendermaßen ab:

$$\begin{array}{lcl} A & : & \text{Anne kommt zur Feier} \\ B & : & \text{Bernd kommt zur Feier} \\ C & : & \text{Christine kommt zur Feier} \\ D & : & \text{Dirk kommt zur Feier} \\ E & : & \text{Eva kommt zur Feier} \end{array}$$

Das im Text zusammengefasste Wissen lässt sich wie folgt repräsentieren.

(1) Wenn Bernd und Anne beide zur Party kommen, dann wird Eva auf keinen Fall kommen.

kurz: Wenn (B und A), dann nicht E *kürzer:* $(B \wedge A) \rightarrow \neg E$

(2) Dirk wird auf keinen Fall kommen, wenn Bernd und Eva beide zur Feier kommen.

kurz: Wenn (B und E), dann nicht D *kürzer:* $(B \wedge E) \rightarrow \neg D$

(3) Eva kommt allenfalls dann, wenn Christine und Dirk kommen.

kurz: Wenn E, dann (C und D) *kürzer:* $E \rightarrow (C \wedge D)$

(4) Christine kommt nur dann, wenn auch Anne kommt.

kurz: Wenn C, dann A *kürzer:* $C \rightarrow A$

(5) Anne kommt nur dann, wenn auch Bernd oder Christine dabei sind.

kurz: Wenn A, dann (B oder C) *kürzer:* $A \rightarrow (B \vee C)$

Syntax und Semantik

Syntax: legt fest, welche Zeichenketten Formeln der Aussagenlogik sind

Semantik: legt fest, welche „Bedeutung“ einzelne Formeln haben

Dies ist analog zur Syntax und Semantik von Java-Programmen:

Die Syntax legt fest, welche Zeichenketten Java-Programme sind, während die Semantik bestimmt, was das Programm tut.

Zur Verdeutlichung werden wir im Folgenden syntaktische Objekte oft in orange darstellen, während wir semantische Aussagen in grün angeben.

Syntax der Aussagenlogik

Notationen

- Die Menge $\mathbb{N}$ der natürlichen Zahlen besteht aus allen nicht-negativen ganzen Zahlen, d.h.

$$\mathbb{N} := \{ 0, 1, 2, 3, \dots \}.$$

- Für ein $n \in \mathbb{N}$ ist

$$[n] := \{1, \dots, n\} = \{ i \in \mathbb{N} : 1 \leqslant i \leqslant n \}.$$

Definition 2.2

Ein Aussagensymbol (oder eine Aussagenvariable, kurz: Variable) hat die Form A_i für ein $i \in \mathbb{N}$.
Die Menge aller Aussagensymbole bezeichnen wir mit AS, d.h.

$$\text{AS} = \{A_i : i \in \mathbb{N}\} = \{A_0, A_1, A_2, A_3, \ldots\}$$

Aussagenlogische Formeln sind Wörter, die über dem folgenden Alphabet gebildet sind.

Definition 2.3

Das Alphabet der Aussagenlogik besteht aus

- den Aussagesymbolen in AS,
- den Junktoren $\neg$, $\wedge$, $\vee$, $\rightarrow$,
- den booleschen Konstanten $\mathbf{0}$, $\mathbf{1}$,
- den Klammersymbolen (,).

Wir schreiben A_{AL}, um das Alphabet der Aussagenlogik zu bezeichnen, d.h.

$$A_{\text{AL}} := \text{AS} \cup \{\neg, \wedge, \vee, \rightarrow, \mathbf{0}, \mathbf{1}, (,)\}$$

Definition 2.4 (Syntax der Aussagenlogik)

Die Menge AL der aussagenlogischen Formeln (kurz: Formeln) ist die folgendermaßen rekursiv definierte Teilmenge von A_{AL}^*:

Basisregeln: (zum Bilden der so genannten atomaren Formeln)

(B0) $\mathbf{0} \in \mathsf{AL}$

(B1) $\mathbf{1} \in \mathsf{AL}$

(BS) Für jedes Aussagensymbol $A_i \in \mathsf{AS}$ gilt: $A_i \in \mathsf{AL}$

Rekursive Regeln:

(R1) Ist $\varphi \in \mathsf{AL}$, so ist auch $\neg\varphi \in \mathsf{AL}$ (Negation)

(R2) Ist $\varphi \in \mathsf{AL}$ und $\psi \in \mathsf{AL}$, so ist auch

- $(\varphi \wedge \psi) \in \mathsf{AL}$ (Konjunktion)
- $(\varphi \vee \psi) \in \mathsf{AL}$ (Disjunktion)
- $(\varphi \to \psi) \in \mathsf{AL}$ (Implikation)

Beispiele

- $(\neg A_0 \vee (A_0 \to A_1))$ $\in$ AL
- $\neg((A_0 \wedge \mathbf{0}) \to \neg A_3)$ $\in$ AL
- $A_1 \vee A_2 \wedge A_3$ $\notin$ AL
- $(\neg A_1)$ $\notin$ AL

Griechische Buchstaben

In der Literatur werden Formeln einer Logik traditionell meistens mit griechischen Buchstaben bezeichnet.

Hier eine Liste der gebräuchlichsten Buchstaben:

Buchstabe	φ	ψ	χ	θ bzw. ϑ	λ	μ	ν	τ	κ
Aussprache	phi	psi	chi	theta	lambda	mü	nü	tau	kappa

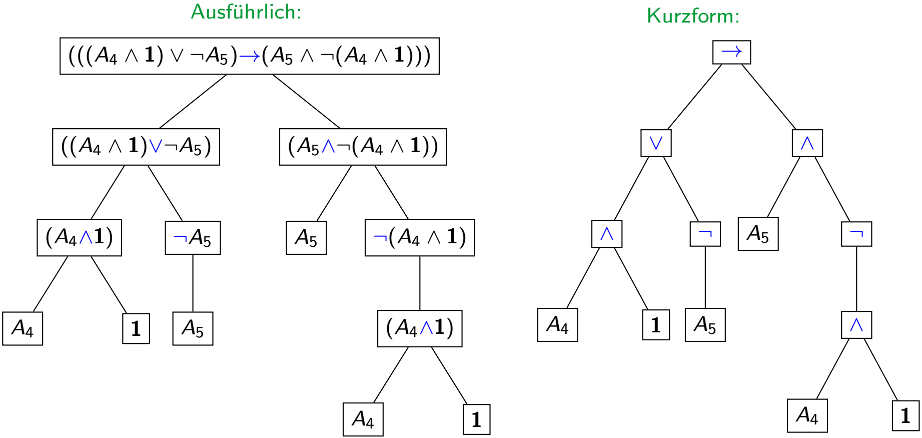
Buchstabe	σ	ρ	ξ	ζ	α	β	γ	δ	ω
Aussprache	sigma	rho	xi	zeta	alpha	beta	gamma	delta	omega

Buchstabe	ε	ι	π	Δ	Γ	Σ	Π	Φ
Aussprache	epsilon	iota	pi	Delta	Gamma	Sigma	Pi	Phi

Syntaxbäume

Die Struktur einer Formel lässt sich bequem in einem Syntaxbaum (englisch: parse tree) darstellen.

Beispiel: Syntaxbaum der Formel $(((A_4 \wedge \mathbf{1}) \vee \neg A_5) \rightarrow (A_5 \wedge \neg(A_4 \wedge \mathbf{1})))$



Subformeln und eindeutige Lesbarkeit

- Jede Formel hat genau einen Syntaxbaum. Diese Aussage ist als das Lemma über die eindeutige Lesbarkeit aussagenlogischer Formeln bekannt.

- Die Formeln ψ, die im ausführlichen Syntaxbaum einer Formel φ als Knotenbeschriftung vorkommen, nennen wir Subformeln (bzw. Teilformeln) von φ.

- Eine Subformel ψ von φ kann an mehreren Knoten des Syntaxbaums vorkommen. Wir sprechen dann von verschiedenen Vorkommen von ψ in φ.

Semantik der Aussagenlogik

Vorüberlegung zur Semantik

- Eine aussagenlogische Formel wird erst zur Aussage, wenn wir alle in ihr vorkommenden Aussagensymbole durch Aussagen ersetzen.
- Wir interessieren uns hier nicht so sehr für die tatsächlichen Aussagen, sondern nur für ihren **Wahrheitswert**, also dafür, ob sie wahr oder falsch sind.
- Um das festzustellen, reicht es, den Aussagensymbolen die Wahrheitswerte der durch sie repräsentierten Aussagen zuzuordnen.
- Die Bedeutung einer Formel besteht also aus ihren Wahrheitswerten unter allen möglichen Wahrheitswerten für die in der Formel vorkommenden Aussagensymbole.

Interpretationen (d.h. Variablenbelegungen)

Wir repräsentieren die Wahrheitswerte wahr und falsch durch 1 und 0.

Definition 2.5

Eine aussagenlogische Interpretation (kurz: Interpretation oder Belegung) ist eine Abbildung

$$\mathcal{I} : \mathsf{AS} \to \{0, 1\}.$$

D.h.: $\mathcal{I}$ „belegt" jedes Aussagensymbol $X \in \mathsf{AS}$ mit einem der beiden Wahrheitswerte 1 (für „wahr") oder 0 (für „falsch"); und $\mathcal{I}(X)$ ist der Wahrheitswert, mit dem das Aussagensymbol X belegt wird.

Semantik der Aussagenlogik

Definition 2.6

Zu jeder Formel $\varphi \in \mathrm{AL}$ und jeder Interpretation $\mathcal{I}$ definieren wir einen Wahrheitswert $[\![\varphi]\!]^{\mathcal{I}}$ rekursiv wie folgt:

Rekursionsanfang:

- $[\![\mathbf{0}]\!]^{\mathcal{I}} := 0$.
- $[\![\mathbf{1}]\!]^{\mathcal{I}} := 1$.
- Für alle $X \in \mathrm{AS}$ gilt: $[\![X]\!]^{\mathcal{I}} := \mathcal{I}(X)$.

Rekursionsschritt:

- Ist $\varphi \in \mathrm{AL}$, so ist $[\![\neg\varphi]\!]^{\mathcal{I}} := \begin{cases} 1 & \text{falls } [\![\varphi]\!]^{\mathcal{I}} = 0, \\ 0 & \text{sonst.} \end{cases}$

Semantik der Aussagenlogik (Fortsetzung)

- Ist $\varphi \in$ AL und $\psi \in$ AL, so ist
  - $[\![(\varphi \wedge \psi)]\!]^{\mathcal{I}} := \begin{cases} 1 & \text{falls } [\![\varphi]\!]^{\mathcal{I}} = [\![\psi]\!]^{\mathcal{I}} = 1, \\ 0 & \text{sonst.} \end{cases}$
  - $[\![(\varphi \vee \psi)]\!]^{\mathcal{I}} := \begin{cases} 0 & \text{falls } [\![\varphi]\!]^{\mathcal{I}} = [\![\psi]\!]^{\mathcal{I}} = 0, \\ 1 & \text{sonst.} \end{cases}$
  - $[\![(\varphi \to \psi)]\!]^{\mathcal{I}} := \begin{cases} 0 & \text{falls } [\![\varphi]\!]^{\mathcal{I}} = 1 \text{ und } [\![\psi]\!]^{\mathcal{I}} = 0, \\ 1 & \text{sonst.} \end{cases}$

Intuitive Bedeutung der Semantik

Boolesche Konstanten: $\mathbf{1}$ und $\mathbf{0}$ bedeuten einfach „wahr“ und „falsch“.

Aussagensymbole: Die Aussagensymbole stehen für irgendwelche Aussagen, von denen uns aber nur der Wahrheitswert interessiert. Dieser wird durch die Interpretation festgelegt.

Negation: $\neg\varphi$ bedeutet „nicht φ“.

Konjunktion: $(\varphi \wedge \psi)$ bedeutet „φ und ψ“.

Disjunktion: $(\varphi \vee \psi)$ bedeutet „φ oder ψ“.

Implikation: $(\varphi \to \psi)$ bedeutet „φ impliziert ψ“ (oder „wenn φ dann ψ“).

Rekursive Definitionen über Formeln

- Ähnlich wie Funktionen auf den natürlichen Zahlen, wie zum Beispiel die Fakultätsfunktion oder die Fibonacci Folge, können wir Funktionen auf den aussagenlogischen Formeln rekursiv definieren.

- Dabei gehen wir von den atomaren Formeln aus und definieren dann den Funktionswert einer zusammengesetzten Formel aus den Funktionswerten ihrer Bestandteile.

- Zur Rechtfertigung solcher Definitionen benötigt man die eindeutige Lesbarkeit aussagenlogischer Formeln, die besagt, dass sich jede Formel eindeutig in ihre Bestandteile zerlegen lässt.

- Wir haben auf diese Weise die Semantik definiert. Wir haben nämlich für jede Interpretation $\mathcal{I}$ rekursiv eine Funktion $[\![\, \cdot \,]\!]^{\mathcal{I}} : \mathsf{AL} \to \{0,1\}$ definiert.

Rekursive Definitionen (Forts.)

Schematisch sieht die rekursive Definition einer Funktion $f : \mathsf{AL} \to M$ (für eine beliebige Menge M) folgendermaßen aus:

Rekursionsanfang:

- Definiere $f(\mathbf{0})$ und $f(\mathbf{1})$.
- Definiere $f(X)$ für alle $X \in \mathsf{AS}$.

Rekursionsschritt:

- Definiere $f(\neg\varphi)$ aus $f(\varphi)$.
- Definiere $f((\varphi \wedge \psi))$ aus $f(\varphi)$ und $f(\psi)$.
- Definiere $f((\varphi \vee \psi))$ aus $f(\varphi)$ und $f(\psi)$.
- Definiere $f((\varphi \to \psi))$ aus $f(\varphi)$ und $f(\psi)$.

Beispiel 2.7

Betrachte die Formel $\varphi \ := \ \big(\, \neg A_0 \ \vee \ (A_5 \to A_1)\,\big)$

und die Interpretation $\mathcal{I} : \mathsf{AS} \to \{0,1\}$ mit

$$\mathcal{I}(A_0) = 1, \quad \mathcal{I}(A_1) = 1, \quad \mathcal{I}(A_5) = 0$$

und $\mathcal{I}(Y) = 0$ für alle $Y \in \mathsf{AS} \setminus \{A_0, A_1, A_5\}$.

Der Wahrheitswert $[\![\varphi]\!]^{\mathcal{I}}$ ist der Wert

$$
\begin{aligned}
[\![\varphi]\!]^{\mathcal{I}} &\overset{\text{Def. 2.6}}{=} \begin{cases} 0, & \text{falls } [\![\neg A_0]\!]^{\mathcal{I}} = 0 \text{ und } [\![(A_5 \to A_1)]\!]^{\mathcal{I}} = 0 \\ 1, & \text{sonst} \end{cases} \\
&\overset{\text{Def. 2.6}}{=} \begin{cases} 0, & \text{falls } [\![A_0]\!]^{\mathcal{I}} = 1 \text{ und } \big([\![A_5]\!]^{\mathcal{I}} = 1 \text{ und } [\![A_1]\!]^{\mathcal{I}} = 0\big) \\ 1, & \text{sonst} \end{cases} \\
&\overset{\text{Def. 2.6}}{=} \begin{cases} 0, & \text{falls } \mathcal{I}(A_0) = 1 \text{ und } \mathcal{I}(A_5) = 1 \text{ und } \mathcal{I}(A_1) = 0 \\ 1, & \text{sonst} \end{cases} \\
&= \quad 1 \quad \text{(denn gemäß obiger Wahl von } \mathcal{I} \text{ gilt } \mathcal{I}(A_5) = 0\text{).}
\end{aligned}
$$

Alternative Art, den Wert $\llbracket\varphi\rrbracket^{\mathcal{I}}$ zu bestimmen

- Ersetze in φ jedes Aussagensymbol X durch seinen gemäß $\mathcal{I}$ festgelegten Wahrheitswert, d.h. durch den Wert $\mathcal{I}(X)$, und rechne dann den Wert des resultierenden booleschen Ausdrucks aus.

- Speziell für die Formel φ und die Interpretation $\mathcal{I}$ aus Beispiel 2.7 ergibt die Ersetzung der Aussagensymbole durch die gemäß $\mathcal{I}$ festgelegten Wahrheitswerte den booleschen Ausdruck

$$\big(\neg 1 \;\vee\; (0 \to 1)\big).$$

- Ausrechnen von $\neg 1$ ergibt den Wert 0.
  Ausrechnen von $(0 \to 1)$ ergibt den Wert 1.

- Insgesamt erhalten wir also $(0 \vee 1)$, was sich zum Wert 1 errechnet.
  Somit erhalten wir, dass $\llbracket\varphi\rrbracket^{\mathcal{I}} = 1$ ist.

Die Modellbeziehung

Definition 2.8

(a) Eine Interpretation $\mathcal{I}$ erfüllt eine Formel $\varphi \in \mathsf{AL}$ (wir schreiben: $\mathcal{I} \models \varphi$), wenn $[\![\varphi]\!]^{\mathcal{I}} = 1$.

Wir schreiben kurz $\mathcal{I} \not\models \varphi$ um auszudrücken, dass $\mathcal{I}$ die Formel φ *nicht* erfüllt (d.h., es gilt $[\![\varphi]\!]^{\mathcal{I}} = 0$).

(b) Eine Interpretation $\mathcal{I}$ erfüllt eine Formelmenge $\Phi \subseteq \mathsf{AL}$ (wir schreiben: $\mathcal{I} \models \Phi$), wenn $\mathcal{I} \models \varphi$ für alle $\varphi \in \Phi$.

(c) Ein Modell einer Formel φ (bzw. einer Formelmenge Φ) ist eine Interpretation $\mathcal{I}$ mit $\mathcal{I} \models \varphi$ (bzw. $\mathcal{I} \models \Phi$).

Das Koinzidenzlemma

- Offensichtlich hängt der Wert $[\![\varphi]\!]^{\mathcal{I}}$ nur von den Werten $\mathcal{I}(X)$ der Aussagensymbole $X \in \mathsf{AS}$ ab, die auch in φ vorkommen.

  Diese Aussage ist als das Koinzidenzlemma der Aussagenlogik bekannt.

- Um $[\![\varphi]\!]^{\mathcal{I}}$ festzulegen, reicht es also, die Werte $\mathcal{I}(X)$ nur für diejenigen Aussagensymbole $X \in \mathsf{AS}$ anzugeben, die in φ vorkommen.

Vereinbarungen zu Interpretationen

- Statt der vollen Interpretation $\mathcal{I} : \mathsf{AS} \to \{0,1\}$ geben wir in der Regel nur endlich viele Werte $\mathcal{I}(X_1), \ldots, \mathcal{I}(X_n)$ an und legen fest, dass $\mathcal{I}(Y) := 0$ für alle $Y \in \mathsf{AS} \setminus \{X_1, \ldots, X_n\}$.

- In den Beispielen legen wir eine Interpretation oft durch eine Wertetabelle fest. Beispielsweise beschreibt die Tabelle

X	A_0	A_1	A_5
$\mathcal{I}(X)$	1	1	0

  die Interpretation $\mathcal{I}$ mit $\mathcal{I}(A_0) = \mathcal{I}(A_1) = 1$ und $\mathcal{I}(A_5) = 0$ und $\mathcal{I}(Y) = 0$ für alle $Y \in \mathsf{AS} \setminus \{A_0, A_1, A_5\}$.

- Wir schreiben $\varphi(X_1, \ldots, X_n)$, um anzudeuten, dass in φ nur Aussagensymbole aus der Menge $\{X_1, \ldots, X_n\}$ vorkommen.

  Für Wahrheitswerte $b_1, \ldots, b_n \in \{0,1\}$ schreiben wir dann $\varphi[b_1, \ldots, b_n]$ anstatt $[\![\varphi]\!]^{\mathcal{I}}$ für eine (bzw. alle) Interpretationen $\mathcal{I}$ mit $\mathcal{I}(X_i) = b_i$ für alle $i \in [n] := \{1, \ldots, n\}$.

Vereinbarungen

- Wir schreiben $(\varphi \leftrightarrow \psi)$ als Abkürzung für $((\varphi \to \psi) \wedge (\psi \to \varphi))$.
- Statt mit $A_0, A_1, A_2, \ldots$ bezeichnen wir Aussagensymbole auch oft mit $A, B, C, \ldots, X, Y, Z, \ldots$ oder mit Varianten wie $X', Y_1, \ldots$.
- Die äußeren Klammern einer Formel lassen wir manchmal weg und schreiben z.B. $(X \wedge Y) \to Z$ an Stelle des (formal korrekten) $((X \wedge Y) \to Z)$.
- Bezüglich Klammerung vereinbaren wir, dass $\neg$ am stärksten bindet, und dass $\wedge$ und $\vee$ stärker binden als $\to$.

  Wir können also z.B. $X \wedge \neg Y \;\to\; Z \vee X$ schreiben und meinen damit $((X \wedge \neg Y) \to (Z \vee X))$.

  <u>Nicht</u> schreiben können wir z.B. $X \wedge Y \vee Z$ (da wir nichts darüber vereinbart haben, wie fehlende Klammern hier zu setzen sind).

- Wir schreiben $\bigwedge_{i=1}^{n} \varphi_i$ bzw. $(\varphi_1 \wedge \ldots \wedge \varphi_n)$ an Stelle von

$$(\cdots((\varphi_1 \wedge \varphi_2) \wedge \varphi_3) \wedge \ldots \wedge \varphi_n)$$

  und nutzen analoge Schreibweisen auch für „$\vee$“ an Stelle von „$\wedge$“.

- Ist M eine endliche Menge aussagenlogischer Formeln, so schreiben wir

$$\bigwedge_{\varphi \in M} \varphi$$

  um die Formel $(\varphi_1 \wedge \cdots \wedge \varphi_n)$ zu bezeichnen, wobei $n = |M|$ die Anzahl der Formeln in M ist und $\varphi_1, \ldots, \varphi_n$ die Auflistung aller Formeln in M in lexikographischer Reihenfolge ist. Formeln sind hierbei Worte über dem Alphabet der Aussagenlogik, wobei die einzelnen Symbole dieses Alphabets folgendermaßen aufsteigend sortiert sind:

$$\mathbf{0},\ \mathbf{1},\ \neg,\ \wedge,\ \vee,\ \rightarrow,\ (,\ ),\ A_0,\ A_1,\ A_2,\ A_3,\ \ldots .$$

  Die analoge Schreibweise nutzen wir auch für „$\vee$“ an Stelle von „$\wedge$“.

- Diese Schreibweisen werden wir manchmal auch kombinieren. Sind zum Beispiel $I = \{i_1, \ldots, i_m\}$ und $J = \{j_1, \ldots, j_n\}$ endliche Mengen und ist für jedes $i \in I$ und $j \in J$ eine Formel $\varphi_{i,j}$ gegeben, so schreiben wir

$$\bigwedge_{i \in I} \bigvee_{j \in J} \varphi_{i,j}$$

um die Formel $(\psi_{i_1} \wedge \cdots \wedge \psi_{i_m})$ zu bezeichnen, wobei für jedes $i \in I$ die Formel ψ_i durch $\psi_i := \left(\varphi_{i,j_1} \vee \cdots \vee \varphi_{i,j_n} \right)$ definiert ist.

Wahrheitstafeln

Für jede Formel $\varphi(X_1, \dots, X_n)$ kann man die Wahrheitswerte unter allen möglichen Interpretationen in einer Wahrheitstafel darstellen. Für alle $(b_1, \dots, b_n) \in \{0,1\}^n$ enthält die Tafel eine Zeile mit den Werten
$b_1 \ \cdots \ b_n \ \mid \ \varphi[b_1, \dots, b_n]$.
Um die Wahrheitstafel für φ auszufüllen, ist es bequem, auch Spalten für (alle oder einige) Subformeln von φ einzufügen.

Beispiel: Wahrheitstafel für die Formel $\varphi(X,Y,Z) := X \vee Y \to X \wedge Z$:

X	Y	Z	$X \vee Y$	$X \wedge Z$	φ
0	0	0	0	0	1
0	0	1	0	0	1
0	1	0	1	0	0
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	0	0
1	1	1	1	1	1

Die Reihenfolge der Zeilen ist dabei unerheblich. Wir vereinbaren allerdings, die Zeilen stets so anzuordnen, dass die Werte $b_1 \cdots b_n \in \{0,1\}^n$, aufgefasst als Binärzahlen, in aufsteigender Reihenfolge aufgelistet werden.

Wahrheitstafeln für die Junktoren

Die Bedeutung der Junktoren kann man mittels ihrer Wahrheitstafeln beschreiben:

X	$\neg X$
0	1
1	0

X	Y	$X \wedge Y$
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	$X \vee Y$
0	0	0
0	1	1
1	0	1
1	1	1

X	Y	$X \rightarrow Y$
0	0	1
0	1	1
1	0	0
1	1	1

Genauso kann man eine Wahrheitstafel für die Formel $X \leftrightarrow Y$, die ja eine Abkürzung für $(X \rightarrow Y) \wedge (Y \rightarrow X)$ ist, bestimmen:

X	Y	$X \leftrightarrow Y$
0	0	1
0	1	0
1	0	0
1	1	1

$X \leftrightarrow Y$ bedeutet also „X genau dann wenn Y“.

Ein Logikrätsel

Beispiel 2.9

Auf der Insel Wafa leben zwei Stämme: Die Was, die immer die Wahrheit sagen, und die Fas, die immer lügen. Ein Reisender besucht die Insel und trifft auf drei Einwohner A, B, C, die ihm Folgendes erzählen:

- A sagt:
  „B und C sagen genau dann die Wahrheit, wenn C die Wahrheit sagt."

- B sagt:
  „Wenn A und C die Wahrheit sagen, dann ist es nicht der Fall, dass A die Wahrheit sagt, wenn B und C die Wahrheit sagen."

- C sagt:
  „B lügt genau dann, wenn A oder B die Wahrheit sagen."

Frage: Welchen Stämmen gehören A, B und C an?

Aussagenlogische Modellierung

Aussagensymbole:

- W_A steht für „A sagt die Wahrheit."
- W_B steht für „B sagt die Wahrheit."
- W_C steht für „C sagt die Wahrheit."

Aussagen der drei Inselbewohner:

- $\varphi_A := \ (W_B \wedge W_C) \ \leftrightarrow \ W_C$
- $\varphi_B := \ (W_A \wedge W_C) \ \rightarrow \ \neg\big((W_B \wedge W_C) \rightarrow \ W_A \big)$
- $\varphi_C := \ \neg W_B \ \leftrightarrow \ (W_A \vee W_B)$

Wir suchen nach einer Interpretation, die die Formel

$$\psi \ := \ (W_A \leftrightarrow \varphi_A) \ \wedge \ (W_B \leftrightarrow \varphi_B) \ \wedge \ (W_C \leftrightarrow \varphi_C)$$

erfüllt.

Lösung mittels Wahrheitstafel

W_A	W_B	W_C	φ_A	φ_B	φ_C	$W_A \leftrightarrow \varphi_A$	$W_B \leftrightarrow \varphi_B$	$W_C \leftrightarrow \varphi_C$	ψ
0	0	0	1	1	0	0	0	1	0
0	0	1	0	1	0	1	0	0	0
0	1	0	1	1	0	0	1	1	0
0	1	1	1	1	0	0	1	0	0
1	0	0	1	1	1	1	0	0	0
1	0	1	0	0	1	0	1	1	0
1	1	0	1	1	0	1	1	1	1
1	1	1	1	0	0	1	0	0	0

Die Interpretation $\mathcal{I}$ mit $\mathcal{I}(W_A) = 1$, $\mathcal{I}(W_B) = 1$, $\mathcal{I}(W_C) = 0$ in Zeile 7 ist die einzige, die die Formel ψ erfüllt.

Gemäß dieser Interpretation sind die Aussagen, die durch die Symbole W_A und W_B repräsentiert werden, wahr, während die Aussage, die durch W_C repräsentiert wird, falsch ist.

Das heißt, die Personen A und B sagen die Wahrheit und sind somit Was, und Person C lügt und ist daher ein Fa.

Computerlesbare Darstellung von Formeln

Wir betrachten das Alphabet $\mathtt{ASCII}$ aller ASCII-Symbole.

Die Menge $\mathsf{AS}_{\mathtt{ASCII}}$ aller ASCII-Repräsentationen von Aussagensymbolen besteht aus allen nicht-leeren Worten über dem Alphabet $\mathtt{ASCII}$, deren erstes Symbol ein Buchstabe ist, und bei dem alle weiteren Symbole Buchstaben oder Ziffern sind.

Die Menge $\mathsf{AL}_{\mathtt{ASCII}}$ aller ASCII-Repräsentationen von aussagenlogischen Formeln ist die rekursiv wie folgt definierte Teilmenge von $\mathtt{ASCII}^*$:

Basisregeln:

- $\mathtt{0} \in \mathsf{AL}_{\mathtt{ASCII}}$, $\mathtt{1} \in \mathsf{AL}_{\mathtt{ASCII}}$ und $w \in \mathsf{AL}_{\mathtt{ASCII}}$ für alle $w \in \mathsf{AS}_{\mathtt{ASCII}}$.

Rekursive Regeln:

- Ist $\varphi \in \mathsf{AL}_{\mathtt{ASCII}}$, so ist auch $\mathtt{\sim}\varphi \in \mathsf{AL}_{\mathtt{ASCII}}$. (Negation)
- Ist $\varphi \in \mathsf{AL}_{\mathtt{ASCII}}$ und $\psi \in \mathsf{AL}_{\mathtt{ASCII}}$, so ist auch
  - $(\varphi \mathtt{/\backslash} \psi) \in \mathsf{AL}_{\mathtt{ASCII}}$ (Konjunktion)
  - $(\varphi \mathtt{\backslash/} \psi) \in \mathsf{AL}_{\mathtt{ASCII}}$ (Disjunktion)
  - $(\varphi \mathtt{->} \psi) \in \mathsf{AL}_{\mathtt{ASCII}}$ (Implikation)
  - $(\varphi \mathtt{<->} \psi) \in \mathsf{AL}_{\mathtt{ASCII}}$ (Biimplikation).

Abstrakte vs. computerlesbare Syntax

Es ist offensichtlich, wie man Formeln aus AL in ihre entsprechende ASCII-Repräsentation übersetzt und umgekehrt. Zum Beispiel ist

$$\big((A_0 \wedge \mathbf{0}) \to \neg A_{13}\big)$$

eine Formel in AL, deren ASCII-Repräsentation die folgende Zeichenkette aus $\text{AL}_{\texttt{ASCII}}$ ist:

```
( (A0 /\ 0) -> ~A13 ).
```

Wir werden meistens mit der „abstrakten Syntax“, d.h. mit der in Definition 2.4 festgelegten Menge AL, arbeiten. Um aber Formeln in Computer-Programme einzugeben, können wir die ASCII-Repräsentation verwenden.

Demo: `snippets of logic`

- ein Formelchecker für die Aussagenlogik
- entwickelt von André Frochaux
- Funktionalitäten u.a.:
  - Syntaxcheck für eingegebene Formeln
  - Ausgabe eines Syntaxbaums
  - Ausgabe einer Wahrheitstafel
- Zugänglich via

  `http://www.snippets-of-logic.net/index_AL.php?lang=de`

Zurück zu Beispiel 2.1 („Geburtstagsfeier“)

Das in Beispiel 2.1 aufgelistete Wissen kann durch folgende aussagenlogische Formel repräsentiert werden:

$$\begin{aligned}\varphi \; := \quad & \big((B \wedge A) \to \neg E\big) \;\wedge\; \big((B \wedge E) \to \neg D\big) \;\wedge \\ & \big(E \to (C \wedge D)\big) \;\wedge\; \big(C \to A\big) \;\wedge\; \big(A \to (B \vee C)\big)\end{aligned}$$

Die Frage

> *„Wie viele (und welche) Freunde werden im besten Fall zur Party kommen?“*

kann nun durch Lösen der folgenden Aufgabe beantwortet werden:

Finde eine Interpretation $\mathcal{I}$ für φ, so dass gilt:

- $\mathcal{I} \models \varphi$ (d.h., $\mathcal{I}$ ist ein Modell von φ) und
- $\big|\{X \in \{A, B, C, D, E\} \;:\; \mathcal{I}(X) = 1\}\big|$ ist so groß wie möglich.

Diese Frage können wir lösen, indem wir

(1) die Wahrheitstafel für φ ermitteln,

(2) alle Zeilen raussuchen, in denen in der mit „φ“ beschrifteten Spalte der Wert 1 steht (das liefert uns genau die Modelle von φ) und

(3) aus diesen Zeilen all jene raussuchen, bei denen in den mit A, B, C, D, E beschrifteten Spalten möglichst viele Einsen stehen. Jede dieser Zeilen repräsentiert dann eine größtmögliche Konstellation von gleichzeitigen Partybesuchern.

Prinzipiell führt diese Vorgehensweise zum Ziel.

Leider ist das Verfahren aber recht aufwändig, da die Wahrheitstafel, die man dabei aufstellen muss, sehr groß wird: Sie hat $2^5 = 32$ Zeilen.

A	B	C	D	E	$E \to (C \wedge D)$	$C \to A$	$(B \wedge E) \to \neg D$	$A \to (B \vee C)$	$(B \wedge A) \to \neg E$	φ
0	0	0	0	0	1	1	1	1	1	1
0	0	0	0	1	0	1	1	1	1	0
0	0	0	1	0	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	0
0	0	1	0	0	1	0	1	1	1	0
0	0	1	0	1	0	0	1	1	1	0
0	0	1	1	0	1	0	1	1	1	0
0	0	1	1	1	1	0	1	1	1	0
0	1	0	0	0	1	1	1	1	1	1
0	1	0	0	1	0	1	1	1	1	0
0	1	0	1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	0	1	1	0
0	1	1	0	0	1	0	1	1	1	0
0	1	1	0	1	0	0	1	1	1	0
0	1	1	1	0	1	0	1	1	1	0
0	1	1	1	1	1	0	0	1	1	0
1	0	0	0	0	1	1	1	0	1	0
1	0	0	0	1	0	1	1	0	1	0
1	0	0	1	0	1	1	1	0	1	0
1	0	0	1	1	0	1	1	0	1	0
1	0	1	0	0	1	1	1	1	1	1
1	0	1	0	1	0	1	1	1	1	0
1	0	1	1	0	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1
1	1	0	0	0	1	1	1	1	1	1
1	1	0	0	1	0	1	1	1	0	0
1	1	0	1	0	1	1	1	1	1	1
1	1	0	1	1	0	1	0	1	0	0
1	1	1	0	0	1	1	1	1	1	1
1	1	1	0	1	0	1	1	1	0	0
1	1	1	1	0	1	1	1	1	1	1
1	1	1	1	1	1	1	0	1	0	0

Modelle für φ werden hier durch grau unterlegte Zeilen repräsentiert.

In der Wahrheitstafel sieht man:

- Es gibt kein Modell für φ, bei dem in den mit A bis E beschrifteten Spalten insgesamt 5 Einsen stehen.
- Es gibt genau zwei Modelle für φ, bei denen in den mit A bis E beschrifteten Spalten insgesamt 4 Einsen stehen, nämlich die beiden Interpretationen $\mathcal{I}_1$ und $\mathcal{I}_2$ mit

$$\mathcal{I}_1(A) = \mathcal{I}_1(C) = \mathcal{I}_1(D) = \mathcal{I}_1(E) = 1 \;\text{ und }\; \mathcal{I}_1(B) = 0$$

und

$$\mathcal{I}_2(A) = \mathcal{I}_2(B) = \mathcal{I}_2(C) = \mathcal{I}_2(D) = 1 \;\text{ und }\; \mathcal{I}_2(E) = 0.$$

Die Antwort auf die Frage *„Wie viele (und welche) Freunde werden bestenfalls zur Party kommen?“* lautet also:

Bestenfalls werden 4 der 5 Freunde kommen, und dafür gibt es zwei Möglichkeiten, nämlich

(1) dass alle außer Bernd kommen, und

(2) dass alle außer Eva kommen.

Erfüllbarkeit, Allgemeingültigkeit und die Folgerungsbeziehung

Erfüllbarkeit

Definition 2.10

Eine Formel $\varphi \in \mathsf{AL}$ heißt erfüllbar, wenn es eine Interpretation gibt, die φ erfüllt.

Eine Formelmenge Φ heißt erfüllbar, wenn es eine Interpretation $\mathcal{I}$ gibt, die Φ erfüllt (d.h. es gilt $\mathcal{I} \models \varphi$ für jedes $\varphi \in \Phi$).

Eine Formel oder Formelmenge, die nicht erfüllbar ist, nennen wir unerfüllbar.

Beobachtung 2.11

(a) *Eine aussagenlogische Formel ist genau dann erfüllbar, wenn in der letzten Spalte ihrer Wahrheitstafel mindestens eine* 1 *steht.*

(b) *Eine endliche Formelmenge* $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ *ist genau dann erfüllbar, wenn die Formel* $\bigwedge_{i=1}^{n} \varphi_i$ *erfüllbar ist.*

Beispiele:

- Die Formel X ist erfüllbar.
- Die Formel $(X \wedge \neg X)$ ist unerfüllbar.

Allgemeingültigkeit

Definition 2.12

Eine Formel $\varphi \in \mathsf{AL}$ ist allgemeingültig, wenn *jede* Interpretation $\mathcal{I}$ die Formel φ erfüllt.

Bemerkung

Allgemeingültige Formeln nennt man auch Tautologien.
Man schreibt auch $\models \varphi$ um auszudrücken, dass φ allgemeingültig ist.

Beobachtung 2.13

Eine aussagenlogische Formel ist genau dann allgemeingültig, wenn in der letzten Spalte ihrer Wahrheitstafel nur 1*en stehen.*

Beispiel: Die Formel $(X \lor \neg X)$ ist allgemeingültig.

Beispiel 2.14

Die Formel $(X \vee Y) \wedge (\neg X \vee Y)$ ist

- erfüllbar, da z.B. die Interpretation $\mathcal{I}$ mit $\mathcal{I}(X) = 0$ und $\mathcal{I}(Y) = 1$ die Formel erfüllt.
- nicht allgemeingültig, da z.B. die Interpretation $\mathcal{I}'$ mit $\mathcal{I}'(X) = 0$ und $\mathcal{I}'(Y) = 0$ die Formel nicht erfüllt.

Die Folgerungsbeziehung

Definition 2.15

Eine Formel $\psi \in \mathsf{AL}$ folgt aus einer Formelmenge $\Phi \subseteq \mathsf{AL}$ (wir schreiben: $\Phi \models \psi$), wenn für jede Interpretation $\mathcal{I}$ gilt: Wenn $\mathcal{I}$ die Formelmenge Φ erfüllt, dann erfüllt $\mathcal{I}$ auch die Formel ψ.

Notation

Für zwei Formeln $\varphi, \psi \in \mathsf{AL}$ schreiben wir kurz $\varphi \models \psi$ an Stelle von $\{\varphi\} \models \psi$ und sagen, dass die Formel ψ aus der Formel φ folgt.

Beispiel 2.16

Sei $\varphi := \big((X \vee Y) \wedge (\neg X \vee Y)\big)$ und $\psi := \big(Y \vee (\neg X \wedge \neg Y)\big)$.
Dann gilt $\varphi \models \psi$, aber es gilt *nicht* $\psi \models \varphi$ (kurz: $\psi \not\models \varphi$), denn:

X	Y	$(X \vee Y)$	$(\neg X \vee Y)$	φ	ψ
0	0	0	1	0	1
0	1	1	1	1	1
1	0	1	0	0	0
1	1	1	1	1	1

In jeder Zeile der Wahrheitstafel, in der in der mit „φ“ beschrifteten Spalte eine 1 steht, steht auch in der mit „ψ“ beschrifteten Spalte eine 1. Somit gilt $\varphi \models \psi$.

Andererseits steht in Zeile 1 in der mit „ψ“ beschrifteten Spalte eine 1 und in der mit „φ“ beschrifteten Spalte eine 0. Für die entsprechende Interpretation $\mathcal{I}$ (mit $\mathcal{I}(X) = 0$ und $\mathcal{I}(Y) = 0$) gilt also $[\![\psi]\!]^{\mathcal{I}} = 1$ und $[\![\varphi]\!]^{\mathcal{I}} = 0$. Daher gilt $\psi \not\models \varphi$.

Beispiel 2.17

Für alle Formeln $\varphi, \psi \in \mathsf{AL}$ gilt:

$$\{\,\varphi,\ \varphi \to \psi\,\} \ \models\ \psi.$$

Dies folgt unmittelbar aus der Definition der Semantik:
Sei $\mathcal{I}$ eine Interpretation mit $\mathcal{I} \models \{\varphi,\ \varphi \to \psi\}$. Dann gilt:

(1) $[\![\varphi]\!]^{\mathcal{I}} = 1$ und

(2) $[\![\varphi \to \psi]\!]^{\mathcal{I}} = 1$, d.h. es gilt $[\![\varphi]\!]^{\mathcal{I}} = 0$ oder $[\![\psi]\!]^{\mathcal{I}} = 1$.

Da $[\![\varphi]\!]^{\mathcal{I}} = 1$ gemäß (1) gilt, folgt gemäß (2), dass $[\![\psi]\!]^{\mathcal{I}} = 1$.

Bemerkung

Man kann die Folgerungsbeziehung $\{\varphi,\ \varphi \to \psi\} \models \psi$ als eine formale Schlussregel auffassen (ähnlich den Syllogismen in Kapitel 1):
Wenn φ und $\varphi \to \psi$ gelten, so muss auch ψ gelten.
Diese Regel, die unter dem Namen Modus Ponens bekannt ist, ist von grundlegender Bedeutung in der Logik.

Zusammenhänge

Lemma 2.18 (Allgemeingültigkeit, Unerfüllbarkeit und Folgerung)

Für jede Formel $\varphi \in \mathsf{AL}$ *gilt:*

(a) φ *ist allgemeingültig* $\iff$ $\neg\varphi$ *ist unerfüllbar* $\iff$ $\mathbf{1} \models \varphi$.

(b) φ *ist unerfüllbar* $\iff$ $\varphi \models \mathbf{0}$.

Lemma 2.19 (Erfüllbarkeit und die Folgerungsbeziehung)

Für alle Formelmengen $\Phi \subseteq \mathsf{AL}$ *und für alle Formeln* $\psi \in \mathsf{AL}$ *gilt:*

$$\Phi \models \psi \quad\iff\quad \Phi \cup \{\neg\psi\} \ \textit{ist unerfüllbar.}$$

Lemma 2.20 (Allgemeingültigkeit und die Folgerungsbeziehung)

(a) *Für jede Formel* $\varphi \in \mathsf{AL}$ *gilt:*

$$\varphi \text{ ist allgemeingültig} \iff \varphi \text{ folgt aus der leeren Menge},$$

kurz:

$$\models \varphi \quad \iff \quad \emptyset \models \varphi.$$

(b) *Für jede Formel* $\psi \in \mathsf{AL}$ *und jede endliche Formelmenge* $\Phi = \{\varphi_1, \ldots, \varphi_n\} \subseteq \mathsf{AL}$ *gilt:*

$$\Phi \models \psi \quad \iff \quad (\varphi_1 \land \cdots \land \varphi_n) \to \psi \text{ ist allgemeingültig.}$$

Bemerkung 2.21

Aus den beiden vorigen Lemmas erhält man leicht, dass für alle Formeln $\varphi, \psi \in \mathsf{AL}$ gilt:

$\varphi \models \psi \iff (\varphi \to \psi)$ ist allgemeingültig $\iff (\varphi \land \neg\psi)$ ist unerfüllbar.

Beweis.

Es sei $\Phi := \{\varphi\}$. Gemäß Lemma 2.20 gilt:

$$\Phi \models \psi \quad \iff \quad (\varphi \to \psi) \text{ ist allgemeingültig.}$$

Somit gilt: $\quad \varphi \models \psi \iff (\varphi \to \psi)$ ist allgemeingültig.

Außerdem gilt gemäß Lemma 2.19:

$$\Phi \models \psi \quad \iff \quad \Phi \cup \{\neg\psi\} \text{ ist unerfüllbar.}$$

Somit gilt: $\quad \varphi \models \psi \iff (\varphi \land \neg\psi)$ ist unerfüllbar. $\square$

Abschnitt 2.2:

Aussagenlogische Modellierung

Beispiel 1: Sudoku

Sudoku

	3							
			1	9	5			
	9	8					6	
8				6				
4					3			1
				2				
	6					2	8	
			4	1	9			5
							7	

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Aussagenlogisches Modell

Koordinaten der Felder:

Feld (i, j) ist das Feld in Zeile i und Spalte j.

Aussagensymbole:

Aussagensymbol $P_{i,j,k}$, für $i, j, k \in [9]$, steht für die Aussage

„Das Feld mit den Koordinaten (i, j) enthält die Zahl k."

Interpretationen beschreiben also Beschriftungen des 9×9-Gitters.

Ziel:

Für jede Anfangsbeschriftung A eine Formelmenge Φ_A, so dass für alle Interpretationen $\mathcal{I}$ gilt:

$$\mathcal{I} \models \Phi_A \quad \iff \quad \mathcal{I} \text{ beschreibt eine korrekte Lösung.}$$

Wir beschreiben zunächst eine Formelmenge $\Phi = \{\varphi_1, \ldots, \varphi_5\}$, die die Grundregeln des Spiels beschreibt.

Beschriftungen:

"Auf jedem Feld steht mindestens eine Zahl":

$$\varphi_1 \;:=\; \bigwedge_{i,j=1}^{9} \bigvee_{k=1}^{9} P_{i,j,k}.$$

"Auf jedem Feld steht höchstens eine Zahl":

$$\varphi_2 \;:=\; \bigwedge_{i,j=1}^{9} \bigwedge_{\substack{k,\ell=1\\ k\neq\ell}}^{9} \neg(P_{i,j,k} \wedge P_{i,j,\ell}).$$

Zeilen:

„Jede Zahl kommt in jeder Zeile vor“:

$$\varphi_3 \ := \ \bigwedge_{i=1}^{9} \bigwedge_{k=1}^{9} \bigvee_{j=1}^{9} P_{i,j,k}.$$

Spalten:

„Jede Zahl kommt in jeder Spalte vor“:

$$\varphi_4 \ := \ \bigwedge_{j=1}^{9} \bigwedge_{k=1}^{9} \bigvee_{i=1}^{9} P_{i,j.k}.$$

Blöcke:

„Jede Zahl kommt in jedem Block vor“:

$$\varphi_5 \ := \ \bigwedge_{i,j=0}^{2} \bigwedge_{k=1}^{9} \bigvee_{i',j'=1}^{3} P_{3i+i',3j+j',k}.$$

Anfangsbeschriftung:

Sei A die Anfangsbeschriftung. Wir setzen

$$\Phi_A \;:=\; \Phi \cup \{\, P_{i,j,k} \;:\; A \text{ beschriftet Feld } (i,j) \text{ mit der Zahl } k \,\}.$$

Automatische Lösung von Sudokus:

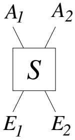
Um ein Sudoku mit Anfangsbeschriftung A zu lösen, können wir nun einfach die Formel $\psi_A := \bigwedge_{\varphi \in \Phi_A} \varphi$ bilden und die Wahrheitstafel zu dieser Formel aufstellen. Falls die Wahrheitstafel zeigt, dass ψ_A kein Modell besitzt, so ist das Sudoku nicht lösbar. Andernfalls können wir ein beliebiges Modell $\mathcal{I}$ von ψ_A hernehmen und daran die Lösung des Sudokus ablesen: Für jedes Feld (i,j) gibt es gemäß unserer Konstruktion der Formel ψ_A genau eine Zahl $k \in [9]$, so dass $\mathcal{I}(P_{i,j,k}) = 1$ ist. Diese Zahl k können wir in Feld (i,j) eintragen und erhalten damit eine Lösung des Sudokus.

Beispiel 2: Automatische Hardwareverifikation

Digitale Schaltkreise

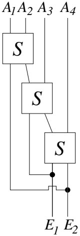
- Digitale Signale werden beschrieben durch 0 („aus“) und 1 („ein“).
- Schaltelemente berechnen ein oder mehrere Ausgangssignale aus einem oder mehreren Eingangssignalen. Das Ein-/Ausgabeverhalten eines Schaltelements lässt sich durch Wahrheitstafeln beschreiben.

  Beispiel:



E_1	E_2	A_1	A_2
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

- Schaltkreise sind Kombinationen solcher Schaltelemente. Beispiel:



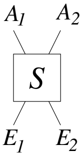
E_1	E_2	A_1	A_2	A_3	A_4
0	0	1	0	0	0
0	1	0	1	0	0
1	0	1	0	1	0
1	1	0	1	0	1

Formalisierung in der Aussagenlogik

Schaltelement:

- Für jeden Ein- und Ausgang ein Aussagensymbol.
- Für jeden Ausgang eine Formel, die den Wert der Ausgangs in Abhängigkeit von den Eingängen beschreibt.

Beispiel:



E_1	E_2	A_1	A_2
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

Aussagensymbole:

$P_1,\ P_2,\quad Q_1,\ Q_2$

Formeln:

$Q_1 \leftrightarrow \neg(P_1 \wedge P_2)$

$Q_2 \leftrightarrow (P_1 \wedge P_2)$

Schaltkreis:

- Für jeden Ein- und Ausgang ein Aussagensymbol,
  sowie für jedes Schaltelement ein Sortiment von Aussagensymbolen.
- Formeln für die Schaltelemente und Formeln für die „Verdrahtung".

Verifikation

Ziel:

Nachweis, dass der Schaltkreis eine gewisse *Korrektheitsbedingung* erfüllt.

Methode:

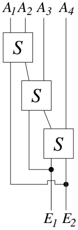
1. Beschreibe den Schaltkreis durch eine Menge Φ von Formeln.
2. Formuliere die Korrektheitsbedingung als Formel ψ.
3. Weise nach, dass ψ aus Φ folgt
   (bzw., dass $\Phi \cup \{\neg\psi\}$ unerfüllbar ist).

Bemerkung

Bei Bedarf kann die Korrektheitsbedingung insbesondere so gewählt werden, dass sie das gewünschte Ein-/Ausgabeverhalten des Schaltkreises vollständig spezifiziert.

Beispiele für Korrektheitsbedingungen

Schaltkreis:



Einige Korrektheitsbedingungen:

- Bei jeder Eingabe ist mindestens eine Ausgabe 1:

$$Q_1 \vee Q_2 \vee Q_3 \vee Q_4.$$

- Bei keiner Eingabe sind mehr als zwei Ausgaben 1:

$$\neg \bigvee_{1 \leqslant i < j < k \leqslant 4} (Q_i \wedge Q_j \wedge Q_k)$$

Vollständige Spezifikation des Ein-/Ausgabeverhaltens:

$$\begin{array}{cllll} & (\neg P_1 \wedge \neg P_2 & \to & Q_1 \wedge \neg Q_2 \wedge \neg Q_3 \wedge \neg Q_4 &) \\ \wedge & (\neg P_1 \wedge P_2 & \to & \neg Q_1 \wedge Q_2 \wedge \neg Q_3 \wedge \neg Q_4 &) \\ \wedge & (P_1 \wedge \neg P_2 & \to & Q_1 \wedge \neg Q_2 \wedge Q_3 \wedge \neg Q_4 &) \\ \wedge & (P_1 \wedge P_2 & \to & \neg Q_1 \wedge Q_2 \wedge \neg Q_3 \wedge Q_4 &) \end{array}$$

Abschnitt 2.3:

Äquivalenz und Adäquatheit

Äquivalenz

Definition 2.22

Zwei Formeln $\varphi, \psi \in \mathrm{AL}$ sind äquivalent (wir schreiben $\varphi \equiv \psi$), wenn sie von den selben Interpretationen erfüllt werden, d.h., wenn für alle Interpretationen $\mathcal{I}$ gilt: $\mathcal{I} \models \varphi \iff \mathcal{I} \models \psi$.

Zwei Formelmengen $\Phi, \Psi \subseteq \mathrm{AL}$ sind äquivalent (wir schreiben $\Phi \equiv \Psi$), wenn sie von den selben Interpretationen erfüllt werden, d.h., wenn für alle Interpretationen $\mathcal{I}$ gilt: $\mathcal{I} \models \Phi \iff \mathcal{I} \models \Psi$.

Beobachtung 2.23

(a) *Zwei Formeln* $\varphi, \psi \in \mathrm{AL}$ *sind genau dann äquivalent, wenn in den letzten Spalten ihrer Wahrheitstafeln jeweils die gleichen Einträge stehen.*

(b) *Für endliche Formelmengen* $\Phi = \{\varphi_1, \ldots, \varphi_m\}$, $\Psi = \{\psi_1, \ldots, \psi_n\} \subseteq \mathrm{AL}$ *gilt*

$$\Phi \equiv \Psi \iff \bigwedge_{i=1}^{m} \varphi_i \equiv \bigwedge_{j=1}^{n} \psi_j.$$

Beispiel: Für alle $X, Y \in \mathrm{AS}$ gilt: $\neg(X \vee Y) \equiv (\neg X \wedge \neg Y)$ und $X \equiv \neg\neg X$.

Äquivalenz und Allgemeingültigkeit

Lemma 2.24

(a) *Für alle Formeln* $\varphi, \psi \in \mathsf{AL}$ *gilt:*

$$\varphi \equiv \psi \qquad \Longleftrightarrow \qquad (\varphi \leftrightarrow \psi) \ \textit{ist allgemeingültig}.$$

(b) *Für alle* $\varphi \in \mathsf{AL}$ *gilt:*

$$\varphi \ \textit{ist allgemeingültig} \qquad \Longleftrightarrow \qquad \varphi \equiv \mathbf{1}.$$

Fundamentale Äquivalenzen

Satz 2.25

Für alle Formeln $\varphi, \psi, \chi \in$ AL gelten die folgenden Äquivalenzen:

(a) *Idempotenz:*

$$\varphi \wedge \varphi \;\equiv\; \varphi\,, \qquad \varphi \vee \varphi \;\equiv\; \varphi.$$

(b) *Kommutativität:*

$$\varphi \wedge \psi \;\equiv\; \psi \wedge \varphi\,, \qquad \varphi \vee \psi \;\equiv\; \psi \vee \varphi.$$

(c) *Assoziativität:*

$$(\varphi \wedge \psi) \wedge \chi \;\equiv\; \varphi \wedge (\psi \wedge \chi)\,, \qquad (\varphi \vee \psi) \vee \chi \;\equiv\; \varphi \vee (\psi \vee \chi).$$

(d) *Absorption:*

$$\varphi \wedge (\varphi \vee \psi) \;\equiv\; \varphi\,, \qquad \varphi \vee (\varphi \wedge \psi) \;\equiv\; \varphi.$$

(Fortsetzung: nächste Folie)

(e) *Distributivität:*

$$\varphi \wedge (\psi \vee \chi) \;\equiv\; (\varphi \wedge \psi) \vee (\varphi \wedge \chi)\,, \qquad \varphi \vee (\psi \wedge \chi) \;\equiv\; (\varphi \vee \psi) \wedge (\varphi \vee \chi).$$

(f) *Doppelte Negation:*

$$\neg\neg\varphi \;\equiv\; \varphi.$$

(g) *De Morgansche Regeln:*

$$\neg(\varphi \wedge \psi) \;\equiv\; \neg\varphi \vee \neg\psi\,, \qquad \neg(\varphi \vee \psi) \;\equiv\; \neg\varphi \wedge \neg\psi.$$

(h) *Tertium Non Datur:*

$$\varphi \wedge \neg\varphi \;\equiv\; \mathbf{0}\,, \qquad \varphi \vee \neg\varphi \;\equiv\; \mathbf{1}.$$

(Fortsetzung: nächste Folie)

(i)

$$\varphi \wedge \mathbf{1} \;\equiv\; \varphi\,, \qquad \varphi \vee \mathbf{0} \;\equiv\; \varphi\,,$$
$$\varphi \wedge \mathbf{0} \;\equiv\; \mathbf{0}\,, \qquad \varphi \vee \mathbf{1} \;\equiv\; \mathbf{1}.$$

(j)

$$\mathbf{1} \;\equiv\; \neg\mathbf{0}\,, \qquad \mathbf{0} \;\equiv\; \neg\mathbf{1}.$$

(k) *Elimination der Implikation:*

$$\varphi \to \psi \;\equiv\; \neg\varphi \vee \psi.$$

Beweis.

Alle hier genannten Äquivalenzen können leicht mit Hilfe der Wahrheisttafelmethode überprüft werden.

Zum Beispiel die erste de Morgansche Regel:

$$\neg(\varphi \wedge \psi) \;\equiv\; \neg\varphi \vee \neg\psi.$$

Wir berechnen dazu folgende Wahrheitstafeln:

φ	ψ	$\varphi \wedge \psi$	$\neg(\varphi \wedge \psi)$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

φ	ψ	$\neg\varphi$	$\neg\psi$	$\neg\varphi \vee \neg\psi$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

Die letzten Spalten der beiden Wahrheitstafeln sind gleich, also sind die Formeln äquivalent.

Rest: Übung. □

Bemerkung

Durch schrittweises Anwenden der in Satz 2.25 aufgelisteten Äquivalenzen kann man eine gegebene Formel in eine zu ihr äquivalente Formel umformen.

Das Dualitätsprinzip

Definition 2.26

Sei $\varphi \in \mathsf{AL}$ eine Formel, in der keine Implikationen vorkommt.
Die zu φ duale Formel ist die Formel $\widetilde{\varphi} \in \mathsf{AL}$, die aus φ entsteht, indem man überall $\mathbf{0}$ durch $\mathbf{1}$, $\mathbf{1}$ durch $\mathbf{0}$, $\wedge$ durch $\vee$ und $\vee$ durch $\wedge$ ersetzt.

Beobachtung 2.27

In Satz 2.25(a)–(e) *und* (g)–(j) *stehen auf der linken Seite jeweils die dualen Formeln der Formeln auf der rechten Seite.*

Satz 2.28 (Dualitätssatz der Aussagenlogik)

Für alle Formeln $\varphi, \psi \in \mathsf{AL}$, *in denen keine Implikation vorkommt, gilt:*

$$\varphi \equiv \psi \quad \Longleftrightarrow \quad \widetilde{\varphi} \equiv \widetilde{\psi}.$$

Beweise per Induktion über den Aufbau von Formeln

- Ähnlich wie wir Aussagen über die natürlichen Zahlen durch vollständige Induktion beweisen können, können wir Aussagen über Formeln per Induktion über den Aufbau der Formeln beweisen.

- Im Induktionsanfang beweisen wir die Aussagen für die atomaren Formeln, und im Induktionschritt schließen wir von den Bestandteilen einer Formel auf die Formel selbst.

- Dieses Vorgehen ist z.B. dadurch gerechtfertigt, dass es sich auch als vollständige Induktion über die Höhe des Syntaxbaumes auffassen lässt.

Schematisch sieht der Beweis einer Aussage $\mathbb{A}(\varphi)$ für alle Formeln $\varphi \in \mathsf{AL}$ wie folgt aus:

Induktionsanfang:

- Beweise $\mathbb{A}(\mathbf{0})$ und $\mathbb{A}(\mathbf{1})$.
- Beweise $\mathbb{A}(X)$ für alle $X \in \mathsf{AS}$.

Induktionsschritt:

- Beweise $\mathbb{A}(\neg\varphi)$ unter der Annahme, dass $\mathbb{A}(\varphi)$ gilt.
- Beweise $\mathbb{A}(\varphi \wedge \psi)$ unter der Annahme, dass $\mathbb{A}(\varphi)$ und $\mathbb{A}(\psi)$ gelten.
- Beweise $\mathbb{A}(\varphi \vee \psi)$ unter der Annahme, dass $\mathbb{A}(\varphi)$ und $\mathbb{A}(\psi)$ gelten.
- Beweise $\mathbb{A}(\varphi \rightarrow \psi)$ unter der Annahme, dass $\mathbb{A}(\varphi)$ und $\mathbb{A}(\psi)$ gelten.

Um den Dualitätssatz zu beweisen benötigen wir zunächst noch eine Definition. Der Kern des Beweises steckt im darauf folgenden Lemma.

Definition 2.29

Sei $\mathcal{I}$ eine Interpretation. Die zu $\mathcal{I}$ duale Interpretation $\widetilde{\mathcal{I}}$ ist definiert durch $\widetilde{\mathcal{I}}(X) := 1 - \mathcal{I}(X)$ für alle $X \in \text{AS}$.

D.h. für alle Aussagensymbole X gilt:

$$\widetilde{\mathcal{I}}(X) \;=\; \begin{cases} 0\,, & \text{falls } \mathcal{I}(X) = 1 \\ 1\,, & \text{falls } \mathcal{I}(X) = 0 \end{cases}$$

Lemma 2.30

Für alle Formeln $\varphi \in \text{AL}$, in denen keine Implikation vorkommt, und alle Interpretationen $\mathcal{I}$ gilt:

$$\mathcal{I} \models \widetilde{\varphi} \quad\Longleftrightarrow\quad \widetilde{\mathcal{I}} \not\models \varphi.$$

Beweis von Satz 2.28.

Seien $\varphi, \psi \in$ AL Formeln, in denen keine Implikation vorkommt.
Wir wollen zeigen, dass gilt: $\varphi \equiv \psi \iff \widetilde{\varphi} \equiv \widetilde{\psi}$.

„$\Longrightarrow$“: Es gilt:[1]

$$
\begin{aligned}
& \varphi \equiv \psi \\
\Longrightarrow \quad & \text{F.a. Interpretationen } \mathcal{I} \text{ gilt: } \left(\widetilde{\mathcal{I}} \models \varphi \iff \widetilde{\mathcal{I}} \models \psi\right) \\
\stackrel{\text{Lemma 2.30}}{\Longrightarrow} \quad & \text{F.a. Interpretationen } \mathcal{I} \text{ gilt: } \left(\mathcal{I} \not\models \widetilde{\varphi} \iff \mathcal{I} \not\models \widetilde{\psi}\right) \\
\Longrightarrow \quad & \text{F.a. Interpretationen } \mathcal{I} \text{ gilt: } \left(\mathcal{I} \models \widetilde{\varphi} \iff \mathcal{I} \models \widetilde{\psi}\right) \\
\Longrightarrow \quad & \widetilde{\varphi} \equiv \widetilde{\psi}.
\end{aligned}
$$

„$\Longleftarrow$“: Es gilt:

$$
\begin{aligned}
\widetilde{\varphi} \equiv \widetilde{\psi} \quad & \Longrightarrow \quad \widetilde{\widetilde{\varphi}} \equiv \widetilde{\widetilde{\psi}} && \text{(andere Beweisrichtung)} \\
& \Longrightarrow \quad \varphi \equiv \psi && \text{(weil } \widetilde{\widetilde{\varphi}} = \varphi \text{ und } \widetilde{\widetilde{\psi}} = \psi\text{).}
\end{aligned}
$$

$\square$

[1] Wir schreiben kurz „f.a.“ als Abkürzung für die Worte „für alle“

Funktionale Vollständigkeit der Aussagenlogik

Im Folgenden bezeichnen wir als Wahrheitstafel eine Tabelle mit $n+1$ Spalten und 2^n Zeilen, die für jedes Tupel $(b_1, \dots, b_n) \in \{0,1\}^n$ genau eine Zeile enthält, deren erste n Einträge $b_1, \dots, b_n$ sind.

Satz 2.31 (Funktionale Vollständigkeit der Aussagenlogik)

Zu jeder Wahrheitstafel gibt es eine Formel $\varphi \in AL$ mit dieser Wahrheitstafel.

Mathematisch präzise lässt sich dieser Satzes wie folgt formulieren:

Für alle $n \in \mathbb{N}$ gibt es zu jeder Funktion $F : \{0,1\}^n \to \{0,1\}$ eine Formel $\varphi(A_1, \dots, A_n) \in \mathrm{AL}$, so dass für alle $(b_1, \dots, b_n) \in \{0,1\}^n$ gilt:

$$F(b_1, \dots, b_n) = 1 \iff \varphi[b_1, \dots, b_n] = 1.$$

Definition 2.32

Funktionen $F : \{0,1\}^n \to \{0,1\}$ (mit $n \in \mathbb{N}$) nennt man Boolesche Funktionen (der Stelligkeit n).

Bevor wir Satz 2.31 beweisen, betrachten wir zunächst ein Beispiel.

Beispiel 2.33

Betrachte die Wahrheitstafel T:

b_1	b_2	b_3	$F(b_1, b_2, b_3)$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Eine Formel $\varphi(A_1, A_2, A_3)$, so dass T die Wahrheitstafel für φ ist, kann man folgendermaßen erzeugen:

- Betrachte alle Zeilen von T, bei denen in der letzten Spalte eine „1“ steht.
- Für jede solche Zeile konstruiere eine Formel, die genau von der zu der Zeile gehörenden Belegung von b_1, b_2, b_3 erfüllt wird.
- Bilde die Disjunktion (d.h. die „Ver*oder*ung“) über all diese Formeln. Dies liefert die gesuchte Formel φ.

In unserer Beispiel-Wahrheitstafel T gibt es genau 3 Zeilen, bei denen in der letzten Spalte eine „1“ steht, nämlich die Zeilen

b_1	b_2	b_3	$F(b_1, b_2, b_3)$	zur jeweiligen Zeile gehörende Formel:
0	0	0	1	$(\neg A_1 \wedge \neg A_2 \wedge \neg A_3)$
0	0	1	1	$(\neg A_1 \wedge \neg A_2 \wedge A_3)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
1	0	1	1	$(A_1 \wedge \neg A_2 \wedge A_3)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Insgesamt erhalten wir dadurch die zur Wahrheitstafel T passende Formel

$$\varphi \;=\; (\neg A_1 \wedge \neg A_2 \wedge \neg A_3) \vee (\neg A_1 \wedge \neg A_2 \wedge A_3) \vee (A_1 \wedge \neg A_2 \wedge A_3).$$

Adäquatheit

Satz 2.31 besagt, dass die Aussagenlogik AL die größtmögliche Aussdruckstärke hat. Dafür reichen allerdings schon „kleinere“ Logiken, wie wir im Folgenden sehen werden.

Definition 2.34

Sei $\tau \subseteq \{\mathbf{0}, \mathbf{1}, \neg, \wedge, \vee, \rightarrow\}$.

(a) AL(τ) sei das Fragment der Logik AL, das aus den Formeln besteht, in denen nur Junktoren und Konstanten aus τ vorkommen.

(b) τ heißt adäquat, wenn jede Formel $\varphi \in$ AL äquivalent zu einer Formel in AL(τ) ist.

Beispiele 2.35

(a) $\{\neg, \wedge\}$, $\{\neg, \vee\}$, $\{\mathbf{0}, \rightarrow\}$ sind adäquat.

(b) $\{\wedge, \vee, \rightarrow\}$ ist nicht adäquat.

Andere Junktoren

- Die Auswahl der Junktoren $\neg, \wedge, \vee, \rightarrow$ (und $\leftrightarrow$ als Abkürzung) für „unsere" aussagenlogische Sprache richtet sich nach dem umgangssprachlichen Gebrauch und den Erfordernissen des formalen Schließens, ist aber in gewisser Weise willkürlich.

- Durch Festlegung ihrer Wahrheitstafeln können wir auch andere Junktoren definieren und erhalten daraus andere aussagenlogische Sprachen.

- Für jede Menge τ von so definierten Junktoren und den boolschen Konstanten (die wir als „nullstellige" Junktoren auffassen können) sei AL(τ) die daraus gebildete aussagenlogische Sprache.

- Satz 2.31 besagt dann, dass jede Formel in AL(τ) zu einer Formel in AL äquivalent ist. Gilt die Umkehrung ebenfalls, so bezeichnen wir τ als adäquat.

Beispiele 1: Exklusives Oder

Der 2-stellige Junktor $\oplus$ sei definiert durch

φ	ψ	$\varphi \oplus \psi$
0	0	0
0	1	1
1	0	1
1	1	0

Intuitiv bedeutet $\varphi \oplus \psi$ „entweder φ oder ψ".
Man nennt $\oplus$ auch exklusives Oder.

Der dreistellige Mehrheitsjunktor

Der 3-stellige Junktor M sei definiert durch

φ	ψ	χ	$M(\varphi,\psi,\chi)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Intuitiv ist $M(\varphi,\psi,\chi)$ also genau dann wahr, wenn mindestens zwei (also die Mehrheit) der Formeln φ,ψ,χ wahr sind.

NAND

Der folgende zweistellige Junktor ist bekannt als NAND-Gatter (not-and) oder Sheffer-Strich:

φ	ψ	$(\varphi \mid \psi)$
0	0	1
0	1	1
1	0	1
1	1	0

Satz 2.36

$\{\,|\,\}$ *ist adäquat.*

Abschnitt 2.4:
Normalformen

Vereinfachende Annahme

In diesem Abschnitt betrachten wir nur Formeln in $\text{AL}(\{\neg, \vee, \wedge\})$.

Rechtfertigung

Die Annahme bedeutet keine wesentliche Einschränkung, weil die Menge $\{\neg, \vee, \wedge\}$ adäquat ist.

NNF

Definition 2.37

Eine Formel ist in Negationsnormalform (NNF), wenn sie zu $\mathrm{AL}(\{\neg, \wedge, \vee\})$ gehört und Negationszeichen nur unmittelbar vor Aussagensymbolen auftreten.

Satz 2.38

Jede aussagenlogische Formel ist äquivalent zu einer Formel in NNF.

Ein NNF-Algorithmus

Eingabe: Formel $\varphi \in \mathsf{AL}(\{\neg, \wedge, \vee\})$.

Ausgabe: Formel φ' in NNF

Verfahren:

1. Wiederhole folgende Schritte:
2. Wenn φ in NNF ist, dann halte mit
   Ausgabe φ.
3. Ersetze eine Subformel von φ der Gestalt
   $\neg(\psi_1 \wedge \psi_2)$ durch $(\neg\psi_1 \vee \neg\psi_2)$
   oder eine Subformel der Gestalt
   $\neg(\psi_1 \vee \psi_2)$ durch $(\neg\psi_1 \wedge \neg\psi_2)$
   oder eine Subformel der Gestalt
   $\neg\neg\psi$ durch ψ.
   Sei φ' die resultierende Formel.
4. $\varphi := \varphi'$.

Korrektheit des NNF-Algorithmus

Satz 2.39

Für jede Eingabeformel $\varphi \in \mathsf{AL}(\{\neg, \wedge, \vee\})$ *gibt der NNF-Algorithmus nach endlich vielen Schritten eine zu* φ *äquivalente Formel* φ' *in NNF aus.*

(hier ohne Beweis)

Bemerkung

Unter Verwendung geeigneter Datenstrukturen lässt sich der NNF-Algorithmus mit linearer Laufzeit implementieren, d.h., Laufzeit $O(n)$ bei Eingabe einer Formel der Länge n.

Beispiel 2.40

Das Ziel ist, die Formel $\Big(\big(\neg A_0 \wedge \neg((A_0 \vee A_1) \to A_0)\big) \to \mathbf{0}\Big)$

in NNF zu bringen, d.h. eine zu ihr äquivalente Formel in NNF zu finden.

Lösung: Wir ersetzen zunächst die Konstanten $\mathbf{0}$ und $\mathbf{1}$ sowie alle Implikationspfeile durch geeignete Formeln aus $\mathrm{AL}(\{\neg, \wedge, \vee\})$ und wenden dann den NNF-Algorithmus an. Der Teil einer Formel, der als Nächstes ersetzt wird, ist im Folgenden jeweils unterstrichen.

$$
\begin{aligned}
& \Big(\big(\neg A_0 \wedge \neg((A_0 \vee A_1) \to A_0)\big) \to \underline{\mathbf{0}}\Big) \\
\equiv \quad & \Big(\big(\neg A_0 \wedge \neg((A_0 \vee A_1) \to A_0)\big) \underline{\to} (A_0 \wedge \neg A_0)\Big) \\
\equiv \quad & \Big(\neg\big(\neg A_0 \wedge \neg((A_0 \vee A_1) \underline{\to} A_0)\big) \vee (A_0 \wedge \neg A_0)\Big) \\
\equiv \quad & \Big(\underline{\neg}\big(\neg A_0 \wedge \neg(\neg(A_0 \vee A_1) \vee A_0)\big) \vee (A_0 \wedge \neg A_0)\Big) \\
\equiv \quad & \Big(\big(\underline{\neg\neg} A_0 \vee \underline{\neg\neg}(\neg(A_0 \vee A_1) \vee A_0)\big) \vee (A_0 \wedge \neg A_0)\Big) \\
\equiv \quad & \Big(\big(A_0 \vee (\underline{\neg}(A_0 \vee A_1) \vee A_0)\big) \vee (A_0 \wedge \neg A_0)\Big) \\
\equiv \quad & \Big(\big(A_0 \vee ((\neg A_0 \wedge \neg A_1) \vee A_0)\big) \vee (A_0 \wedge \neg A_0)\Big).
\end{aligned}
$$

Diese Formel ist offensichtlicherweise in NNF.

Klammern bei Konjunktionen und Disjunktionen

Weil $\wedge$ assoziativ ist, können wir Formeln der Gestalt $\bigwedge_{i=1}^{n} \varphi_i$ etwas großzügiger interpretieren. Von nun an stehe $\bigwedge_{i=1}^{n} \varphi_i$ für $\varphi_1 \wedge \cdots \wedge \varphi_n$ mit *irgendeiner* Klammerung.

Entsprechend verfahren wir mit Disjunktionen.

Beispiel

Die Formel $\bigwedge_{i=1}^{4} \varphi_i$ kann für jede der folgenden Formeln stehen:

$$(((\varphi_1 \wedge \varphi_2) \wedge \varphi_3) \wedge \varphi_4) \, ,$$

$$((\varphi_1 \wedge (\varphi_2 \wedge \varphi_3)) \wedge \varphi_4) \, ,$$

$$((\varphi_1 \wedge \varphi_2) \wedge (\varphi_3 \wedge \varphi_4)) \, ,$$

$$(\varphi_1 \wedge ((\varphi_2 \wedge \varphi_3) \wedge \varphi_4)) \, ,$$

$$(\varphi_1 \wedge (\varphi_2 \wedge (\varphi_3 \wedge \varphi_4))) \, .$$

DNF und KNF

Definition 2.41

(a) Ein Literal ist eine Formel der Gestalt X oder $\neg X$, wobei $X \in \text{AS}$.
Im ersten Fall sprechen wir von einem positiven, im zweiten Fall von einem negativen Literal.

(b) Eine Formel ist in disjunktiver Normalform (DNF), wenn sie eine Disjunktion von Konjunktionen von Literalen ist, d.h., wenn sie die Form

$$\bigvee_{i=1}^{n} \Big(\bigwedge_{j=1}^{m_i} \lambda_{i,j} \Big)$$

hat, wobei $n, m_1, \ldots, m_n \geqslant 1$ sind und die $\lambda_{i,j}$ für alle $i \in [n]$ und $j \in [m_i]$ Literale sind. Die Subformeln $\kappa_i := \bigwedge_{j=1}^{m_i} \lambda_{i,j}$, für $i \in [n]$, nennen wir die (konjunktiven) Klauseln der Formel.

Beispiele:

- $(A_1 \wedge \neg A_2 \wedge A_3) \vee (\neg A_2 \wedge \neg A_3) \vee (A_2 \wedge A_1)$ ist in DNF
- $A_1 \vee \neg A_2 \vee A_3$ ist in DNF (mit $n = 3$ und $m_1 = m_2 = m_3 = 1$)
- $A_1 \wedge \neg A_2 \wedge A_3$ ist in DNF (mit $n = 1$ und $m_1 = 3$) und gleichzeitig ist diese Formel eine konjunktive Klausel

(c) Eine Formel ist in konjunktiver Normalform (KNF), wenn sie eine Konjunktion von Disjunktion von Literalen ist, d.h., wenn sie die Form

$$\bigwedge_{i=1}^{n} \Big(\bigvee_{j=1}^{m_i} \lambda_{i,j} \Big)$$

hat, wobei $n, m_1, \ldots, m_n \geqslant 1$ sind und die $\lambda_{i,j}$ für alle $i \in [n]$ und $j \in [m_i]$ Literale sind. Die Subformeln $\kappa_i := \bigvee_{j=1}^{m_i} \lambda_{i,j}$, für $i \in [n]$, nennen wir die (disjunktiven) Klauseln der Formel.

Beispiele:

- $(A_1 \vee \neg A_2 \vee A_3) \wedge (\neg A_2 \vee \neg A_3) \wedge (A_2 \vee A_1)$ ist in KNF
- $A_1 \vee \neg A_2 \vee A_3$ ist in KNF (mit $n = 1$ und $m_1 = 3$) und gleichzeitig ist diese Formel eine disjunktive Klausel
- $A_1 \wedge \neg A_2 \wedge A_3$ ist in KNF (mit $n = 3$ und $m_1 = m_2 = m_3 = 1$)

Normalformen spielen in vielen Anwendungsgebieten eine wichtige Rolle. Beispielsweise geht man in der Schaltungstechnik (Hardware-Entwurf) oft von DNF-Formeln aus, während bei der aussagenlogischen Modellbildung oftmals KNF-Formeln auftreten, da sich eine Sammlung von einfach strukturierten Aussagen sehr gut durch eine Konjunktion von Klauseln ausdrücken lässt.

Satz 2.42

Jede aussagenlogische Formel ist äquivalent zu einer Formel in DNF und zu einer Formel in KNF.

Bemerkung 2.43

Der Beweis von Satz 2.42 zeigt Folgendes:
Um für eine gegebene Formel ψ eine äquivalente Formel φ in

- DNF zu erzeugen, können wir die Wahrheitstafel für ψ aufstellen und dann wie in Beispiel 2.33 vorgehen (bzw. $\varphi := A_1 \wedge \neg A_1$ setzen, falls ψ unerfüllbar ist).
- KNF zu erzeugen, können wir wie folgt vorgehen:
  (1) Stelle die Wahrheitstafel für ψ auf.
  (2) Falls in der letzten Spalte nur „1“en stehen, setze $\varphi := A_1 \vee \neg A_1$.
  (3) Ansonsten gehe wie folgt vor:
    - Betrachte alle Zeilen der Wahrheitstafel, bei denen in der letzten Spalte eine „0“ steht.
    - Für jede solche Zeile konstruiere die disjunktive Klausel, die von allen Interpretationen außer der zur Zeile gehörenden erfüllt wird.

      Beispiel: Wenn die Zeile der Wahrheitstafel die Form

      $$0\ 1\ 1 \mid 0$$

      hat, so gehört dazu die disjunktive Klausel

      $$A_1 \vee \neg A_2 \vee \neg A_3.$$

    - Bilde die Konjunktion all dieser disjunktiven Klauseln.
      Dies liefert die gesuchte KNF-Formel φ.

Wenn eine Formel sehr viele verschiedene Aussagensymbole enthält, die zur Formel gehörige Wahrheitstafel also sehr groß ist, ist das gerade beschriebene Verfahren zur Umformung in DNF oder KNF sehr zeitaufwändig.

In solchen Fällen ist es ratsam, stattdessen zu versuchen, die gewünschte Normalform durch Äquivalenzumformungen zu erzeugen.

Beispiel 2.44

Sei $\varphi := \Big(\big(\neg A_0 \wedge (A_0 \to A_1) \big) \vee (A_2 \to A_3) \Big)$.

Transformation von φ in NNF: siehe Tafel

Transformation in DNF: siehe Tafel

Transformation in KNF: siehe Tafel

Je nach Formel muss man ggf. die Distributivitätsregel mehrmals anwenden, bis man eine Formel der gewünschten Normalform erhält.

Ein DNF-Algorithmus

Eingabe: Formel $\varphi \in \mathsf{AL}(\{\neg, \wedge, \vee\})$ in NNF.

Ausgabe: Formel φ'' in DNF

Verfahren:

1. Wiederhole folgende Schritte:
2. Wenn φ in DNF ist, dann halte mit Ausgabe φ.
3. Ersetze eine Subformel von φ der Gestalt $(\psi_1 \wedge (\psi_2 \vee \psi_3))$ durch $((\psi_1 \wedge \psi_2) \vee (\psi_1 \wedge \psi_3))$ oder eine Subformel der Gestalt $((\psi_1 \vee \psi_2) \wedge \psi_3)$ durch $((\psi_1 \wedge \psi_3) \vee (\psi_2 \wedge \psi_3))$. Sei φ' die resultierende Formel.
4. $\varphi := \varphi'$.

Satz 2.45

Für jede Eingabeformel φ in NNF gibt der DNF-Algorithmus nach endlich vielen Schritten eine zu φ äquivalente Formel φ'' in DNF aus.

(hier ohne Beweis)

Analog kann man auch einen „KNF-Algorithmus“ angeben, der bei Eingabe einer NNF-Formel eine äquivalente Formel in KNF erzeugt (Details: Übung).

Eine kleine Formel mit großer DNF

Satz 2.46

Sei $n \in \mathbb{N}$ mit $n \geqslant 1$, seien $X_1, \ldots, X_n$ und $Y_1, \ldots, Y_n$ genau $2n$ verschiedene Aussagensymbole und sei

$$\varphi_n \;:=\; \bigwedge_{i=1}^{n} (\, X_i \vee \neg Y_i \,).$$

Jede zu φ_n äquivalente Formel in DNF hat mindestens 2^n konjunktive Klauseln.

Beweis: Übung

Korollar 2.47

<u>Jeder</u> Algorithmus, der bei Eingabe von beliebigen aussagenlogischen Formeln dazu äquivalente Formeln in DNF erzeugt, hat eine Laufzeit, die im worst-case exponentiell ist, d.h., $2^{\Omega(n)}$ bei Eingabe von Formeln der Länge n.

Abschnitt 2.5:

Der Endlichkeitssatz

Der Endlichkeitssatz (auch bekannt als Kompaktheitssatz)

Um nachzuweisen, dass eine gegebene *unendliche* Formelmenge erfüllbar ist, ist der folgende Satz sehr nützlich.

Satz 2.48 (Der Endlichkeitssatz der Aussagenlogik)

Für jede Formelmenge $\Phi \subseteq \mathsf{AL}$ *gilt:*

$$\Phi \textit{ ist erfüllbar} \quad \iff \quad \textit{Jede } \underline{\textit{endliche}} \textit{ Teilmenge von } \Phi \textit{ ist erfüllbar.}$$

Korollar 2.49 (Variante des Endlichkeitssatzes)

Sei $\Phi \subseteq \mathsf{AL}$ *und sei* $\psi \in \mathsf{AL}$. *Dann gilt:*

$$\Phi \models \psi \quad \iff \quad \textit{Es gibt eine endliche Teilmenge } \Gamma \textit{ von } \Phi, \textit{ so dass } \Gamma \models \psi.$$

Anwendung: Färbbarkeit

Zur Erinnerung:

- Ein Graph $G = (V, E)$ besteht aus einer nicht-leeren Menge V von Knoten und einer Menge $E \subseteq \big\{\{x, y\} : x, y \in V,\ x \neq y\big\}$ von (ungerichteten) Kanten.
- Ein Subgraph eines Graphen $G = (V, E)$ ist ein Graph $H = (V', E')$ mit $V' \subseteq V$ und $E' \subseteq E$.
- Ein Graph ist endlich (bzw. unendlich), wenn seine Knotenmenge endlich (bzw. unendlich) ist.

Definition 2.50

Sei $k \in \mathbb{N}$ mit $k \geqslant 1$.
Eine k-Färbung eines Graphen $G = (V, E)$ ist eine Abbildung $f : V \to [k]$, so dass für alle Kanten $\{v, w\} \in E$ gilt: $f(v) \neq f(w)$.
G heißt k-färbbar, falls es eine k-Färbung von G gibt.

Satz 2.51

Sei $k \in \mathbb{N}$ mit $k \geqslant 1$.
Ein unendlicher Graph G mit Knotenmenge $\mathbb{N}$ ist genau dann k-färbbar, wenn jeder endliche Subgraph von G k-färbbar ist.

Abschnitt 2.6:

Resolution

Um nachzuweisen, dass eine gegebene KNF-Formel *unerfüllbar* ist, ist das im Folgenden vorgestellte Resolutionsverfahren nützlich.

Beispiel 2.52

Wir wollen nachweisen, dass die KNF-Formel

$$\varphi := (\neg P \vee \neg R) \wedge (P \vee \neg R) \wedge (\neg Q \vee S) \wedge (Q \vee R \vee T) \wedge \neg T \wedge (\neg S \vee R)$$

unerfüllbar ist. Dazu können wir wie folgt argumentieren:

Angenommen, eine Interpretation $\mathcal{I}$ erfüllt φ.

- Dann gilt $\mathcal{I} \models \neg T$.
- Aus $\mathcal{I} \models Q \vee R \vee T$ und $\mathcal{I} \models \neg T$ folgt dann $\mathcal{I} \models Q \vee R$.
- Aus $\mathcal{I} \models Q \vee R$ und $\mathcal{I} \models \neg Q \vee S$ folgt $\mathcal{I} \models R \vee S$.
- Aus $\mathcal{I} \models R \vee S$ und $\mathcal{I} \models \neg S \vee R$ folgt $\mathcal{I} \models R$.
- Aus $\mathcal{I} \models \neg P \vee \neg R$ und $\mathcal{I} \models P \vee \neg R$ folgt $\mathcal{I} \models \neg R$.

  Das ist ein *Widerspruch*. Somit ist φ *nicht* erfüllbar.

Umwandlung in kleine KNF-Formeln

Das Resolutionsverfahren, das wir im Folgenden vorstellen, funktioniert nur für KNF-Formeln.

Wir wissen bereits:

- Zu jeder Formel φ gibt es eine äquivalente Formel in KNF.
- Aber möglicherweise ist die kleinste zu φ äquivalente KNF-Formel exponentiell groß in der Größe von φ.

Wenn es uns nur um die Frage geht, ob eine Formel φ (un)erfüllbar ist, ist es aber auch gar nicht nötig, eine zu φ äquivalente KNF-Formel zu finden. Es reicht, eine zu φ erfüllbarkeitsäquivalente KNF-Formel zu konstruieren.

Definition 2.53

Zwei Formeln φ und ψ heißen erfüllbarkeitsäquivalent, falls gilt:

$$\varphi \text{ ist erfüllbar} \iff \psi \text{ ist erfüllbar.}$$

Eine beliebige Formel in eine *erfüllbarkeitsäquivalente* KNF-Formel umzuwandeln, ist in Linearzeit möglich.

Beispiel 2.54

Um die Formel

$$\varphi \ := \quad (\, P \to \neg Q \,) \ \vee \ (\, \neg (\, P \wedge Q \,) \ \wedge \ R \,)$$

in eine erfüllbarkeitsäquivalente KNF-Formel umzuformen, können wir wie folgt vorgehen.

Das Tseitin-Verfahren

Auf die gleiche Weise wie in Beispiel 2.54 können wir jede beliebige aussagenlogische Formel in eine erfüllbarkeitsäquivalente KNF-Formel umwandeln. Dieses Verfahren wird Tseitin-Verfahren genannt. Eine Laufzeitanalyse zeigt, dass das Tseitin-Verfahren in Linearzeit durchgeführt werden kann. Insgesamt erhalten wir so den folgenden Satz.

Satz 2.55

Zu jeder aussagenlogischen Formel φ gibt es eine aussagenlogische Formel φ_K mit folgenden Eigenschaften:

(a) φ_K *ist erfüllbarkeitsäquivalent zu* φ.

(b) φ_K *ist in 3-KNF, d.h., in KNF, wobei jede disjunktive Klausel aus höchstens 3 Literalen besteht (wir sagen: die Klauseln haben Länge* $\leqslant 3$*).*

(c) $|\varphi_K| = O(|\varphi|)$.

Außerdem gibt es einen Algorithmus, der φ_K bei Eingabe von φ in Linearzeit berechnet.

Notation

$|\varphi|$ bezeichnet die Länge (bzw. Größe) einer aussagenlogischen Formel φ, d.h. die Länge von φ aufgefasst als Wort über dem Alphabet A_{AL}.

Repräsentation von KNF-Formeln

Für den Rest diese Abschnitts werden wir nur noch KNF-Formeln betrachten, und wenn wir von Klauseln sprechen, meinen wir stets disjunktive Klauseln, also Disjunktionen von Literalen.

Für das Resolutionsverfahren ist die folgende Repräsentation von Klauseln und KNF-Formeln sehr hilfreich:

- Eine Klausel $(\lambda_1 \vee \cdots \vee \lambda_\ell)$, die aus Literalen $\lambda_1, \ldots, \lambda_\ell$ besteht, identifizieren wir mit der Menge $\{\lambda_1, \ldots, \lambda_\ell\}$ ihrer Literale.

  Beispiel: Wir schreiben z.B. $\{A_1, \neg A_2, A_3\}$ um die Klausel $(A_1 \vee \neg A_2 \vee A_3)$ zu bezeichnen.

  D.h.: Ab jetzt sind disjunktive Klauseln für uns dasselbe wie endliche Mengen von Literalen. Wenn wir von einer Klausel sprechen, meinen wir eine endliche Menge von Literalen und identifizieren diese mit der Formel, die aus der Disjunktion all dieser Literale besteht.

  Spezialfall: Die leere Menge $\emptyset$ entspricht der unerfüllbaren Formel $\mathbf{0}$ (die wiederum der „Formel" entspricht, die aus der Disjunktion aller Literale aus $\emptyset$ besteht).

- Eine KNF-Formel $\varphi = \bigwedge_{i=1}^{m} \gamma_i$, die aus (disjunktiven) Klauseln $\gamma_1, \ldots, \gamma_m$ besteht, identifizieren wir mit der Menge $\Gamma := \{\gamma_1, \ldots, \gamma_m\}$ ihrer Klauseln.

  Offensichtlicherweise gilt für alle Interpretationen $\mathcal{I}$:

$$\mathcal{I} \models \varphi \iff \mathcal{I} \models \Gamma.$$

Beispiel: Die KNF-Formel $\varphi = A_1 \wedge (\neg A_2 \vee A_1) \wedge (A_3 \vee \neg A_2 \vee \neg A_1)$ repräsentieren wir durch die endliche Klauselmenge

$$\{\ A_1,\ \ (\neg A_2 \vee A_1),\ \ (A_3 \vee \neg A_2 \vee \neg A_1)\ \}$$

bzw. durch

$$\{\ \{A_1\},\ \ \{\neg A_2, A_1\},\ \ \{A_3, \neg A_2, \neg A_1\}\ \}$$

„Erfüllbarkeit von KNF-Formeln“ ist damit im Wesentlichen dasselbe Problem wie „Erfüllbarkeit von endlichen Mengen von Klauseln“.

Resolution

Notation

Für ein Literal λ sei

$$\bar{\lambda} := \begin{cases} \neg X\,, & \text{falls } \lambda \text{ von der Form } X \text{ für ein } X \in \text{AS ist} \\ X\,, & \text{falls } \lambda \text{ von der Form } \neg X \text{ für ein } X \in \text{AS ist.} \end{cases}$$

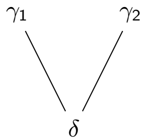
Wir nennen $\overline{\lambda}$ auch das Negat von λ.

Definition 2.56 (Resolutionsregel)

Seien γ_1, γ_2 und δ endliche Mengen von Literalen (d.h. disjunktive Klauseln). Dann ist δ eine Resolvente von γ_1 und γ_2, wenn es ein Literal λ gibt, so dass gilt:

$$\lambda \in \gamma_1, \qquad \overline{\lambda} \in \gamma_2 \qquad \text{und} \qquad \delta = \ \big(\gamma_1 \setminus \{\lambda\}\big) \ \cup \ \big(\gamma_2 \setminus \{\overline{\lambda}\}\big).$$

Graphische Darstellung:



„δ ist eine Resolvente von γ_1 und γ_2.“

Das Resolutionslemma

Notation

Ein Klausel ist eine endliche Menge von Literalen (eine solche Klausel repräsentiert die Disjunktion der in ihr enthaltenen Literale).
Eine Klauselmenge ist eine (endliche oder unendliche) Menge von Klauseln.

Lemma 2.57 (Resolutionslemma)

Sei Γ eine Klauselmenge, seien $\gamma_1, \gamma_2 \in \Gamma$ und sei δ eine Resolvente von γ_1 und γ_2. Dann sind die Klauselmengen Γ und $\Gamma \cup \{\delta\}$ äquivalent.

Resolutionsableitungen und -widerlegungen

Definition

Sei Γ eine Klauselmenge.

(a) Eine Resolutionsableitung einer Klausel δ aus Γ ist ein Tupel $(\delta_1, \ldots, \delta_\ell)$ von Klauseln, so dass gilt: $\ell \geqslant 1$, $\delta_\ell = \delta$, und für alle $i \in [\ell]$ ist

- $\delta_i \in \Gamma$, oder
- es gibt $j, k \in [i{-}1]$, so dass δ_i eine Resolvente von δ_j und δ_k ist.

(b) Eine Resolutionswiderlegung von Γ ist eine Resolutionsableitung der leeren Klausel aus Γ.

Zur Erinnerung:

Eine Klausel δ ist genau dann eine Resolvente zweier Klauseln γ_1 und γ_2, wenn es ein Literal λ gibt, so dass gilt:

$$\lambda \in \gamma_1, \qquad \overline{\lambda} \in \gamma_2 \qquad \text{und} \qquad \delta = \big(\gamma_1 \setminus \{\lambda\}\big) \cup \big(\gamma_2 \setminus \{\overline{\lambda}\}\big).$$

Notation 2.58

(a) Wir schreiben kurz $\Gamma \vdash_R \delta$ um auszudrücken, dass es eine Resolutionsableitung von δ aus Γ gibt.

Insbesondere bedeutet $\Gamma \vdash_R \emptyset$, dass es eine Resolutionswiderlegung von Γ gibt.

(b) An Stelle von $(\delta_1, \ldots, \delta_\ell)$ schreiben wir Resolutionsableitungen der besseren Lesbarkeit halber oft zeilenweise, also

$$\begin{array}{cl} (1) & \delta_1 \\ (2) & \delta_2 \\ \vdots & \\ (\ell) & \delta_\ell \end{array}$$

und geben am Ende jeder Zeile eine kurze Begründung an.

Beispiel 2.59

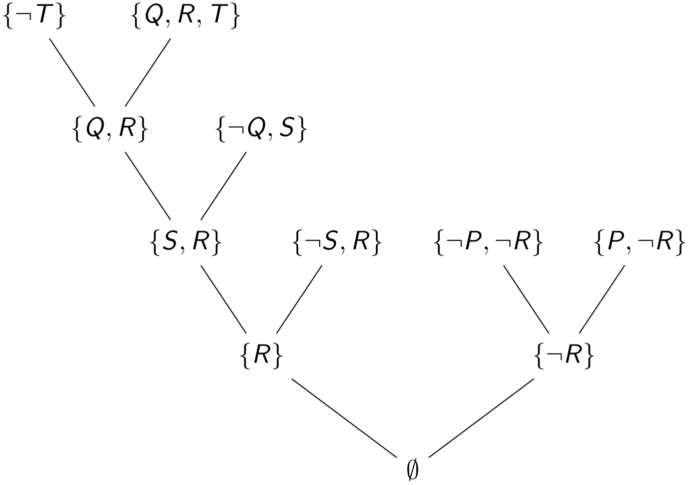
Sei

$$\Gamma \ := \ \{ \ \{\neg P, \neg R\}, \ \{P, \neg R\}, \ \{\neg Q, S\}, \ \{Q, R, T\}, \ \{\neg T\}, \ \{\neg S, R\} \ \}$$

Eine Resolutionswiderlegung von Γ ist:

$$\begin{array}{rll}
(1) & \{\neg T\} & \text{(in } \Gamma\text{)} \\
(2) & \{Q, R, T\} & \text{(in } \Gamma\text{)} \\
(3) & \{Q, R\} & \text{(Resolvente von (1), (2))} \\
(4) & \{\neg Q, S\} & \text{(in } \Gamma\text{)} \\
(5) & \{S, R\} & \text{(Resolvente von (3), (4))} \\
(6) & \{\neg S, R\} & \text{(in } \Gamma\text{)} \\
(7) & \{R\} & \text{(Resolvente von (5), (6))} \\
(8) & \{\neg P, \neg R\} & \text{(in } \Gamma\text{)} \\
(9) & \{P, \neg R\} & \text{(in } \Gamma\text{)} \\
(10) & \{\neg R\} & \text{(Resolvente von (8), (9))} \\
(11) & \emptyset & \text{(Resolvente von (7), (10))}
\end{array}$$

Graphische Darstellung der Resolutionswiderlegung



Korrektheit und Vollständigkeit der Resolution

Satz 2.60

Für jede Klauselmenge Γ *gilt:*

$$\Gamma \vdash_R \emptyset \quad \Longleftrightarrow \quad \Gamma \text{ ist unerfüllbar.}$$

D.h.: Eine Klauselmenge hat genau dann eine Resolutionswiderlegung, wenn sie unerfüllbar ist.

Vorsicht

Beim Anwenden der Resolutionsregel (Definition 2.56) darf immer nur ein Literal λ betrachtet werden.

Beispiel:

Betrachte die Klauselmenge $\Gamma := \{\gamma_1, \gamma_2\}$ mit $\gamma_1 := \{X, Y\}$ und $\gamma_2 := \{\neg X, \neg Y\}$ (wobei X und Y zwei verschiedene Ausagensymbole sind).

Der Satz von Haken

Für eine endliche Klauselmenge Γ sei die *Größe* von Γ die Zahl

$$\|\Gamma\| \;:=\; \sum_{\gamma\in\Gamma} |\gamma|,$$

wobei $|\gamma|$ die Anzahl der Literale in γ bezeichnet.

Der folgende (schwer zu beweisende) Satz zeigt, dass es im Worst-Case exponentiell lange dauern kann, eine Resolutionswiderlegung zu finden.

Satz 2.61 (Satz von Haken, 1985)

Es gibt Konstanten $c, d > 0$ und endliche Klauselmengen Γ_n für $n \geqslant 1$, so dass für alle $n \in \mathbb{N}$ mit $n \geqslant 1$ gilt:

1. $\|\Gamma_n\| \leqslant n^c$,
2. Γ_n *ist unerfüllbar, und*
3. *jede Resolutionswiderlegung von Γ_n hat Länge $\geqslant 2^{dn}$.*

(Hier ohne Beweis)

Abschnitt 2.7:

Erfüllbarkeitsalgorithmen

Das aussagenlogische Erfüllbarkeitsproblem

Wir betrachten im Folgenden Algorithmen für das

Aussagenlogische Erfüllbarkeitsproblem:

Eingabe:	eine Formel $\varphi \in$ AL
Ausgabe:	„erfüllbar“, falls φ erfüllbar ist;
	„unerfüllbar“, sonst.

Notation

Im Folgenden bezeichnet n immer die Anzahl der in φ vorkommenden verschiedenen Aussagensymbole, und $m := |\varphi|$ bezeichnet die Länge von φ (aufgefasst als Wort über dem Alphabet der Aussagenlogik).

Varianten des Erfüllbarkeitsproblems

Berechnen einer erfüllenden Interpretation:

Zusätzlich soll bei erfüllbaren Formeln noch ein Modell berechnet werden, d.h., ein Tupel $(b_1, \dots, b_n) \in \{0,1\}^n$, so dass $\varphi[b_1, \dots, b_n] = 1$.

Einschränkung auf KNF-Formeln:

Oft beschränkt man sich auf Eingabeformeln in KNF oder sogar 3-KNF. Das ist keine wesentliche Einschränkung, weil sich mit Hilfe des Tseitin-Verfahrens jede Formel in Linearzeit in eine erfüllbarkeitsäquivalente Formel in 3-KNF transformieren lässt (Satz 2.55).
Das Erfüllbarkeitsproblem für Formeln in KNF bzw. 3-KNF bezeichnet man mit SAT bzw. 3-SAT.

Komplexität des Erfüllbarkeitsproblems

Satz 2.62 (Satz von Cook und Levin, ≈1971)

Das aussagenlogische Erfüllbarkeitsproblem (und sogar die Einschränkung 3-SAT) ist NP-vollständig.

Die Komplexitätsklassen P und NP, der Begriff der NP-Vollständigkeit, sowie ein Beweis des Satzes von Cook und Levin werden in der Vorlesung *Einführung in die Theoretische Informatik* behandelt.

Bemerkung

- Wenn also $\mathrm{P} \neq \mathrm{NP}$ ist (was allgemein vermutet wird), gibt es für das aussagenlogische Erfüllbarkeitsproblem keinen Polynomialzeitalgorithmus.
- Man vermutet sogar, dass es eine Konstante $c > 1$ gibt, so dass jeder Algorithmus für 3-SAT eine worst-case Laufzeit von $\Omega(c^n)$ hat.
  Diese Vermutung ist unter dem Namen „Exponential Time Hypothesis“ (ETH) bekannt.
- Der im Worst-Case beste derzeit bekannte Algorithmus für 3-SAT hat eine Laufzeit von etwa $O(1.4^n)$.

Der Wahrheitstafelalgorithmus

Lemma 2.63

Es gibt einen Linearzeitalgorithmus, der bei Eingabe einer Formel $\varphi(A_1, \ldots, A_n) \in \mathrm{AL}$ und eines Tupels $(b_1, \ldots, b_n) \in \{0, 1\}^n$ den Wert $\varphi[b_1, \ldots, b_n]$ berechnet.

Beweis: Übung.

Der folgende Algorithmus löst das aussagenlogische Erfüllbarkeitsproblem.

Wahrheitstafelalgorithmus

Eingabe: eine Formel $\varphi \in \mathrm{AL}$

1. Berechne die Wahrheitstafel für φ.
2. Falls in der letzten Spalte mindestens eine 1 auftaucht, gib „erfüllbar“ aus, sonst gib „unerfüllbar“ aus.

Laufzeit: $O(m \cdot 2^n)$ (sogar im „Best-Case“)

Der Resolutionsalgorithmus

Der Resolutionsalgorithmus probiert einfach alle möglichen Resolutionsableitungen durch und testet so, ob es eine Resolutionswiderlegung gibt (d.h. die Klauselmenge unerfüllbar ist).

Resolutionsalgorithmus

Eingabe: eine endliche Klauselmenge Γ (entspricht einer KNF-Formel)

1. Wiederhole, bis keine neuen Klauseln mehr generiert werden:
   Füge alle Resolventen aller Klauseln aus Γ zu Γ hinzu.
2. Falls $\emptyset \in \Gamma$, gib „unerfüllbar" aus, sonst gib „erfüllbar" aus.

Laufzeit: $2^{O(n)}$ (weil es bei n Aussagensymbolen 4^n verschiedene Klauseln gibt).

Der Davis-Putnam-Logemann-Loveland Algorithmus

Der DPLL-Algorithmus ist ein in der Praxis sehr erfolgreicher Algorithmus, der die Wahrheitstafelmethode mit Resolution kombiniert.

Ähnlich wie bei dem Wahrheitstafelalgorithmus durchsucht der DPLL-Algorithmus systematisch den Raum aller möglichen Interpretationen und testet, ob diese die gegebene Klauselmenge erfüllen. Resolution wird dabei dazu verwendet, die Suche geschickt zu steuern und Dinge, die während der Suche bereits über die Klauselmenge „gelernt" wurden, weiterzuverwenden.

Der DPLL-Algorithmus ist die Basis moderner SAT-Solver, die Klauselmengen, die aus Millionen von Klauseln und Hunderttausenden von Aussagensymbolen bestehen, auf Erfüllbarkeit testen können.

DPLL-Algorithmus

Eingabe: eine endliche Klauselmenge Γ (entspricht einer KNF-Formel)

1. Vereinfache Γ. *% Details dazu: siehe nächste Folie*
2. Falls $\Gamma = \emptyset$, gib „erfüllbar“ aus.
3. Falls $\emptyset \in \Gamma$, gib „unerfüllbar“ aus.
4. Wähle ein Literal λ.
5. *% probiere aus, ob Γ ein Modell hat, bei dem das Literal λ erfüllt wird:*
   Löse rekursiv $\Gamma \cup \big\{\{\lambda\}\big\}$. Falls dies erfüllbar ist, gib „erfüllbar“ aus.
6. *% probiere aus, ob Γ ein Modell hat, bei dem das Literal $\overline{\lambda}$ erfüllt wird:*
   Löse rekursiv $\Gamma \cup \big\{\{\overline{\lambda}\}\big\}$. Falls dies erfüllbar ist, gib „erfüllbar“ aus. Sonst gib „unerfüllbar“ aus.

Vereinfachungsheuristiken, die in Schritt 1. angewendet werden:

- Unit Propagation: Für alle „Einerklauseln" $\{\lambda\} \in \Gamma$ (wobei λ ein Literal ist), bilde alle Resolventen von $\{\lambda\}$ mit anderen Klauseln und streiche anschließend alle Klauseln, die λ enthalten. Wiederhole dies, so lange es Einerklauseln gibt.

  Präzise:
  Für jede „Einerklausel" $\{\lambda\} \in \Gamma$ tue Folgendes:
  1. Ersetze jede Klausel $\gamma \in \Gamma$ durch die Klausel $\gamma \setminus \{\overline{\lambda}\}$.
  2. Entferne aus Γ jede Klausel, die das Literal λ enthält.

  Wiederhole dies, so lange es in Γ Einerklauseln gibt.

- Pure Literal Rule: Literale λ, deren Negat $\overline{\lambda}$ nirgendwo in der Klauselmenge auftaucht, können auf 1 gesetzt werden. Alle Klauseln, die ein solches Literal enthalten, sind dann wahr und können gestrichen werden.

- Streiche Klauseln, die Obermengen von anderen Klauseln sind (dies ist allerdings ineffizient und wird in der Praxis zumeist weggelassen).

Man sieht leicht, dass der DPLL-Algorithmus stets die korrekte Antwort gibt (d.h., er terminiert immer, und er gibt genau dann „erfüllbar" aus, wenn die eingegebene Klauselmenge Γ erfüllbar ist).

Laufzeit des DPLL-Algorithmus:

$O(m \cdot 2^n)$ im Worst-Case, in der Praxis aber häufig sehr effizient.

Beispiel 2.64

Sei $\Gamma :=$

$$\begin{aligned}\{\ &\{X_1, \neg X_5, \neg X_6, X_7\},\ \{\neg X_1, X_2, \neg X_5\},\ \{\neg X_1, \neg X_2, \neg X_3, \neg X_5, \neg X_6\},\\ &\{X_1, X_2, \neg X_4, X_7\},\ \{\neg X_4, \neg X_6, \neg X_7\},\ \{X_3, \neg X_5, X_7\},\\ &\{X_3, \neg X_4, \neg X_5\},\ \{X_5, \neg X_6\},\ \{X_5, X_4, \neg X_8\},\\ &\{X_1, X_3, X_5, X_6, X_7\},\ \{\neg X_7, X_8\},\ \{\neg X_6, \neg X_7, \neg X_8\}\ \}\end{aligned}$$

Abschnitt 2.8:

Hornformeln

Hornklauseln und Hornformeln

Hornformeln sind spezielle aussagenlogische Formeln, die die Basis der logischen Programmierung bilden, und für die das Erfüllbarkeitsproblem effizient gelöst werden kann.

Definition 2.65

Eine Hornklausel ist eine disjunktive Klausel, in der höchstens ein positives Literal vorkommt.
Eine Hornformel ist eine Konjunktion endlich vieler Hornklauseln.

Beispiele

- $\{\neg X, \neg Y, \neg Z\}$ (bzw. $\neg X \vee \neg Y \vee \neg Z$) ist eine Hornklausel.
- $\{\neg X, \neg Y, Z\}$ (bzw. $\neg X \vee \neg Y \vee Z$) ist eine Hornklausel.
- $\{\neg X, Y, Z\}$ (bzw. $\neg X \vee Y \vee Z$) ist <u>keine</u> Hornklausel.
- $\{X\}$ (bzw. X) ist eine Hornklausel.
- $\emptyset$ ist eine Hornklausel.
- $(X \vee \neg Y) \wedge (\neg Z \vee \neg X \vee \neg Y) \wedge Y$ ist eine Hornformel.

Hornklauseln als Implikationen

- Eine Hornklausel der Form $\{\neg X_1, \ldots, \neg X_{n-1}, X_n\}$ (bzw. $\neg X_1 \vee \cdots \vee \neg X_{n-1} \vee X_n$) ist äquivalent zur Formel
$$(X_1 \wedge \cdots \wedge X_{n-1}) \rightarrow X_n.$$
Solche Klauseln werden auch „Regeln“ (oder „Prozedurklauseln“) genannt.

- Eine Hornklausel der Form $\{\neg X_1, \ldots, \neg X_{n-1}\}$ ist äquivalent zur Formel
$$(X_1 \wedge \ldots \wedge X_{n-1}) \rightarrow \mathbf{0}.$$
Solche Klauseln werden auch „Zielklauseln“ (oder „Frageklauseln“) genannt.

- Eine Hornklausel der Form $\{X_1\}$ ist äquivalent zur Formel
$$\mathbf{1} \rightarrow X_1.$$
Solche Klauseln werden auch „Tatsachenklausel“ genannt.

- Die leere (Horn-)Klausel $\emptyset$ ist unerfüllbar und daher äquivalent zur Formel
$$\mathbf{1} \rightarrow \mathbf{0}.$$

Der Streichungsalgorithmus

Der folgende Algorithmus löst das Erfüllbarkeitsproblem für Hornformeln in Polynomialzeit.

Wir geben zunächst den Algorithmus an, betrachten dann Beispielläufe davon, analysieren die Laufzeit und zeigen danach, dass der Algorithmus korrekt ist, d.h. stets die richtige Antwort gibt.

Streichungsalgorithmus

Eingabe: eine endliche Menge Γ von Hornklauseln

1. Wiederhole:
2. Falls $\emptyset \in \Gamma$, so halte mit Ausgabe „unerfüllbar“.
3. Falls Γ keine Tatsachenklausel (d.h. Klausel $\{X\}$ mit $X \in$ AS) enthält, so halte mit Ausgabe „erfüllbar“.
   % Γ wird erfüllt, indem jedes Aussagensymbol mit 0 belegt wird
4. Wähle eine Tatsachenklausel $\{X\} \in \Gamma$.
   % Idee: Um Γ zu erfüllen, muss X mit dem Wert 1 belegt werden
5. Streiche $\neg X$ aus allen Klauseln $\delta \in \Gamma$, die das Literal $\neg X$ enthalten.
   % Wenn X den Wert 1 hat, trägt $\neg X$ nichts zum Erfüllen einer Klausel bei
6. Streiche aus Γ alle Klauseln $\delta \in \Gamma$, die das Literal X enthalten (d.h. entferne aus Γ alle $\delta \in \Gamma$, für die gilt: $X \in \delta$).
   % Wenn X den Wert 1 hat, sind solche Klauseln erfüllt

Beispiele 2.66

Wir wenden den Streichungsalgorithmus auf die beiden folgenden Mengen von Hornklauseln an.

(a) $\Gamma_a := \ \{ \ S \to \mathbf{0}, \ \ (P \wedge Q) \to R, \ \ (S \wedge R) \to \mathbf{0}, \ \ (U \wedge T \wedge Q) \to P, \ \ (U \wedge T) \to Q, \ \ \mathbf{1} \to U, \ \ \mathbf{1} \to T \ \}$

(b) $\Gamma_b := \ \{ \ (Q \wedge P) \to T, \ \ (U \wedge T \wedge Q) \to R, \ \ (U \wedge T) \to Q, \ \ \mathbf{1} \to U, \ \ R \to \mathbf{0}, \ \ \mathbf{1} \to T \ \}$

Laufzeit des Streichungsalgorithmus

Man sieht leicht, dass in jedem Schleifendurchlauf die Anzahl der Klauseln in Γ kleiner wird. Daher terminiert der Algorithmus nach maximal m Schleifendurchläufen, wobei m die Anzahl der Klauseln in der Eingabemenge Γ ist.

In jedem einzelnen Schleifendurchlauf betrachtet der Algorithmus alle Klauseln der aktuellen Klauselmenge und führt dabei $O(n)$ Schritte durch, wobei $n = \|\Gamma\|$ die Größe der Klauselmenge ist.

Insgesamt terminiert der Streichungsalgorithmus also nach $O(m \cdot n)$ Schritten, d.h. in Zeit polynomiell in der Größe von Γ.

Satz 2.67

Die Laufzeit des Streichungsalgorithmus ist $O(m \cdot n)$, wobei $m = |\Gamma|$ die Anzahl der Hornklauseln in der eingegebenen Menge Γ und $n = \|\Gamma\|$ die Größe von Γ ist.

Bemerkung

Eine Variante des Streichungsalgorithmus läuft sogar in Linearzeit, d.h. in Zeit $O(n)$.

Der Streichungsalgorithmus und Resolution

Lemma 2.68

Sei Γ_0 eine endliche Menge von Hornklauseln und δ eine Klausel, die zu irgendeinem Zeitpunkt während des Laufs des Streichungsalgorithmus bei Eingabe Γ_0 in der vom Algorithmus gespeicherten Menge Γ liegt. Dann gilt: $\Gamma_0 \vdash_R \delta$.

Korrektheit des Streichungsalgorithmus

Satz 2.69

Der Streichungsalgorithmus ist korrekt.

Das heißt, bei Eingabe einer endlichen Menge Γ_0 von Hornklauseln hält der Algorithmus mit Ausgabe „erfüllbar", falls Γ_0 erfüllbar ist, und mit Ausgabe „nicht erfüllbar", falls Γ_0 unerfüllbar ist.

Kapitel 3:

Logik erster Stufe

Abschnitt 3.1:

Strukturen

Strukturen

Wir führen einen allgemeinen Strukturbegriff ein, der es uns erlaubt:

- mathematische Strukturen wie Gruppen, Körper, Vektorräume, Graphen, etc.
- und die gängigen Modelle der Informatik wie Transitionssysteme, endliche Automaten, relationale Datenbanken, Schaltkreise, etc.

zu beschreiben.

Signaturen

Definition 3.1

Eine Signatur (auch Vokabular oder Symbolmenge) ist eine Menge σ von Relations-, Funktions- und/oder Konstantensymbolen.

Jedes Relationsymbol $R \in \sigma$ und jedes Funktionssymbol $f \in \sigma$ hat eine Stelligkeit (bzw. Arität, engl. arity)

$$\mathrm{ar}(R) \in \mathbb{N} \setminus \{0\} \qquad \text{bzw.} \qquad \mathrm{ar}(f) \in \mathbb{N} \setminus \{0\}.$$

Notation

- In diesem Kapitel bezeichnet der griechische Buchstabe σ (in Worten: sigma) immer eine Signatur.

- Für Relationssymbole verwenden wir normalerweise Großbuchstaben wie R, P, Q, E, für Funktionsymbole verwenden wir meistens Kleinbuchstaben wie f, g, h und für Konstantensymbole Kleinbuchstaben wie c, d.

- Gelegentlich verwenden wir als Relations- und Funktionssymbole auch Zeichen wie $\leqslant$ (2-stelliges Relationssymbol) und $+, \cdot$ (2-stellige Funktionssymbole), und wir verwenden $\underline{0}, \underline{1}$ als Konstantensymbole.

- Die Stelligkeit eines Relations- oder Funktionssymbols deuten wir häufig an, indem wir sie mit Schrägstrich hinter das Symbol schreiben.

Beispiel

Die Notation $R/2$ deutet an, dass R ein 2-stelliges Relationssymbol ist.

Strukturen

Definition 3.2

Eine σ-Struktur $\mathcal{A}$ besteht aus folgenden Komponenten:

- einer nicht-leeren Menge A, dem Universum von $\mathcal{A}$ (auch: Träger, engl. universe, domain),
- für jedes Relationssymbol $R \in \sigma$ und für $k := \text{ar}(R)$ gibt es eine k-stellige Relation $R^{\mathcal{A}} \subseteq A^k$,
- für jedes Funktionssymbol $f \in \sigma$ und für $k := \text{ar}(f)$ gibt es eine k-stellige Funktion $f^{\mathcal{A}} : A^k \to A$, und
- für jedes Konstantensymbol $c \in \sigma$ gibt es ein Element $c^{\mathcal{A}} \in A$.

Notation

- Wir beschreiben σ-Strukturen oft in Tupelschreibweise:
  $\mathcal{A} = \big(A,\ (S^{\mathcal{A}})_{S\in\sigma}\big)$.

  Falls $\sigma = \{S_1, \ldots, S_k\}$ endlich ist, schreiben wir auch
  $\mathcal{A} = \big(A,\ S_1^{\mathcal{A}}, \ldots, S_k^{\mathcal{A}}\big)$.

- Wir bezeichnen σ-Strukturen meistens mit „kalligraphischen“ Buchstaben wie $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{W}, \ldots$. Das Universum der Strukturen bezeichnen wir dann durch die entsprechenden lateinischen Großbuchstaben, also $A, B, C, W, \ldots$.

Mengen

Für die leere Signatur $\sigma := \emptyset$ bestehen σ-Strukturen nur aus ihrem Universum, sind also einfach (nicht-leere) Mengen.

Graphen

In diesem Kapitel bezeichnet E immer ein zweistelliges Relationssymbol.

- Ein gerichteter Graph (kurz: Digraph) $\mathcal{G} = (V^{\mathcal{G}}, E^{\mathcal{G}})$ mit Knotenmenge $V^{\mathcal{G}}$ und Kantenmenge $E^{\mathcal{G}}$ ist eine $\{E\}$-Struktur. Das Universum ist die Knotenmenge $V^{\mathcal{G}}$.

- Einen ungerichteten Graphen $\mathcal{G} = (V^{\mathcal{G}}, E^{\mathcal{G}})$ mit Knotenmenge $V^{\mathcal{G}}$ und Kantenmenge $E^{\mathcal{G}}$ repräsentieren wir durch eine $\{E\}$-Struktur $\mathcal{A} = (A, E^{\mathcal{A}})$ mit Universum $A = V^{\mathcal{G}}$ und Relation $E^{\mathcal{A}} = \{(u, v) \ : \ \{u, v\} \in E^{\mathcal{G}}\}$. Insbesondere ist $E^{\mathcal{A}}$ *symmetrisch*.

Eigenschaften zweistelliger Relationen

Definition 3.3

Sei $\mathcal{A} = (A, R^{\mathcal{A}})$, wobei $R^{\mathcal{A}}$ eine zweistellige Relation über der Menge A ist (d.h. $(A, R^{\mathcal{A}})$ ist ein gerichteter Graph).

(a) $R^{\mathcal{A}}$ heißt reflexiv, wenn für alle $a \in A$ gilt: $(a, a) \in R^{\mathcal{A}}$.
$R^{\mathcal{A}}$ heißt irreflexiv, wenn für alle $a \in A$ gilt: $(a, a) \notin R^{\mathcal{A}}$.

(b) $R^{\mathcal{A}}$ heißt symmetrisch, wenn für alle $a, b \in A$ gilt:

Wenn $(a, b) \in R^{\mathcal{A}}$, dann ist auch $(b, a) \in R^{\mathcal{A}}$.

$R^{\mathcal{A}}$ heißt antisymmetrisch, wenn für alle $a, b \in A$ mit $a \neq b$ gilt:

Wenn $(a, b) \in R^{\mathcal{A}}$, dann $(b, a) \notin R^{\mathcal{A}}$.

(c) $R^{\mathcal{A}}$ heißt transitiv, wenn für alle $a, b, c \in A$ gilt:

Wenn $(a, b) \in R^{\mathcal{A}}$ und $(b, c) \in R^{\mathcal{A}}$, dann auch $(a, c) \in R^{\mathcal{A}}$.

(d) $R^{\mathcal{A}}$ heißt konnex, wenn für alle $a, b \in A$ gilt:

$$(a, b) \in R^{\mathcal{A}} \text{ oder } (b, a) \in R^{\mathcal{A}} \text{ oder } a = b.$$

Äquivalenzrelationen

Eine Äquivalenzrelation auf eine Menge A ist eine 2-stellige Relation über A, die reflexiv, transitiv und symmetrisch ist.

Beispiele

(a) Gleichheit: Für jede Menge M ist $\{(m,m) \;:\; m \in M\}$ eine Äquivalenzrelation auf M.

(b) Gleichmächtigkeit: Für jede endliche Menge M und deren Potenzmenge $\mathcal{P}(M)$ gilt: $\{(A,B) \;:\; A,B \subseteq M,\ |A| = |B|\}$ ist eine Äquivalenzrelation auf $\mathcal{P}(M)$.

(c) Logische Äquivalenz: Die Relation $\{\,(\varphi,\psi) \;:\; \varphi,\psi \in \mathsf{AL},\ \varphi \equiv \psi\,\}$ ist eine Äquivalenzrelation auf der Menge AL aller aussagenlogischen Formeln.

Ordnungen

In diesem Kapitel bezeichnet $\leqslant$ sei immer ein zweistelliges Relationssymbol. Für $\leqslant$ verwenden wir Infixschreibweise, d.h., wir schreiben $x \leqslant^{\mathcal{A}} y$ statt $(x,y) \in \leqslant^{\mathcal{A}}$.

(a) Eine Präordnung ist eine $\{\leqslant\}$-Struktur $\mathcal{A} = (A, \leqslant^{\mathcal{A}})$, bei der $\leqslant^{\mathcal{A}}$ reflexiv und transitiv ist.

(b) Eine partielle Ordnung (oder Halbordnung) ist eine Präordnung $\mathcal{A}$, bei der $\leqslant^{\mathcal{A}}$ antisymmetrisch ist.

(c) Eine lineare (oder totale) Ordnung ist eine partielle Ordnung $\mathcal{A}$, bei der $\leqslant^{\mathcal{A}}$ konnex ist.

Beispiele

(a) Die „kleiner-gleich" Relation auf $\mathbb{N}$ (oder $\mathbb{Z}$ oder $\mathbb{R}$) ist eine lineare Ordnung; die „größer-gleich" auch.

(b) Für jede Menge M ist die Teilmengenrelation $\subseteq$ eine partielle Ordnung auf der Potenzmenge $\mathcal{P}(M)$; aber keine lineare Ordnung, sofern M mindestens zwei Elemente besitzt. Dasselbe gilt für die Obermengenrelation $\supseteq$.

(c) Die Folgerungsrelation für aussagenlogische Formeln:
$\{(\varphi,\psi) \ : \ \varphi,\psi \in \mathsf{AL},\ \varphi \models \psi\}$ ist eine Präordnung auf der Menge AL, aber keine partielle Ordnung.

Arithmetische Strukturen

$+$ und $\cdot$ seien immer zweistellige Funktionssymbole, für die wir Infixschreibweise verwenden. $\underline{0}$ und $\underline{1}$ seien Konstantensymbole.

- Der Körper der reellen Zahlen ist die $\{+, \cdot, \underline{0}, \underline{1}\}$-Struktur $\mathcal{A}_\mathbb{R}$, so dass $A_\mathbb{R} := \mathbb{R}$, $+^{\mathcal{A}_\mathbb{R}}$ und $\cdot^{\mathcal{A}_\mathbb{R}}$ sind die normale Addition bzw. Multiplikation auf $\mathbb{R}$, und $\underline{0}^{\mathcal{A}_\mathbb{R}} := 0$, $\underline{1}^{\mathcal{A}_\mathbb{R}} := 1$.

- Der Ring der ganzen Zahlen ist die $\{+, \cdot, \underline{0}, \underline{1}\}$-Struktur $\mathcal{A}_\mathbb{Z}$, so dass $A_\mathbb{Z} := \mathbb{Z}$, $+^{\mathcal{A}_\mathbb{Z}}$ und $\cdot^{\mathcal{A}_\mathbb{Z}}$ sind die normale Addition bzw. Multiplikation auf $\mathbb{Z}$, und $\underline{0}^{\mathcal{A}_\mathbb{Z}} := 0$, $\underline{1}^{\mathcal{A}_\mathbb{Z}} := 1$.

- Das Standardmodell der Arithmetik ist die $\{+, \cdot, \leqslant, \underline{0}, \underline{1}\}$-Struktur $\mathcal{A}_\mathbb{N}$, so dass $A_\mathbb{N} := \mathbb{N}$ ist; die Funktionen $+^{\mathcal{A}_\mathbb{N}}$ und $\cdot^{\mathcal{A}_\mathbb{N}}$ und die Relation $\leqslant^{\mathcal{A}_\mathbb{N}}$ sind die normale Addition, Multiplikation bzw. Ordnung auf $\mathbb{N}$, und $\underline{0}^{\mathcal{A}_\mathbb{N}} := 0$, $\underline{1}^{\mathcal{A}_\mathbb{N}} := 1$.

- Der zweielementige Körper ist die $\{+, \cdot, \underline{0}, \underline{1}\}$-Struktur $\mathcal{F}_2$ mit Universum $F_2 := \{0, 1\}$, den Funktionen $+^{\mathcal{F}_2}$ und $\cdot^{\mathcal{F}_2}$ der Addition bzw. Multiplikation modulo 2, und $\underline{0}^{\mathcal{F}_2} := 0$, $\underline{1}^{\mathcal{F}_2} := 1$.

Wörter als Strukturen

Sei Σ ein endliches, nicht-leeres Alphabet. Für jedes $a \in \Sigma$ sei P_a ein einstelliges Relationssymbol, und es sei

$$\sigma_\Sigma \quad := \quad \{\leqslant\} \cup \{P_a : a \in \Sigma\}.$$

Für jedes nicht-leere Wort $w := w_1 \cdots w_n \in \Sigma^*$ mit $w_1, \ldots, w_n \in \Sigma$ sei $\mathcal{A}_w$ die σ_Σ-Struktur

- mit Universum $A_w := [n]$, für die gilt:
- $\leqslant^{\mathcal{A}_w}$ ist die natürliche lineare Ordnung auf $[n]$, d.h., $\leqslant^{\mathcal{A}_w} = \{\ (i,j)\ :\ i,j \in \mathbb{N},\ 1 \leqslant i \leqslant j \leqslant n\ \}$,
- Für jedes $a \in \Sigma$ ist $P_a^{\mathcal{A}_w} := \{i \in [n] : w_i = a\}$.

Beispiel

Sei $\Sigma := \{a, b, c\}$. Für $w := abacaba$ ist $\mathcal{A}_w$ die folgende σ_Σ-Struktur:

- $A_w = \{1, 2, 3, 4, 5, 6, 7\}$
- $\leqslant^{\mathcal{A}_w} = \ \{\ (i,j)\ :\ i,j \in \mathbb{N},\ 1 \leqslant i \leqslant j \leqslant 7\ \}$
- $P_a^{\mathcal{A}_w} = \{1, 3, 5, 7\}, \quad P_b^{\mathcal{A}_w} = \{2, 6\}, \quad P_c^{\mathcal{A}_w} = \{4\}.$

Wortstrukturen

Eine Wortstruktur über Σ ist eine σ_Σ-Struktur $\mathcal{A}$ für die gilt:

- das Universum A von $\mathcal{A}$ ist endlich,
- $(A, \leqslant^{\mathcal{A}})$ ist eine lineare Ordnung,
- für jedes $i \in A$ gibt es genau ein $a \in \Sigma$, so dass $i \in P_a^{\mathcal{A}}$.

Beispiel 3.4

Sei $\Sigma := \{a, b, c\}$. Die σ_Σ-Struktur $\mathcal{B}$ mit

- Universum $B = \{\diamondsuit, \heartsuit, \spadesuit, \clubsuit\}$,
- linearer Ordnung $\leqslant^{\mathcal{B}}$, die besagt, dass $\diamondsuit < \heartsuit < \spadesuit < \clubsuit$ ist, d.h.
  $\leqslant^{\mathcal{B}} = \{(\diamondsuit,\diamondsuit), (\diamondsuit,\heartsuit), (\diamondsuit,\spadesuit), (\diamondsuit,\clubsuit), (\heartsuit,\heartsuit), (\heartsuit,\spadesuit), (\heartsuit,\clubsuit), (\spadesuit,\spadesuit), (\spadesuit,\clubsuit), (\clubsuit,\clubsuit)\}$,
- $P_a^{\mathcal{B}} = \{\diamondsuit,\ \clubsuit\}$
- $P_b^{\mathcal{B}} = \{\heartsuit,\ \spadesuit\}$,
- $P_c^{\mathcal{B}} = \emptyset$,

ist eine Wortstruktur, die das Wort $w = abba$ repräsentiert.

Transitionssysteme

- Sei σ_A eine Menge von zweistelligen Relationssymbolen, die wir als Aktionen bezeichnen und σ_P eine Menge von einstelligen Relationssymbolen, die wir als Propositionen oder Eigenschaften bezeichnen.

- Ein (σ_A, σ_P)-Transitionssystem ist eine $(\sigma_A \cup \sigma_P)$-Struktur $\mathcal{T}$.

- Die Elemente des Universums T von $\mathcal{T}$ bezeichnen wir als Zustände des Systems.

- Die Tripel (s, R, t), wobei $(s, t) \in R^{\mathcal{T}}$ für ein $R \in \sigma_A$, bezeichnen wir als die Übergänge oder Transitionen des Systems.

- Sei c ein Konstantensymbol. Ein (σ_A, σ_P)-Transitionssystem mit Anfangszustand ist eine $(\sigma_A \cup \sigma_P \cup \{c\})$-Struktur $\mathcal{T}$. Den Zustand $c^{\mathcal{T}}$ bezeichnen wir als den Anfangszustand des Systems.

Beispiel

Ein endlicher Automat $(Q, \Sigma, q_0, \delta, F)$ lässt sich wie folgt als Transitionssystem $\mathcal{T}$ mit Anfangszustand beschreiben:

- $\sigma_A := \{R_a : a \in \Sigma\}$ und $\sigma_P := \{P_F\}$
- $T := Q$
- für jedes $a \in \Sigma$ ist $R_a^{\mathcal{T}} := \{ (q, q') : q' \in \delta(q, a) \}$
- $P_F^{\mathcal{T}} := F$.
- $c^{\mathcal{T}} := q_0$.

Beispiel

Folgendes Transitionssystem $\mathcal{T}$ mit zwei Aktionen namens `druckauftrag` und `kein_auftrag` und einer Eigenschaft namens `druckt` ist ein stark vereinfachtes Modell des Verhaltens eines Druckers:

- $T := \{\, \mathit{warte},\ \mathit{arbeite} \,\}$,
- $\mathtt{druckauftrag}^{\mathcal{T}} := \{\, (\mathit{warte}, \mathit{arbeite}),\ (\mathit{arbeite}, \mathit{arbeite}) \,\}$,
- $\mathtt{kein\_auftrag}^{\mathcal{T}} := \{\, (\mathit{warte}, \mathit{warte}),\ (\mathit{arbeite}, \mathit{warte}) \,\}$,
- $\mathtt{druckt}^{\mathcal{T}} := \{\, \mathit{arbeite} \,\}$.

Relationale Datenbanken

- Relationale Datenbanken bestehen aus endlich vielen endlichen Tabellen.
- Jede solche Tabelle lässt sich als Relation auffassen, die Zeilen der Tabelle entsprechen dabei den Tupeln in der Relation.
- Eine relationale Datenbank entspricht dann einer endlichen Struktur, deren Universum aus allen potentiellen Einträgen in einzelnen Zellen der Tabellen besteht, und die für jede Tabelle in der Datenbank eine Relation enthält.

Beispiel: Eine Kinodatenbank

Kino			
Name	Adresse	Stadtteil	Telefonnummer
Babylon	Dresdner Str. 126	Kreuzberg	030 61 60 96 93
Casablanca	Friedenstr. 12-13	Adlershof	030 67 75 75 2
Filmtheater am Friedrichshain	Bötzowstr. 1-5	Prenzlauer Berg	030 42 84 51 88
Kino International	Karl-Marx-Allee 33	Mitte	030 24 75 60 11
Moviemento	Kotbusser Damm 22	Kreuzberg	030 692 47 85
Urania	An der Urania 17	Schöneberg	030 21 89 09 1

Film		
Name	Regisseur	Schauspieler
Alien	Ridley Scott	Sigourney Weaver
Blade Runner	Ridley Scott	Harrison Ford
Blade Runner	Ridley Scott	Sean Young
Brazil	Terry Gilliam	Jonathan Pryce
Brazil	Terry Gilliam	Kim Greist
Casablanca	Michael Curtiz	Humphrey Bogart
Casablanca	Michael Curtiz	Ingrid Bergmann
Gravity	Alfonso Cuaron	Sandra Bullock
Gravity	Alfonso Cuaron	George Clooney
Monuments Men	George Clooney	George Clooney
Monuments Men	George Clooney	Matt Damon
Resident Evil	Paul Anderson	Milla Jovovich
Terminator	James Cameron	Arnold Schwarzenegger
Terminator	James Cameron	Linda Hamilton
Terminator	James Cameron	Michael Biehn
...	...	...

Programm		
Kino	Film	Zeit
Babylon	Casablanca	17:30
Babylon	Gravity	20:15
Casablanca	Blade Runner	15:30
Casablanca	Alien	18:15
Casablanca	Blade Runner	20:30
Casablanca	Resident Evil	20:30
Filmtheater am Friedrichshain	Resident Evil	20:00
Filmtheater am Friedrichshain	Resident Evil	21:30
Filmtheater am Friedrichshain	Resident Evil	23:00
Kino International	Casablanca	18:00
Kino International	Brazil	20:00
Kino International	Brazil	22:00
Moviemento	Gravity	17:00
Moviemento	Gravity	19:30
Moviemento	Alien	22:00
Urania	Monuments Men	17:00
Urania	Monuments Men	20:00

Die Kinodatenbank als Struktur

Signatur: $\sigma_{\text{KINO}} := \{ R_{Kino}/4, \ R_{Film}/3, \ R_{Prog}/3 \} \cup \{ \text{'}c\text{'} : c \in \texttt{ASCII}^* \}$

Die Kinodatenbank wird dargestellt als σ_{KINO}-Struktur $\mathcal{D}$.

Universum:

$$D := \texttt{ASCII}^* \supseteq \{ \text{Babylon, Dresdner Str. 126, Kreuzberg, 030 61 60 96 93, Casablanca, } \dots \text{, 20:00} \}.$$

Relationen:

$$\begin{aligned}
R^{\mathcal{D}}_{Kino} := \{ & (\text{Babylon, Dresdner Str. 126, Kreuzberg, 030 61 60 96 93}), \\
& (\text{Casablanca, Friedenstr. 12-13, Adlershof, 030 67 75 75 2}), \\
& \dots, \\
& (\text{Urania, An der Urania 17, Schöneberg, 030 21 89 09 1}) \ \}
\end{aligned}$$

$$\begin{aligned}
R^{\mathcal{D}}_{Film} := \{ & (\text{Alien, Ridley Scott, Sigourney Weaver}), \\
& (\text{Blade Runner, Ridley Scott, Harrison Ford}), \quad \dots \quad \}
\end{aligned}$$

$$\begin{aligned}
R^{\mathcal{D}}_{Prog} := \{ & (\text{Babylon, Casablanca, 17:30}), \\
& (\text{Babylon, Gravity, 20:15}), \quad \dots \quad \}.
\end{aligned}$$

Konstanten: $\text{'}c\text{'}^{\mathcal{D}} := c$, für jedes $c \in \texttt{ASCII}^*$.

D.h.: jedes Konstantensymbol wird durch den zwischen den Hochkommas stehenden Text interpretiert.

Restriktionen und Expansionen

Definition 3.5

Seien σ und τ Signaturen mit $\sigma \subseteq \tau$.

(a) Die σ-Restriktion einer τ-Struktur $\mathcal{B}$ ist die σ-Struktur $\mathcal{B}|_\sigma$ mit $B|_\sigma := B$ und $S^{\mathcal{B}|_\sigma} := S^{\mathcal{B}}$ für jedes $S \in \sigma$.

D.h.: Ist $\mathcal{B} = \big(B, (S^{\mathcal{B}})_{S\in\tau}\big)$, so ist $\mathcal{B}|_\sigma = \big(B, (S^{\mathcal{B}})_{S\in\sigma}\big)$.

(b) Eine τ-Struktur $\mathcal{B}$ ist eine τ-Expansion einer σ-Struktur $\mathcal{A}$, wenn $\mathcal{A} = \mathcal{B}|_\sigma$.

Beispiel

Die $\{+, \underline{0}\}$-Restriktion des Standardmodells der Arithmetik ist die Struktur

$$\mathcal{A}_{\mathbb{N}}|_{\{+,\underline{0}\}} \quad = \quad (\mathbb{N}, +^{\mathcal{A}_{\mathbb{N}}}, \underline{0}^{\mathcal{A}_{\mathbb{N}}}),$$

wobei $+^{\mathcal{A}_{\mathbb{N}}}$ die natürliche Addition auf $\mathbb{N}$ und $\underline{0}^{\mathcal{A}_{\mathbb{N}}}$ die natürliche Zahl 0 ist.

Man bezeichnet diese Struktur als das Standardmodell der Presburger Arithmetik.

Prinzipielle Gleichheit von Strukturen

Frage: Wann sind zwei σ-Strukturen $\mathcal{A}$ und $\mathcal{B}$ „prinzipiell gleich“?

Antwort: Wenn $\mathcal{B}$ aus $\mathcal{A}$ entsteht, indem man die Elemente des Universums von $\mathcal{A}$ umbenennt.

Dies wird in der folgenden Definition präzisiert.

Isomorphismen

Definition 3.6

Seien $\mathcal{A}$ und $\mathcal{B}$ σ-Strukturen. Ein Isomorphismus von $\mathcal{A}$ nach $\mathcal{B}$ ist eine Abbildung $\pi : A \to B$ mit folgenden Eigenschaften:

1. π ist bijektiv.
2. Für alle $k \in \mathbb{N} \setminus \{0\}$, alle k-stelligen Relationssymbole $R \in \sigma$ und alle k-Tupel $(a_1, \ldots, a_k) \in A^k$ gilt:
$$(a_1, \ldots, a_k) \in R^{\mathcal{A}} \iff \big(\pi(a_1), \ldots, \pi(a_k)\big) \in R^{\mathcal{B}}.$$
3. Für alle Konstantensymbole $c \in \sigma$ gilt:
$$\pi\big(c^{\mathcal{A}}\big) = c^{\mathcal{B}}.$$
4. Für alle $k \in \mathbb{N} \setminus \{0\}$, alle k-stelligen Funktionssymbole $f \in \sigma$ und alle k-Tupel $(a_1, \ldots, a_k) \in A^k$ gilt:
$$\pi\big(f^{\mathcal{A}}(a_1, \ldots, a_k)\big) = f^{\mathcal{B}}\big(\pi(a_1), \ldots, \pi(a_k)\big).$$

Isomorphie

Notation

Seien $\mathcal{A}$ und $\mathcal{B}$ σ-Strukturen. Wir schreiben $\pi : \mathcal{A} \cong \mathcal{B}$, um anzudeuten, dass π ein Isomorphismus von $\mathcal{A}$ nach $\mathcal{B}$ ist.

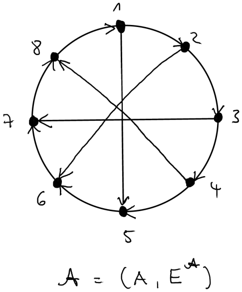
Definition 3.7

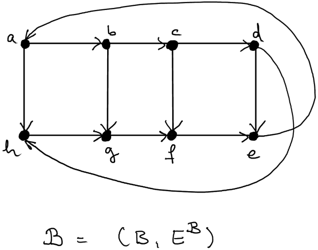
Zwei σ-Strukturen $\mathcal{A}$ und $\mathcal{B}$ heißen isomorph (wir schreiben: $\mathcal{A} \cong \mathcal{B}$), wenn es einen Isomorphismus von $\mathcal{A}$ nach $\mathcal{B}$ gibt.

Beispiele 3.8

(a) Seien A, B nicht-leere Mengen. Dann sind die $\emptyset$-Strukturen $\mathcal{A} := (A)$ und $\mathcal{B} := (B)$ genau dann isomorph, wenn A und B gleichmächtig sind (d.h. es gibt eine Bijektion von A nach B).

(b) Seien $\mathcal{A} = (A, E^{\mathcal{A}})$ und $\mathcal{B} = (B, E^{\mathcal{B}})$ die beiden folgenden Digraphen:





Dann ist $\pi : A \to B$ mit

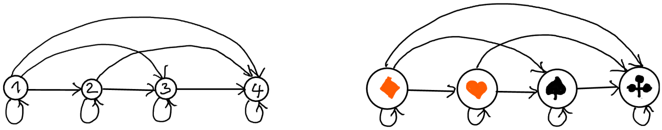
i	1	2	3	4	5	6	7	8
$\pi(i)$	a	b	c	d	h	g	f	e

ein Isomorphismus von $\mathcal{A}$ nach $\mathcal{B}$.

(c) Sei $\mathcal{A} = (A, \leqslant^{\mathcal{A}})$ mit $A = \{1,2,3,4\}$ und

$$\leqslant^{\mathcal{A}} \;=\; \{\,(i,j)\ :\ i,j \in \mathbb{N},\ 1 \leqslant i \leqslant j \leqslant 4\,\},$$

und sei $\mathcal{B} = (B, \leqslant^{\mathcal{A}})$ mit $B = \{\diamondsuit, \heartsuit, \spadesuit, \clubsuit\}$, wobei $\leqslant^{\mathcal{B}}$ wie in Beispiel 3.4 definiert ist. Skizze:



Dann ist $\pi : A \to B$ mit

i	1	2	3	4
$\pi(i)$	$\diamondsuit$	$\heartsuit$	$\spadesuit$	$\clubsuit$

ein Isomorphismus von $\mathcal{A}$ nach $\mathcal{B}$.

Allgemein gilt: Sind A und B endliche Mengen mit $|A| = |B|$, und sind $\leqslant^{\mathcal{A}}$ und $\leqslant^{\mathcal{B}}$ lineare Ordnungen auf A und B, so ist die Abbildung $\pi : A \to B$, die das (bzgl. $\leqslant^{\mathcal{A}}$) kleinste Element in A auf das (bzgl. $\leqslant^{\mathcal{B}}$) kleinste Element in B abbildet, und allgemein für jedes $i \in \{1, \ldots, |A|\}$ das (bzgl. $\leqslant^{\mathcal{A}}$) i-kleinste Element in A auf das (bzgl. $\leqslant^{\mathcal{B}}$) i-kleinste Element in B abbildet, ein Isomorphismus von $\mathcal{A} := (A, \leqslant^{\mathcal{A}})$ nach $\mathcal{B} := (B, \leqslant^{\mathcal{B}})$.

(d) Sind $\leqslant^{\mathbb{N}}$ und $\leqslant^{\mathbb{Z}}$ die natürlichen linearen Ordnungen auf $\mathbb{N}$ und $\mathbb{Z}$, so sind die $\{\leqslant\}$-Strukturen $\mathcal{N} := (\mathbb{N}, \leqslant^{\mathbb{N}})$ und $\mathcal{Z} := (\mathbb{Z}, \leqslant^{\mathbb{Z}})$ nicht isomorph (kurz: $\mathcal{N} \not\cong \mathcal{Z}$).

Beweis: Angenommen, $\pi : \mathbb{N} \to \mathbb{Z}$ ist ein Isomorphismus von $\mathcal{N}$ nach $\mathcal{Z}$. Sei $z := \pi(0)$. In $\mathbb{Z}$ gibt es ein Element $z' \in \mathbb{Z}$ mit $z' < z$ (z.B. $z' = z - 1$). Da π surjektiv ist, muss es ein $n' \in \mathbb{N}$ geben, so dass $\pi(n') = z'$. Wegen $z' \neq z$ muss $n' \neq 0$ gelten (da π injektiv ist). Somit gilt:

$$0 \leqslant^{\mathbb{N}} n' \qquad \text{aber} \qquad z \not\leqslant^{\mathbb{Z}} z'.$$

Also ist π kein Isomorphismus von $\mathcal{N}$ nach $\mathcal{Z}$. Widerspruch!

(e) Sei $\sigma := \{f, c\}$, wobei f ein 2-stelliges Funktionssymbol und c ein Konstantensymbol ist. Sei $\mathcal{A} := (A, f^{\mathcal{A}}, c^{\mathcal{A}})$, wobei gilt:

- $A := \mathbb{N}$ ist die Menge aller natürlichen Zahlen,
- $f^{\mathcal{A}} := +^{\mathcal{A}_{\mathbb{N}}}$ ist die natürliche Addition auf $\mathbb{N}$,
- $c^{\mathcal{A}} := 0$ ist die natürliche Zahl 0

und sei $\mathcal{B} := (B, f^{\mathcal{B}}, c^{\mathcal{B}})$, wobei

- $B := \{2^n \ : \ n \in \mathbb{N}\}$ ist die Menge aller Zweierpotenzen,
- $f^{\mathcal{B}} : B \times B \to B$ ist die Funktion mit

$$f^{\mathcal{B}}(b_1, b_2) \ := \ b_1 \cdot b_2, \qquad \text{für alle } \ b_1, b_2 \in B$$

- $c^{\mathcal{B}} := 1 = 2^0 \in B$.

Dann gilt: $\mathcal{A} \cong \mathcal{B}$, und die Abbildung $\pi : A \to B$ mit $\pi(n) := 2^n$ für alle $n \in \mathbb{N}$ ist ein Isomorphismus von $\mathcal{A}$ nach $\mathcal{B}$, denn:

Isomorphie ist eine Äquivalenzrelation

Lemma 3.9

Isomorphie ist eine Äquivalenzrelation auf der Klasse aller σ-Strukturen. D.h.:
Für alle σ-Strukturen $\mathcal{A}, \mathcal{B}, \mathcal{C}$ gilt:

1. $\mathcal{A} \cong \mathcal{A}$ *(Reflexivität),*
2. $\mathcal{A} \cong \mathcal{B} \implies \mathcal{B} \cong \mathcal{A}$ *(Symmetrie),*
3. $\mathcal{A} \cong \mathcal{B}$ *und* $\mathcal{B} \cong \mathcal{C} \implies \mathcal{A} \cong \mathcal{C}$ *(Transitivität).*

Beweis: Übung.

Abschnitt 3.2:

Terme der Logik erster Stufe

Individuenvariablen

Definition 3.10

Eine Individuenvariable (auch: Variable erster Stufe; kurz: Variable) hat die Form v_i für ein $i \in \mathbb{N}$.

Die Menge aller Variablen bezeichnen wir mit VAR, d.h.

$$\text{VAR} \quad = \quad \{v_0,\ v_1,\ v_2,\ v_3,\ \dots\} \quad = \quad \{v_i \ : \ i \in \mathbb{N}\}.$$

Terme der Logik erster Stufe

Definition 3.11

(a) Für eine Signatur σ sei $A_{\sigma\text{-Terme}}$ das Alphabet, das aus allen Elementen in VAR, allen Konstanten- und Funktionssymbolen in σ, den Klammern (,) und dem Komma , besteht.

(b) Die Menge T_σ aller σ-Terme ist die wie folgt rekursiv definierte Teilmenge von ${A_{\sigma\text{-Terme}}}^*$:

Basisregeln:

- Für jedes Konstantensymbol $c \in \sigma$ ist $c \in \mathrm{T}_\sigma$.
- Für jede Variable $x \in \mathrm{VAR}$ ist $x \in \mathrm{T}_\sigma$.

Rekursive Regel:

- Für jedes Funktionssymbol $f \in \sigma$ und für $k := \mathrm{ar}(f)$ gilt:
  Sind $t_1 \in \mathrm{T}_\sigma, \ldots, t_k \in \mathrm{T}_\sigma$, so ist auch $f(t_1, \ldots, t_k) \in \mathrm{T}_\sigma$.

(c) Die Menge aller Terme der Logik der ersten Stufe ist $\mathrm{T} := \bigcup_{\sigma \text{ Signatur}} \mathrm{T}_\sigma$.

Beispiele

Sei $\sigma := \{ f/2,\ c \}$.

Folgende Worte sind σ-Terme:

$$c, \quad v_4, \quad f(c,c), \quad f(c, f(c, v_0))\,.$$

Folgende Worte sind <u>keine</u> σ-Terme:

$$0, \quad f(0,c), \quad f(v_0, c, v_1), \quad f^{\mathcal{A}}(2,3)\,.$$

Belegungen und Interpretationen

Definition 3.12

Sei σ eine Signatur.

(a) Eine Belegung in einer σ-Struktur $\mathcal{A}$ ist eine Abbildung $\beta : \text{VAR} \to A$.

D.h.: β ordnet jeder Variablen $x \in \text{VAR}$ ein Element $\beta(x)$ aus dem Universum von $\mathcal{A}$ zu.

(b) Eine σ-Interpretation ist ein Paar

$$\mathcal{I} = (\mathcal{A}, \beta),$$

bestehend aus einer σ-Struktur $\mathcal{A}$ und einer Belegung β in $\mathcal{A}$.

Die Auswertung von Termen in Interpretationen

Wir wollen Terme nun in Interpretationen „auswerten“.

Die Auswertung von Term t in einer Interpretation $\mathcal{I} = (\mathcal{A}, \beta)$ soll dasjenige Element aus A liefern, das man erhält, wenn man

- die in t vorkommenden Variablen gemäß der Belegung β interpretiert,
- die in t vorkommenden Konstantensymbole c gemäß ihrer Interpretation $c^{\mathcal{A}}$ in $\mathcal{A}$ belegt,
- die in t vorkommenden Funktionssymbole f gemäß ihrer Interpretation $f^{\mathcal{A}}$ in $\mathcal{A}$ belegt

und dann nach und nach den resultierenden Term ausrechnet.

Dies wird in der folgenden Definition präzisiert.

Semantik von σ-Termen

Definition 3.13

Sei σ eine Signatur. Rekursiv über den Aufbau von T_σ definieren wir eine Funktion $[\![\cdot]\!]^{\cdot}$, die jedem σ-Term t und jeder σ-Interpretation $\mathcal{I} = (\mathcal{A}, \beta)$ einen Wert $[\![t]\!]^{\mathcal{I}} \in A$ zuordnet:

- Für alle $x \in \mathrm{VAR}$ ist $[\![x]\!]^{\mathcal{I}} := \beta(x)$.
- Für alle Konstantensymbole $c \in \sigma$ ist $[\![c]\!]^{\mathcal{I}} := c^{\mathcal{A}}$.
- Für alle Funktionssymbole $f \in \sigma$, für $k := \mathrm{ar}(f)$, und für alle σ-Terme $t_1, \ldots, t_k \in \mathrm{T}_\sigma$ gilt:

$$[\![\, f(t_1, \ldots, t_k)\,]\!]^{\mathcal{I}} \quad := \quad f^{\mathcal{A}}\left([\![t_1]\!]^{\mathcal{I}}, \ldots, [\![t_k]\!]^{\mathcal{I}} \right).$$

Beispiel

Sei $\sigma = \{\ f/2,\ c\ \}$, und sei $\mathcal{A} = (A, f^{\mathcal{A}}, c^{\mathcal{A}})$ die σ-Struktur mit $A = \mathbb{N}$, $f^{\mathcal{A}} = +^{\mathcal{A}_{\mathbb{N}}}$ (die Addition auf den natürlichen Zahlen) und $c^{\mathcal{A}} = 0$ (die natürliche Zahl 0).

Sei $\beta : \mathrm{VAR} \to A$ eine Belegung mit $\beta(v_1) = 1$ und $\beta(v_2) = 7$, und sei $\mathcal{I} := (\mathcal{A}, \beta)$.

Sei t der σ-Term $f(v_2, f(v_1, c))$. Dann gilt:

$$\begin{aligned} [\![t]\!]^{\mathcal{I}} &= f^{\mathcal{A}}\Big(\beta(v_2),\ f^{\mathcal{A}}\big(\beta(v_1),\ c^{\mathcal{A}}\big)\Big) \\ &= f^{\mathcal{A}}\Big(7,\ f^{\mathcal{A}}(1, 0)\Big) \\ &= \Big(7 + \big(1 + 0\big)\Big) \\ &= 8. \end{aligned}$$

Abschnitt 3.3:
Syntax der Logik erster Stufe

Vergleich zwischen Aussagenlogik und Logik erster Stufe

Die Logik erster Stufe übernimmt, verändert und erweitert die Syntax der Aussagenlogik.

- Was gleich bleibt:
  - Die Junktoren $\neg$, $\wedge$, $\vee$, $\rightarrow$ werden übernommen.
- Was sich verändert:
  - Variablen stehen nicht mehr für „wahre" oder „falsche" Ausagen, sondern für Elemente im Universum einer σ-Struktur.
  - Variablen sind keine atomaren Formeln mehr.
- Was neu hinzukommt:
  - Es gibt Quantoren $\exists$ und $\forall$ (für „es existiert" und „für alle").
  - Es gibt Symbole für Elemente aus der Signatur σ.
  - Es können σ-Terme benutzt werden, um Elemente im Universum einer σ-Struktur zu bezeichnen.

Das Alphabet der Logik erster Stufe

Definition 3.14

Sei σ eine Signatur.
Das Alphabet $A_{\mathsf{FO}[\sigma]}$ der Logik erster Stufe über σ besteht aus

- allen Symbolen in $A_{\sigma\text{-Terme}}$,
- allen Symbolen in σ,
- den Quantoren $\exists$ (Existenzquantor) und $\forall$ (Allquantor),
- dem Gleichheitssymbol $=$,
- den Junktoren $\neg, \wedge, \vee, \rightarrow$.

D.h.:

$$A_{\mathsf{FO}[\sigma]} \;=\; \mathsf{VAR} \cup \sigma \cup \{\exists, \forall\} \cup \{=\} \cup \{\neg, \wedge, \vee, \rightarrow\} \cup \{(,)\} \cup \{,\}.$$

Syntax der Logik erster Stufe

Definition 3.15

Sei σ eine Signatur. Die Menge $\mathrm{FO}[\sigma]$ aller Formeln der Logik erster Stufe über der Signatur σ (kurz: $\mathrm{FO}[\sigma]$-Formeln; „FO“ steht für die englische Bezeichnung der Logik erster Stufe: first-order logic) ist die folgendermaßen rekursiv definierte Teilmenge von $A^*_{\mathrm{FO}[\sigma]}$:

Basisregeln:

- Für alle σ-Terme t_1 und t_2 in T_σ gilt:

$$t_1 = t_2 \quad \in \quad \mathrm{FO}[\sigma].$$

- Für jedes Relationssymbol $R \in \sigma$, für $k := \mathrm{ar}(R)$ und für alle σ-Terme $t_1, \ldots, t_k$ in T_σ gilt:

$$R(t_1, \ldots, t_k) \quad \in \quad \mathrm{FO}[\sigma].$$

$\mathrm{FO}[\sigma]$-Formeln der Form $t_1 = t_2$ oder $R(t_1, \ldots, t_k)$ heißen atomare σ-Formeln.

Rekursive Regeln:

- Ist $\varphi \in \mathsf{FO}[\sigma]$, so ist auch $\neg\varphi \in \mathsf{FO}[\sigma]$.
- Ist $\varphi \in \mathsf{FO}[\sigma]$ und $\psi \in \mathsf{FO}[\sigma]$, so ist auch
  - $(\varphi \wedge \psi) \in \mathsf{FO}[\sigma]$,
  - $(\varphi \vee \psi) \in \mathsf{FO}[\sigma]$,
  - $(\varphi \to \psi) \in \mathsf{FO}[\sigma]$.
- Ist $\varphi \in \mathsf{FO}[\sigma]$ und $x \in \mathsf{VAR}$, so ist auch
  - $\exists x\, \varphi \in \mathsf{FO}[\sigma]$,
  - $\forall x\, \varphi \in \mathsf{FO}[\sigma]$.

Beispiel 3.16

Sei $\sigma = \{ f/2,\ c \}$.

Folgende Worte aus $A^*_{\mathrm{FO}[\sigma]}$ sind FO[σ]-Formeln:

- $f(v_0, v_1) = c$ (atomare σ-Formel)
- $\forall v_2\, f(v_2, c) = v_2$
- $\neg\exists v_3 \, \big(f(v_3, v_3) = v_3 \ \wedge \ \neg\, v_3{=}c\big)$

Folgende Worte sind <u>keine</u> FO[σ]-Formeln:

- $\big(f(v_0, v_1) = c\big)$
- $\big(\exists v_2\, f(v_2, c) = v_2\big)$
- $f(f(c, c), v_1)$ (ist ein σ-Term, aber keine FO[σ]-Formel)
- $\exists c\, f(v_0, c) = v_0$

Beispiel 3.17

Sei $\sigma = \{E/2\}$.
Folgendes ist eine FO[σ]-Formel:

$$\forall v_0 \, \forall v_1 \Big(\big(E(v_0, v_1) \ \wedge \ E(v_1, v_0) \big) \ \rightarrow \ v_0 = v_1 \Big)$$

Intuition zur Semantik:

In einem gerichteten Graphen $\mathcal{A} = (A, E^{\mathcal{A}})$ sagt diese Formel Folgendes aus:

„Für alle Knoten $a_0 \in A$ und
für alle Knoten $a_1 \in A$ gilt:
falls $(a_0, a_1) \in E^{\mathcal{A}}$ und $(a_1, a_0) \in E^{\mathcal{A}}$, so ist $a_0 = a_1$.“

Die Formel sagt in einem Digraph $\mathcal{A} = (A, E^{\mathcal{A}})$ also aus, dass die Kantenrelation $E^{\mathcal{A}}$ antisymmetrisch ist.

Notation

- Statt mit $v_0, v_1, v_2, \dots$ bezeichnen wir Variablen oft auch mit $x, y, z, \dots$ oder mit Varianten wie $x', y_1, y_2, \dots$.

- Ähnlich wie bei der Aussagenlogik schreiben wir $(\varphi \leftrightarrow \psi)$ als Abkürzung für die Formel $\big((\varphi \to \psi) \land (\psi \to \varphi)\big)$.

- Die Menge aller Formeln der Logik der ersten Stufe ist

$$\mathsf{FO} \;:=\; \bigcup_{\sigma \text{ Signatur}} \mathsf{FO}[\sigma].$$

Abschnitt 3.4:

Semantik der Logik erster Stufe

Bevor wir die Semantik der Logik erster Stufe formal definieren, betrachten wir zunächst einige Beispiele, um ein intuitives Verständnis der Semantik der Logik erster Stufe zu erlangen.

Beispiele zur Semantik der Logik erster Stufe

Gerichtete Graphen

Beispiel 3.18

Sei $\sigma = \{E/2\}$.

(a) Die FO[σ]-Formel

$$\varphi \quad := \quad \forall x \forall y \big(E(x,y) \to E(y,x) \big)$$

besagt:

„Für alle Knoten x und für alle Knoten y gilt: Falls es eine Kante von x nach y gibt, so gibt es auch eine Kante von y nach x."

Für jeden Digraphen $\mathcal{A} = (A, E^{\mathcal{A}})$ gilt daher:

$$\mathcal{A} \text{ erfüllt } \varphi \quad \Longleftrightarrow \quad E^{\mathcal{A}} \text{ ist symmetrisch.}$$

Umgangssprachlich sagen wir auch: „Die Formel φ sagt in einem Digraphen $\mathcal{A}$ aus, dass dessen Kantenrelation symmetrisch ist."

(b) Die folgende FO[σ]-Formel drückt aus, dass es von Knoten x zu Knoten y einen Weg der Länge 3 gibt:

$$\varphi(x,y) \quad := \quad \exists z_1\, \exists z_2\, \Big(\big(E(x,z_1) \,\wedge\, E(z_1,z_2) \big) \,\wedge\, E(z_2,y) \Big).$$

(c) Die FO[σ]-Formel

$$\forall x\, \forall y\, \exists z_1\, \exists z_2\, \Big(\big(E(x,z_1) \,\wedge\, E(z_1,z_2) \big) \,\wedge\, E(z_2,y) \Big)$$

sagt in einem Digraph $\mathcal{A}$ aus, dass es zwischen je 2 Knoten einen Weg der Länge 3 gibt.

Verwandtschaftsbeziehungen

Um Verwandtschaftsbeziehungen zu modellieren, können wir eine Signatur σ nutzen, die aus den folgenden Symbolen besteht:

- 1-stellige Funktionssymbole *Vater*, *Mutter*
  (Bedeutung: $x = \mathit{Mutter}(y)$ besagt: „x ist die Mutter von y“.)
- 2-stellige Relationssymbole *Geschwister*, *Vorfahr*
  (Bedeutung: $\mathit{Geschwister}(x, y)$ besagt, dass x und y Geschwister sind; $\mathit{Vorfahr}(x, y)$ besagt, dass x ein Vorfahr von y ist.)

Generelles Wissen über Verwandtschaftsbeziehungen lässt sich durch Formeln der Logik erster Stufe repräsentieren, z.B.:

- „Personen mit gleichem Vater und gleicher Mutter sind Geschwister“:

$$\forall x \forall y \Big(\big(\big(\mathit{Vater}(x) = \mathit{Vater}(y) \ \wedge \ \mathit{Mutter}(x) = \mathit{Mutter}(y) \big) \ \wedge \ \neg\, x = y \big) \\ \rightarrow \ \mathit{Geschwister}(x, y) \Big)$$

- „Eltern sind gerade die unmittelbaren Vorfahren“:

$$\forall x \forall y \Big(\big(x{=}\mathit{Vater}(y) \ \lor \ x{=}\mathit{Mutter}(y) \big) \\ \leftrightarrow \ \big(\mathit{Vorfahr}(x,y) \ \land \ \neg \exists z \, \big(\mathit{Vorfahr}(x,z) \land \mathit{Vorfahr}(z,y) \big) \big) \Big)$$

- „Die Relation *Vorfahr* ist transitiv“:

$$\forall x \forall y \forall z \Big(\big(\mathit{Vorfahr}(x,y) \land \mathit{Vorfahr}(y,z) \big) \ \to \ \mathit{Vorfahr}(x,z) \Big)$$

- Die folgende Formel $\varphi(x,y)$ besagt „x ist Tante oder Onkel von y“:

$$\varphi(x,y) \ := \quad \exists z \Big(\mathit{Geschwister}(x,z) \ \land \ \big(z{=}\mathit{Mutter}(y) \ \lor \ z{=}\mathit{Vater}(y) \big) \Big)$$

- Die folgende Formel $\psi(x)$ besagt „x ist Vater von genau 2 Kindern“:

$$\psi(x) \;:=\; \exists y_1 \exists y_2 \Big(\big(\big(x{=}Vater(y_1) \land x{=}Vater(y_2) \big) \land \neg\, y_1{=}y_2 \big) \\ \land\ \forall z\, \big(x{=}Vater(z) \ \rightarrow\ \big(z{=}y_1 \lor z{=}y_2 \big) \big) \Big)$$

Formale Definition der Semantik der Logik erster Stufe

Um die formale Definition der Semantik der Logik erster Stufe angeben zu können, benötigen wir noch folgende Begriffe:

Notation

- Ist β eine Belegung in einer σ-Struktur $\mathcal{A}$, ist $x \in \mathrm{VAR}$ und ist $a \in A$, so sei
$$\beta\tfrac{a}{x}$$
die Belegung mit $\beta\frac{a}{x}(x) = a$ und $\beta\frac{a}{x}(y) = \beta(y)$ für alle $y \in \mathrm{VAR} \setminus \{x\}$.

- Ist $\mathcal{I} = (\mathcal{A}, \beta)$ eine σ-Interpretation, ist $x \in \mathrm{VAR}$ und ist $a \in A$, so sei
$$\mathcal{I}\tfrac{a}{x} \;:=\; (\mathcal{A}, \beta\tfrac{a}{x}).$$

Semantik der Logik erster Stufe

Definition 3.19

Sei σ eine Signatur. Rekursiv über den Aufbau von $\mathrm{FO}[\sigma]$ definieren wir eine Funktion $[\![\cdot]\!]^{\cdot}$, die jeder $\mathrm{FO}[\sigma]$-Formel φ und jeder σ-Interpretation $\mathcal{I} = (\mathcal{A}, \beta)$ einen Wahrheitswert (kurz: Wert) $[\![\varphi]\!]^{\mathcal{I}} \in \{0, 1\}$ zuordnet:

Rekursionsanfang:

- Für alle σ-Terme t_1 und t_2 in T_σ gilt:

$$[\![t_1 = t_2]\!]^{\mathcal{I}} := \begin{cases} 1, & \text{falls } [\![t_1]\!]^{\mathcal{I}} = [\![t_2]\!]^{\mathcal{I}} \\ 0, & \text{sonst.} \end{cases}$$

- Für jedes Relationssymbol $R \in \sigma$, für $k := \mathrm{ar}(R)$ und für alle σ-Terme $t_1, \ldots, t_k \in \mathrm{T}_\sigma$ gilt:

$$[\![R(t_1, \ldots, t_k)]\!]^{\mathcal{I}} := \begin{cases} 1, & \text{falls } \left([\![t_1]\!]^{\mathcal{I}}, \ldots, [\![t_k]\!]^{\mathcal{I}}\right) \in R^{\mathcal{A}} \\ 0, & \text{sonst.} \end{cases}$$

Rekursionsschritt:

- Ist $\varphi \in \mathrm{FO}[\sigma]$ und ist $x \in \mathrm{VAR}$, so ist

$$[\![\exists x\, \varphi]\!]^{\mathcal{I}} \quad := \quad \begin{cases} 1, & \text{falls es (mind.) ein } a \in A \text{ gibt, so dass } [\![\varphi]\!]^{\mathcal{I}\frac{a}{x}} = 1 \\ 0, & \text{sonst} \end{cases}$$

$$[\![\forall x\, \varphi]\!]^{\mathcal{I}} \quad := \quad \begin{cases} 1, & \text{falls für } \textit{jedes } a \in A \text{ gilt: } [\![\varphi]\!]^{\mathcal{I}\frac{a}{x}} = 1 \\ 0, & \text{sonst} \end{cases}$$

- Die Semantik der Junktoren $\neg, \wedge, \vee, \rightarrow$ ist wie in der Aussagenlogik definiert, d.h. für alle $\varphi \in \mathrm{FO}[\sigma]$ und $\psi \in \mathrm{FO}[\sigma]$ gilt:

$$\llbracket \neg\varphi \rrbracket^{\mathcal{I}} := \begin{cases} 1, & \text{falls } \llbracket \varphi \rrbracket^{\mathcal{I}} = 0 \\ 0, & \text{sonst} \end{cases}$$

$$\llbracket (\varphi \wedge \psi) \rrbracket^{\mathcal{I}} := \begin{cases} 1, & \text{falls } \llbracket \varphi \rrbracket^{\mathcal{I}} = 1 \text{ und } \llbracket \psi \rrbracket^{\mathcal{I}} = 1 \\ 0, & \text{sonst} \end{cases}$$

$$\llbracket (\varphi \vee \psi) \rrbracket^{\mathcal{I}} := \begin{cases} 0, & \text{falls } \llbracket \varphi \rrbracket^{\mathcal{I}} = 0 \text{ und } \llbracket \psi \rrbracket^{\mathcal{I}} = 0 \\ 1, & \text{sonst} \end{cases}$$

$$\llbracket (\varphi \rightarrow \psi) \rrbracket^{\mathcal{I}} := \begin{cases} 0, & \text{falls } \llbracket \varphi \rrbracket^{\mathcal{I}} = 1 \text{ und } \llbracket \psi \rrbracket^{\mathcal{I}} = 0 \\ 1, & \text{sonst} \end{cases}$$

Beispiel 3.20

Sei $\sigma = \{E/2\}$. Betrachte die FO[σ]-Formel

$$\varphi \ := \ \forall x \forall y \ \big(\ E(x,y) \rightarrow E(y,x) \ \big)$$

Für jede σ-Interpretation $\mathcal{I} = (\mathcal{A}, \beta)$ gilt:

$$\begin{aligned}
[\![\varphi]\!]^{\mathcal{I}} = 1 \quad &\iff \quad \text{für alle } a \in A \text{ gilt: } [\![\forall y(E(x,y) \rightarrow E(y,x))]\!]^{\mathcal{I}\frac{a}{x}} = 1 \\
&\iff \quad \text{für alle } a \in A \text{ gilt: für alle } b \in A \text{ gilt:} \\
&\qquad\qquad [\![(E(x,y) \rightarrow E(y,x))]\!]^{\mathcal{I}\frac{a}{x}\frac{b}{y}} = 1 \\
&\iff \quad \text{für alle } a \in A \text{ und alle } b \in A \text{ gilt:} \\
&\qquad\qquad \text{Falls } [\![E(x,y)]\!]^{\mathcal{I}\frac{a}{x}\frac{b}{y}} = 1, \text{ so } [\![E(y,x)]\!]^{\mathcal{I}\frac{a}{x}\frac{b}{y}} = 1 \\
&\iff \quad \text{für alle } a \in A \text{ und alle } b \in B \text{ gilt:} \\
&\qquad\qquad \text{Falls } (a,b) \in E^{\mathcal{A}}, \text{ so } (b,a) \in E^{\mathcal{A}} \\
&\iff \quad E^{\mathcal{A}} \text{ ist symmetrisch}
\end{aligned}$$

Die Modellbeziehung

Definition 3.21

Sei σ eine Signatur.

(a) Eine σ-Interpretation $\mathcal{I}$ *erfüllt* eine Formel $\varphi \in \mathrm{FO}[\sigma]$ (wir schreiben: $\mathcal{I} \models \varphi$), wenn $[\![\varphi]\!]^{\mathcal{I}} = 1$.

(b) Eine σ-Interpretation $\mathcal{I}$ *erfüllt* eine Formelmenge $\Phi \subseteq \mathrm{FO}[\sigma]$ (wir schreiben: $\mathcal{I} \models \Phi$), wenn $\mathcal{I} \models \varphi$ für alle $\varphi \in \Phi$ gilt.

(c) Ein *Modell* einer Formel φ (bzw. einer Formelmenge Φ) ist eine Interpretation $\mathcal{I}$ mit $\mathcal{I} \models \varphi$ (bzw. $\mathcal{I} \models \Phi$).

Konventionen

- Terme bezeichnen wir mit t, s und Varianten $s', t_1, t_2, \ldots$.
- Formeln bezeichnen wir mit φ, ψ, χ und Varianten $\psi', \varphi_1, \varphi_2, \ldots$.
- Formelmengen bezeichnen wir mit Φ, Ψ und Varianten $\Psi', \Phi_1, \Phi_2, \ldots$.

Subformeln, Subterme und Syntaxbäume

- Eine Formel ψ ist Subformel einer Formel φ, wenn ψ als Teilwort in φ vorkommt (insbes. ist jede Formel eine Subformel von sich selbst).

  Beispiel: $\psi :=\ E(v_0, v_1)$ ist Subformel der Formel $\exists v_0 \forall v_1\, E(v_0, v_1)$

- Ein Term s ist Subterm eines Terms t, wenn s als Teilwort in t vorkommt (insbes. ist jeder Term ein Subterm von sich selbst).

  Beispiel: $f(c, c)$ ist Subterm des Terms $f(v_0, f(c, c))$.

- Sei $\xi \in \mathrm{T} \cup \mathrm{FO}$, d.h. ξ ist ein Term oder eine Formel der Logik erster Stufe.
  - Ähnlich wie bei aussagenlogischen Formeln können wir einen Syntaxbaum für ξ definieren.
  - Das Lemma über die eindeutige Lesbarkeit von Termen und Formeln besagt, dass jeder Term und jede Formel genau einen Syntaxbaum hat.
  - Die Subterme von ξ (falls $\xi \in \mathrm{T}$) bzw. Subformeln von ξ (falls $\xi \in \mathrm{FO}$) sind dann alle Terme bzw. Formeln, die im Syntaxbaum vorkommen.

Das Isomorphielemma

Das Isomorphielemma besagt, dass isomorphe Objekte (Strukturen bzw. Interpretationen) dieselben Formeln der Logik erster Stufe erfüllen.

Um diese Aussage präzise formulieren zu können, benötigen wir die folgende Notation.

Isomorphismen, Belegungen und Interpretationen

Definition 3.22

Sei σ eine Signatur, seien $\mathcal{A}, \mathcal{B}$ isomorphe σ-Strukturen und sei π ein Isomorphismus von $\mathcal{A}$ nach $\mathcal{B}$ (kurz: $\pi : \mathcal{A} \cong \mathcal{B}$).

(a) Für jede Belegung β in $\mathcal{A}$ sei $\pi\beta$ die Belegung in $\mathcal{B}$, so dass für alle $x \in \mathrm{VAR}$ gilt:

$$\pi\beta(x) \quad = \quad \pi\big(\beta(x)\big).$$

(b) Für eine Interpretation $\mathcal{I} = (\mathcal{A}, \beta)$ schreiben wir $\pi\mathcal{I}$ für die Interpretation

$$\pi\mathcal{I} \quad := \quad (\mathcal{B}, \pi\beta).$$

Aus dieser Definition folgt direkt:

Lemma 3.23

Sei σ eine Signatur, seien $\mathcal{A}, \mathcal{B}$ isomorphe σ-Strukturen, sei $\pi : \mathcal{A} \cong \mathcal{B}$, sei β eine Belegung in $\mathcal{A}$ und sei $\mathcal{I} := (\mathcal{A}, \beta)$.
Für jedes $x \in \mathrm{VAR}$, für jedes $a \in A$, für $\mathcal{I}' := \mathcal{I}\frac{a}{x}$ und für $b := \pi(a)$ gilt:

$$\pi\mathcal{I}' \quad = \quad (\pi\mathcal{I})\tfrac{b}{x}.$$

Das Isomorphielemma

Satz 3.24 (Das Isomorphielemma der Logik erster Stufe)

Sei σ eine Signatur, seien $\mathcal{A}, \mathcal{B}$ isomorphe σ-Strukturen und sei $\pi : \mathcal{A} \cong \mathcal{B}$. Für jede Belegung β in $\mathcal{A}$ und die σ-Interpretation $\mathcal{I} := (\mathcal{A}, \beta)$ gilt:

(a) *Für jeden σ-Term $t \in \mathrm{T}_\sigma$ ist* $[\![t]\!]^{\pi\mathcal{I}} \; = \; \pi\big([\![t]\!]^{\mathcal{I}}\big)$.

(b) *Für jede* FO[σ]*-Formel φ gilt:* $\pi\mathcal{I} \models \varphi \iff \mathcal{I} \models \varphi$.

Wir werden das Isomorphielemma per Induktion über den Aufbau von Termen und Formeln beweisen. Hierzu zunächst ein kurzer Überblick darüber, wie solche Induktionsbeweise prinzipiell aufgebaut sind.

Beweise per Induktion über den Aufbau von Termen und Formeln

- Ähnlich wie Aussagen über die aussagenlogischen Formeln können wir Aussagen über Terme und Formeln der Logik der erster Stufe per Induktion über den Aufbau von T_σ bzw. $FO[\sigma]$ beweisen.

- Im Induktionsanfang beweisen wir die Aussagen für die gemäß Basisregeln definierten Terme bzw. Formeln. Im Induktionschritt schließen wir von den Subtermen bzw. Subformeln auf den Term bzw. die Formel selbst.

- Wie bei der Aussagenlogik ist dieses Vorgehen gerechtfertigt, weil es sich auch als vollständige Induktion über die Höhe des Syntaxbaums auffassen lässt.

Beweise per Induktion über den Aufbau von Termen

Schematisch sieht der Beweis einer Aussage $\mathbb{A}(t)$ für alle Terme $t \in \mathrm{T}_\sigma$ wie folgt aus:

Induktionsanfang:

- Beweise, dass für alle Konstantensymbole $c \in \sigma$ die Aussage $\mathbb{A}(c)$ gilt.
- Beweise, dass für alle Variablen $x \in \mathrm{VAR}$ die Aussage $\mathbb{A}(x)$ gilt.

Induktionsschritt:

- Betrachte jedes Funktionssymbol $f \in \sigma$, sei $k := \mathrm{ar}(f)$, und seien $t_1, \ldots, t_k$ beliebige σ-Terme. Beweise, dass $\mathbb{A}\big(f(t_1, \ldots, t_k)\big)$ gilt, und verwende dazu die Induktionsannahme, dass $\mathbb{A}(t_i)$ für jedes $i \in [k]$ gilt.

Mit dieser Vorgehensweise beweisen wir nun Teil (a) des Isomorphielemmas.

Teil (b) des Isomorphielemmas beweisen wir per Induktion über den Aufbau von Formeln. Prinzipiell sind solche Induktionsbeweise wie folgt aufgebaut.

Beweise per Induktion über den Aufbau von Formeln

Schematisch sieht der Beweis einer Aussage $\mathbb{A}(\varphi)$ für alle FO[σ]-Formeln φ wie folgt aus:

Induktionsanfang:

- Beweise, dass für alle σ-Terme $t_1, t_2 \in \mathrm{T}_\sigma$ die Aussage $\mathbb{A}(t_1{=}t_2)$ gilt.
- Beweise, dass für alle Relationssymbole $R \in \sigma$, für $k := \mathrm{ar}(R)$ und für alle σ-Terme $t_1, \ldots, t_k \in \mathrm{T}_\sigma$ die Aussage $\mathbb{A}\big(R(t_1, \ldots, t_k)\big)$ gilt

Induktionsschritt:

Seien φ und ψ beliebige FO[σ]-Formeln. Die Induktionsannahme besagt, dass die Aussagen $\mathbb{A}(\varphi)$ und $\mathbb{A}(\psi)$ gelten.

Im Induktionsschritt muss dann gezeigt werden, dass

- für jede Variable $x \in \text{VAR}$ die Aussage $\mathbb{A}(\exists x\, \varphi)$ gilt,
- für jede Variable $x \in \text{VAR}$ die Aussage $\mathbb{A}(\forall x\, \varphi)$ gilt,
- die Aussage $\mathbb{A}(\neg\varphi)$ gilt,
- die Aussage $\mathbb{A}\big((\varphi \wedge \psi)\big)$ gilt,
- die Aussage $\mathbb{A}\big((\varphi \vee \psi)\big)$ gilt,
- die Aussage $\mathbb{A}\big((\varphi \to \psi)\big)$ gilt.

Mit dieser Vorgehensweise beweisen wir nun Teil (b) des Isomorphielemmas.

Das Koinzidenzlemma

Ähnlich wie für die Aussagenlogik gilt auch für die Logik erster Stufe ein Koinzidenzlemma, das besagt, dass der Wert $[\![t]\!]^{\mathcal{I}}$ eines Terms t bzw. der Wert $[\![\varphi]\!]^{\mathcal{I}}$ einer Formel φ nur abhängt von

- denjenigen Bestandteilen von $\mathcal{A}$, die explizit in t bzw. φ vorkommen, und
- den Belegungen $\beta(x)$ derjenigen Variablen x, die in t vorkommen bzw. die in φ vorkommen und nicht im Wirkungsbereich eines Quantors stehen.

Um diese Aussage präzise zu formulieren, sind folgende Begriffe nützlich.

Definition 3.25

(a) Ist ξ ein Term oder eine Formel der Logik erster Stufe, so schreiben wir

- $\sigma(\xi)$, um die Menge aller Relations-, Funktions- und Konstantensymbole zu bezeichnen, die in ξ vorkommen,
- $\mathrm{var}(\xi)$, um die Menge aller in ξ vorkommenden Variablen zu bezeichnen.

(b) Ist φ eine Formel und x eine Variable, so heißt jedes Vorkommen von x in einer Subformel von φ, die von der Form $\exists x\psi$ oder $\forall x\psi$ ist, gebunden. Jedes andere Vorkommen von x in φ heißt frei.

Beispiel:

$$\varphi \;:=\; \big(\, f(v_0, c){=}v_3 \;\wedge\; \exists v_0\, f(v_0, v_1){=}c \,\big)$$

Das erste Vorkommen von v_0 in φ ist frei, das zweite und dritte Vorkommen von v_0 in φ ist gebunden. Die Vorkommen von v_1 und v_3 in φ sind frei.

Freie Variablen

Definition 3.26

Die Menge frei(φ) aller freien Variablen einer Formel φ besteht aus allen Variablen, die mindestens ein freies Vorkommen in φ haben.
Die Menge frei(φ) lässt sich rekursiv über den Aufbau von Formeln wie folgt definieren:

$$\begin{aligned}
\text{frei}(R(t_1,\dots,t_k)) &:= \text{var}(t_1)\cup\cdots\cup\text{var}(t_k)\\
\text{frei}(t_1{=}t_2) &:= \text{var}(t_1)\cup\text{var}(t_2)\\
\text{frei}(\neg\varphi) &:= \text{frei}(\varphi)\\
\text{frei}((\varphi*\psi)) &:= \text{frei}(\varphi)\cup\text{frei}(\psi) \quad \text{für alle } *\in\{\wedge,\vee,\rightarrow\}\\
\text{frei}(\exists x\,\varphi) &:= \text{frei}(\forall x\,\varphi) \quad := \quad \text{frei}(\varphi)\setminus\{x\}.
\end{aligned}$$

Beispiele:

- $\text{frei}(f(v_0,c){=}v_3) \;=\; \{v_0,v_3\}$
- $\text{frei}(\exists v_0\, f(v_0,v_1){=}c) \;=\; \{v_1\}$
- $\text{frei}(\,(\,f(v_0,c){=}v_3 \;\wedge\; \exists v_0\, f(v_0,v_1){=}c\,)\,) \;=\; \{v_0,v_3,v_1\}$

Das Koinzidenzlemma

Satz 3.27 (Koinzidenzlemma für Terme)

Sei $\mathcal{I}_1 = (\mathcal{A}_1, \beta_1)$ eine σ_1-Interpretation und sei $\mathcal{I}_2 = (\mathcal{A}_2, \beta_2)$ eine σ_2-Interpretation, wobei σ_1 und σ_2 Signaturen seien. Sei $t \in \mathrm{T}$ ein Term mit $\sigma(t) \subseteq \sigma_1 \cap \sigma_2$, so dass gilt:

1. $\mathcal{A}_1|_{\sigma(t)} \;=\; \mathcal{A}_2|_{\sigma(t)}$

   (d.h., die $\sigma(t)$-Redukte von $\mathcal{A}_1$ und $\mathcal{A}_2$ sind identisch), und
2. $\beta_1(x) = \beta_2(x)$, *für alle $x \in \mathrm{var}(t)$.*

Dann gilt: $[\![t]\!]^{\mathcal{I}_1} \;=\; [\![t]\!]^{\mathcal{I}_2}$.

Beweis: Per Induktion über den Aufbau von Termen. Details: Übung. $\square$

Satz 3.28 (Koinzidenzlemma für FO-Formeln)

Sei $\mathcal{I}_1 = (\mathcal{A}_1, \beta_1)$ eine σ_1-Interpretation und sei $\mathcal{I}_2 = (\mathcal{A}_2, \beta_2)$ eine σ_2-Interpretation, wobei σ_1 und σ_2 Signaturen seien.
Sei $\varphi \in \mathrm{FO}$ eine Formel der Logik erster Stufe mit $\sigma(\varphi) \subseteq \sigma_1 \cap \sigma_2$, so dass gilt:

1. $\mathcal{A}_1|_{\sigma(\varphi)} \;=\; \mathcal{A}_2|_{\sigma(\varphi)}$, *und*
2. $\beta_1(x) = \beta_2(x)$, *für alle $x \in \mathrm{frei}(\varphi)$.*

Dann gilt: $\mathcal{I}_1 \models \varphi \iff \mathcal{I}_2 \models \varphi$.

Beweis: Per Induktion über den Aufbau von Formeln. Details: Übung. $\square$

Notation für Terme

- Für einen Term $t \in \mathrm{T}_\sigma$ schreiben wir $t(x_1, \ldots, x_n)$, um anzudeuten, dass $\mathrm{var}(t) \subseteq \{x_1, \ldots, x_n\}$.

- Sei $\mathcal{A}$ eine σ-Struktur und seien $a_1, \ldots, a_n \in A$.
  Auf Grund des Koinzidenzlemmas gilt

$$[\![t]\!]^{(\mathcal{A},\beta)} \;=\; [\![t]\!]^{(\mathcal{A},\beta')}$$

  für alle Belegungen $\beta, \beta' : \mathrm{VAR} \to A$, so dass $\beta(x_i) = a_i = \beta'(x_i)$ für alle $i \in [n]$ gilt. Wir schreiben oft

$$t^{\mathcal{A}}[a_1, \ldots, a_n],$$

  um das Element $[\![t]\!]^{(\mathcal{A},\beta)}$ zu bezeichnen.

- Für Terme $t \in \mathrm{T}_\sigma$, in denen keine Variable vorkommt, d.h. $\mathrm{var}(t) = \emptyset$ (so genannte Grundterme), schreiben wir einfach $t^{\mathcal{A}}$.

Notation für Formeln

- Für eine FO[σ]-Formel φ schreiben wir $\varphi(x_1, \ldots, x_n)$, um anzudeuten, dass $\text{frei}(\varphi) \subseteq \{x_1, \ldots, x_n\}$.

- Ist $\mathcal{A}$ eine σ-Struktur und sind $a_1, \ldots, a_n \in A$, so schreiben wir

$$\mathcal{A} \models \varphi[a_1, \ldots, a_n]$$

wenn $(\mathcal{A}, \beta) \models \varphi$ für eine Belegung $\beta : \text{VAR} \to A$ mit $\beta(x_i) = a_i$ für alle $i \in [n]$ gilt.

Auf Grund des Koinzidenzlemmas gilt dann auch für alle Belegungen $\beta' : \text{VAR} \to A$ mit $\beta'(x_i) = a_i$ für alle $i \in [n]$, dass $(\mathcal{A}, \beta') \models \varphi$.

Sätze der Logik erster Stufe

Definition 3.29

Sei σ eine Signatur.

(a) Ein FO[σ]-Satz (kurz: Satz) ist eine FO[σ]-Formel φ mit $\text{frei}(\varphi) = \emptyset$.

(b) Wir schreiben S_σ, um die Menge aller FO[σ]-Sätze zu bezeichnen und setzen

$$\mathsf{S} := \bigcup_{\sigma \text{ Signatur}} \mathsf{S}_\sigma.$$

(c) Für einen FO[σ]-Satz φ und eine σ-Struktur $\mathcal{A}$ schreiben wir $\mathcal{A} \models \varphi$, um auszudrücken, dass $(\mathcal{A}, \beta) \models \varphi$ für eine (und gemäß Koinzidenzlemma daher für jede) Belegung β in $\mathcal{A}$ gilt.

(d) Für eine Menge $\Phi \subseteq \mathsf{S}_\sigma$ von FO[σ]-Sätzen schreiben wir $\mathcal{A} \models \Phi$, falls $\mathcal{A} \models \varphi$ für *jedes* $\varphi \in \Phi$ gilt.

Als direkte Folgerung aus dem Isomorphielemma erhalten wir, dass für isomorphe σ-Strukturen $\mathcal{A}$ und $\mathcal{B}$ und für alle FO[σ]-Sätze φ gilt:

$$\mathcal{A} \models \varphi \iff \mathcal{B} \models \varphi.$$

Modellklassen und Definierbarkeit

Definition 3.30

Sei σ eine Signatur und sei $\Phi \subseteq \mathsf{S}_\sigma$ (d.h. Φ ist eine Menge von FO[σ]-Sätzen).

(a) Die Modellklasse von Φ ist die Klasse $\mathrm{MOD}_\sigma(\Phi)$ aller σ-Strukturen $\mathcal{A}$ für die gilt: $\mathcal{A} \models \Phi$.

(b) Für eine Klasse $\mathfrak{C}$ von σ-Strukturen sagen wir

Φ definiert (oder axiomatisiert) $\mathfrak{C}$,

falls $\mathfrak{C} = \mathrm{MOD}_\sigma(\Phi)$.

(c) Für einen FO[σ]-Satz φ setzen wir $\mathrm{MOD}_\sigma(\varphi) := \mathrm{MOD}_\sigma(\{\varphi\})$ und sagen, dass φ die Klasse $\mathfrak{C} := \mathrm{MOD}_\sigma(\varphi)$ definiert (bzw. axiomatisiert).

Als direkte Folgerung aus dem Isomorphielemma erhalten wir:

Korollar 3.31

Für jede Signatur σ und jedes $\Phi \subseteq \mathsf{S}_\sigma$ ist $\mathrm{MOD}_\sigma(\Phi)$ unter Isomorphie abgeschlossen. D.h. für isomorphe σ-Strukturen $\mathcal{A}$ und $\mathcal{B}$ gilt:

$$\mathcal{A} \in \mathrm{MOD}_\sigma(\Phi) \quad \Longleftrightarrow \quad \mathcal{B} \in \mathrm{MOD}_\sigma(\Phi).$$

Abschnitt 3.5:

Beispiele für Formeln der Logik erster Stufe in verschiedenen Anwendungsbereichen

Notation

- Ab jetzt verwenden wir für die Logik erster Stufe ähnliche Klammerkonventionen wie bei der Aussagenlogik.

- Für gewisse zweistellige Funktionssymbole wie $+, \cdot$ und zweistellige Relationssymbole wie $\leqslant$ verwenden wir Infix- statt Präfixnotation. Dabei setzen wir auf natürliche Weise Klammern, um die eindeutige Lesbarkeit zu gewährleisten.

- Wir schreiben $x < y$ als Abkürzung für die Formel $\big(x \leqslant y \ \land \ \neg x{=}y \big)$.

Ordnungen

Beispiel 3.32

Wir betrachten Strukturen und Formeln über der Signatur $\sigma := \{\leqslant\}$.

Zur Erinnerung: Eine σ-Struktur $\mathcal{A} = (A, \leqslant^{\mathcal{A}})$ ist eine lineare Ordnung, falls gilt:

(1) $\leqslant^{\mathcal{A}}$ ist reflexiv,

- d.h. für alle $a \in A$ gilt: $a \leqslant^{\mathcal{A}} a$
- d.h. $\mathcal{A} \models \varphi_{refl}$, wobei

$$\varphi_{refl} \ := \quad \forall x \ \ x \leqslant x$$

(2) $\leqslant^{\mathcal{A}}$ ist transitiv,

- d.h. für alle $a, b, c \in A$ gilt: Wenn $a \leqslant^{\mathcal{A}} b$ und $b \leqslant^{\mathcal{A}} c$, dann auch $a \leqslant^{\mathcal{A}} c$
- d.h. $\mathcal{A} \models \varphi_{trans}$, wobei

$$\varphi_{trans} \ := \quad \forall x \forall y \forall z \ \Big(\big(x \leqslant y \ \wedge \ y \leqslant z \big) \ \rightarrow \ x \leqslant z \Big)$$

(3) $\leqslant^{\mathcal{A}}$ ist antisymmetrisch,

- d.h. für alle $a, b \in A$ mit $a \neq b$ gilt: Wenn $a \leqslant^{\mathcal{A}} b$, dann $b \not\leqslant^{\mathcal{A}} a$
- d.h. $\mathcal{A} \models \varphi_{antisym}$, wobei

$$\varphi_{antisym} \quad := \quad \forall x \forall y \Big(\neg\, x = y \ \rightarrow \big(x \leqslant y \ \rightarrow \ \neg\, y \leqslant x \big) \Big)$$

(4) $\leqslant^{\mathcal{A}}$ ist konnex,

- d.h. für alle $a, b \in A$ gilt: $a \leqslant^{\mathcal{A}} b$ oder $b \leqslant^{\mathcal{A}} a$ oder $a = b$
- d.h. $\mathcal{A} \models \varphi_{konnex}$, wobei

$$\varphi_{konnex} \quad := \quad \forall x \forall y \Big(x \leqslant y \ \vee \ y \leqslant x \ \vee \ x = y \Big)$$

Insgesamt gilt für jede $\{\leqslant\}$-Struktur $\mathcal{A} = (A, \leqslant^{\mathcal{A}})$:
$\mathcal{A} = (A, \leqslant^{\mathcal{A}})$ ist eine lineare Ordnung $\iff$ $\mathcal{A} \models \varphi_{lin.Ord}$, wobei

$$\varphi_{lin.Ord} \quad := \quad \varphi_{refl} \ \wedge \ \varphi_{antisym} \ \wedge \ \varphi_{trans} \ \wedge \ \varphi_{konnex}$$

Der FO$[\sigma]$-Satz $\varphi_{lin.Ord}$ definiert (bzw. axiomatisiert) also die Klasse aller linearen Ordnungen.

Arithmetik

Beispiel 3.33

Wir betrachten Formeln über der Signatur $\sigma := \{+, \cdot, \leqslant, \underline{0}, \underline{1}\}$ und ihre Bedeutung im Standardmodell $\mathcal{A}_{\mathbb{N}}$ der Arithmetik.

- Gesucht: Eine FO[σ]-Formel $\varphi_-(x, y, z)$, die besagt „$x - y = z$“.
  Präzise: Für alle $a, b, c \in \mathbb{N}$ soll gelten:
  $$\mathcal{A}_{\mathbb{N}} \models \varphi_-[a, b, c] \iff a - b = c.$$
  Lösung:
  $$\varphi_-(x, y, z) := \quad x = z + y$$

- Gesucht: Eine FO[σ]-Formel $\varphi_|(x, y)$, die besagt „x teilt y“.
  Präzise: Für alle $a, b \in \mathbb{N}$ soll gelten:
  $$\mathcal{A}_{\mathbb{N}} \models \varphi_|[a, b] \iff \text{es gibt ein } c \in \mathbb{N}, \text{ so dass } a \cdot c = b.$$
  Lösung:
  $$\varphi_|(x, y) := \quad \exists z \ \ x \cdot z = y$$

- Gesucht: Eine FO[σ]-Formel $\varphi_{\equiv}(x,y,z)$, die besagt „$x \equiv y \pmod{z}$“.

  Präzise: Für alle $a,b,c \in \mathbb{N}$ soll gelten:

$$\mathcal{A}_{\mathbb{N}} \models \varphi_{\equiv}[a,b,c] \quad \Longleftrightarrow \quad a \equiv b \pmod{c} \quad \text{d.h.} \quad c \;\Big|\; |a-b|$$

Lösung:

$$\varphi_{\equiv}(x,y,z) \;:=\; \exists w \Big(\underbrace{\big(\varphi_{-}(x,y,w) \vee \varphi_{-}(y,x,w)\big)}_{\text{„}w = |x-y|\text{“}} \wedge \underbrace{\varphi_{|}(z,w)}_{\text{„}z \mid w\text{“}} \Big)$$

- Gesucht: Eine FO[σ]-Formel $\varphi_{prim}(x)$, die besagt „x ist eine Primzahl".

  Präzise: Für alle $a \in \mathbb{N}$ soll gelten:

  $\mathcal{A}_\mathbb{N} \models \varphi_{prim}[a] \iff$ a ist eine Primzahl

  d.h. $a \geqslant 2$ und a ist nur durch sich selbst und durch 1 teilbar.

  Lösung:

  $$\varphi_{prim}(x) \ := \ \underbrace{\underline{1}+\underline{1} \leqslant x}_{„x \,\geqslant\, 2"} \ \land \ \forall z \left(\underbrace{\varphi_{|}(z,x)}_{„z \,|\, x"} \ \to \ \big(z = x \ \lor \ z = \underline{1} \big) \right)$$

- Gesucht: Ein FO[σ]-Satz φ_∞, der in $\mathcal{A}_\mathbb{N}$ besagt

  „Es gibt unendlich viele Primzahlen".

  Lösung:

  $$\varphi_\infty \ := \ \forall y \, \exists x \left(y \leqslant x \ \land \ \varphi_{prim}(x) \right)$$

  In $\mathcal{A}_\mathbb{N}$ besagt dieser Satz, dass es für jede natürliche Zahl b eine natürliche Zahl $a \geqslant b$ gibt, die eine Primzahl ist.

Worte

Beispiel 3.34

Wir betrachten das Alphabet $\Sigma := \{a, b\}$ und die Signatur $\sigma_\Sigma = \{\leqslant, P_a, P_b\}$.

Zur Erinnerung: Wir repräsentieren ein nicht-leeres Wort $w \in \Sigma^*$ durch die σ_Σ-Struktur $\mathcal{A}_w$, deren Universum aus der Menge $\{1, \ldots, |w|\}$ aller Positionen in w besteht, und bei der $P_a^{\mathcal{A}_w}$ (bzw. $P_b^{\mathcal{A}_w}$) aus allen Positionen besteht, an denen der Buchstabe a (bzw. b) steht.

Gesucht: Ein FO$[\sigma_\Sigma]$-Satz φ, so dass für jedes nicht-leere Wort $w \in \Sigma^*$ gilt:

$$\mathcal{A}_w \models \varphi \quad\iff\quad w \text{ ist von der Form } a^* b^*.$$

Lösung: Wir konstruieren eine Formel φ, die besagt, dass es eine Position x gibt, so dass alle Positionen links von x den Buchstaben a tragen und alle Positionen rechts von x den Buchstaben b tragen.

$$\varphi \;:=\; \exists x\, \forall y\, \Big(\big(y < x \to P_a(y) \big) \;\land\; \big(x < y \to P_b(y) \big) \Big)$$

Wie bereits vereinbart, schreiben wir hier „$x < y$“ als Abkürzung für die Formel $(x \leqslant y \land \neg\, x = y)$.

Transitionssysteme

Beispiel 3.35

Sei σ_A eine Menge von Aktionen und σ_P eine Menge von Propositionen. Wir betrachten Formeln über der Signatur $\sigma := \sigma_A \cup \sigma_P$.

Zur Erinnerung: Ein (σ_A, σ_P)-Transitionssystem ist eine $(\sigma_A \cup \sigma_P)$-Struktur $\mathcal{T}$.

- Sei

$$\varphi \ := \ \ \forall x \left(\ P(x) \ \rightarrow \ \exists y \ R(x,y) \ \right)$$

  wobei $P \in \sigma_P$ und $R \in \sigma_A$ ist.

  Dann gilt für alle (σ_A, σ_P)-Transitionssysteme $\mathcal{T}$:

  $\mathcal{T} \models \varphi \iff$ In allen Zuständen von $\mathcal{T}$, in denen P gilt, ist die Aktion R möglich.

- Sei $n \geqslant 2$, seien $R_1, \ldots, R_n \in \sigma_A$ und sei $P_F \in \sigma_P$.

  Gesucht: Eine FO[σ]-Formel $\psi_n(x)$, so dass für jedes (σ_A, σ_P)-Transitionssystem $\mathcal{T}$ und jeden Zustand $t \in T$ gilt:

  $\mathcal{T} \models \psi[t] \iff$ Vom Zustand t aus lässt sich mittels der Folge $(R_1, \ldots, R_n)$ von Aktionen ein Zustand erreichen, in dem P_F gilt.

  Lösung: Für $n = 3$ können wir beispielsweise folgende Formel wählen:

  $$\psi_3(x) \;:=\; \exists y_1 \exists y_2 \exists y_3 \Big(R_1(x, y_1) \wedge R_2(y_1, y_2) \wedge R_3(y_2, y_3) \wedge P_F(y_3) \Big)$$

  Allgemein ist für $n \geqslant 2$ die Formel $\psi_n(x)$ von der folgenden Form:

  $$\exists y_1 \exists y_2 \cdots \exists y_n \Big(R_1(x, y_1) \wedge R_2(y_1, y_2) \wedge \cdots \wedge R_n(y_{n-1}, y_n) \wedge P_F(y_n) \Big)$$

Abschnitt 3.6:

Logik und Datenbanken

Datenbanken

Zur Erinnerung: Wir repräsentieren eine Kinodatenbank, die Informationen über Kinos, Filme und das aktuelle Programm enthält, durch eine Struktur über der Signatur $\sigma_{\mathsf{KINO}} :=$

$$\{\, R_{Kino}/4,\ \ R_{Film}/3,\ \ R_{Prog}/3 \,\} \ \cup\ \{\, \text{`}c\text{'} \ :\ c \in \mathtt{ASCII}^* \,\}$$

und können so z.B. die folgende Kinodatenbank als σ_{KINO}-Struktur $\mathcal{D}$ auffassen, deren Universum D aus der Menge aller Worte über dem ASCII-Alphabet besteht.

Beispiel: Eine Kinodatenbank

Kino			
Name	Adresse	Stadtteil	Telefonnummer
Babylon	Dresdner Str. 126	Kreuzberg	030 61 60 96 93
Casablanca	Friedenstr. 12-13	Adlershof	030 67 75 75 2
Filmtheater am Friedrichshain	Bötzowstr. 1-5	Prenzlauer Berg	030 42 84 51 88
Kino International	Karl-Marx-Allee 33	Mitte	030 24 75 60 11
Moviemento	Kotbusser Damm 22	Kreuzberg	030 692 47 85
Urania	An der Urania 17	Schöneberg	030 21 89 09 1

Film		
Name	Regisseur	Schauspieler
Alien	Ridley Scott	Sigourney Weaver
Blade Runner	Ridley Scott	Harrison Ford
Blade Runner	Ridley Scott	Sean Young
Brazil	Terry Gilliam	Jonathan Pryce
Brazil	Terry Gilliam	Kim Greist
Casablanca	Michael Curtiz	Humphrey Bogart
Casablanca	Michael Curtiz	Ingrid Bergmann
Gravity	Alfonso Cuaron	Sandra Bullock
Gravity	Alfonso Cuaron	George Clooney
Monuments Men	George Clooney	George Clooney
Monuments Men	George Clooney	Matt Damon
Resident Evil	Paul Anderson	Milla Jovovich
Terminator	James Cameron	Arnold Schwarzenegger
Terminator	James Cameron	Linda Hamilton
Terminator	James Cameron	Michael Biehn
...	...	...

Programm		
Kino	Film	Zeit
Babylon	Casablanca	17:30
Babylon	Gravity	20:15
Casablanca	Blade Runner	15:30
Casablanca	Alien	18:15
Casablanca	Blade Runner	20:30
Casablanca	Resident Evil	20:30
Filmtheater am Friedrichshain	Resident Evil	20:00
Filmtheater am Friedrichshain	Resident Evil	21:30
Filmtheater am Friedrichshain	Resident Evil	23:00
Kino International	Casablanca	18:00
Kino International	Brazil	20:00
Kino International	Brazil	22:00
Moviemento	Gravity	17:00
Moviemento	Gravity	19:30
Moviemento	Alien	22:00
Urania	Monuments Men	17:00
Urania	Monuments Men	20:00

Die Kinodatenbank als Struktur

Signatur: $\sigma_{\text{KINO}} \ := \ \{\ R_{Kino}/4,\ \ R_{Film}/3,\ \ R_{Prog}/3\ \}\ \cup\ \{\ \text{'}c\text{'}\ :\ c \in \texttt{ASCII}^*\ \}$

Die Kinodatenbank wird dargestellt als σ_{KINO}-Struktur $\mathcal{D}$.

Universum:

$$D \ := \ \texttt{ASCII}^* \ \supseteq \ \{\text{Babylon, Dresdner Str. 126, Kreuzberg, 030 61 60 96 93, Casablanca, \ldots, 20:00}\}.$$

Relationen:

$$
\begin{aligned}
R_{Kino}^{\mathcal{D}} := \{ \ & (\text{Babylon, Dresdner Str. 126, Kreuzberg, 030 61 60 96 93}),\\
& (\text{Casablanca, Friedenstr. 12-13, Adlershof, 030 67 75 75 2}),\\
& \ldots,\\
& (\text{Urania, An der Urania 17, Schöneberg, 030 21 89 09 1}) \ \}\\
R_{Film}^{\mathcal{D}} := \{ \ & (\text{Alien, Ridley Scott, Sigourney Weaver}),\\
& (\text{Blade Runner, Ridley Scott, Harrison Ford}), \quad \ldots \quad \}\\
R_{Prog}^{\mathcal{D}} := \{ \ & (\text{Babylon, Casablanca, 17:30}),\\
& (\text{Babylon, Gravity, 20:15}), \quad \ldots \quad \}.
\end{aligned}
$$

Konstanten: $\text{'}c\text{'}^{\mathcal{D}} \ := \ c$, für jedes $c \in \texttt{ASCII}^*$.

D.h.: jedes Konstantensymbol wird durch den zwischen den Hochkommas stehenden Text interpretiert.

Beispiel 3.36

(a) Die Anfrage

„Gib die Titel aller Filme aus, die um 22:00 Uhr beginnen."

lässt sich durch folgende FO[σ_{KINO}]-Formel $\varphi_1(x_T)$ beschreiben:

$$\varphi_1(x_T) \ := \quad \exists x_K \ R_{Prog}(x_K, x_T, \text{‘22:00’})$$

(b) Die Anfrage

„Gib die Titel aller Filme aus, in denen George Clooney mitspielt oder Regie führt"

lässt sich durch folgende FO[σ_{KINO}]-Formel beschreiben: $\varphi_2(x_T) :=$

$$\exists x_R \ R_{Film}(x_T, x_R, \text{‘George Clooney’}) \ \lor \ \exists x_S \ R_{Film}(x_T, \text{‘George Clooney’}, x_S)$$

(c) Die Anfrage

„Gib Name und Stadtteil aller Kinos aus, in denen ein Film läuft, in dem George Clooney mitspielt oder Regie führt"

lässt sich durch folgende FO[σ_{KINO}]-Formel beschreiben: $\varphi_3(x_K, x_{St}) :=$

$$\exists x_A\, \exists x_{Tel}\;\; R_{Kino}(x_K, x_A, x_{St}, x_{Tel}) \quad \wedge$$
$$\exists x_T\, \exists x_Z\; \Big(R_{Prog}(x_K, x_T, x_Z) \;\wedge$$
$$\big(\exists x_R\;\, R_{Film}(x_T, x_R, \text{‘George Clooney'}) \;\vee\; \exists x_S\;\, R_{Film}(x_T, \text{‘George Clooney'}, x_S)\big)\Big)$$

Die erste Zeile der Formel stellt sicher, dass x_K ein Kino und x_S dessen Stadtteil ist; die Zeilen 2 und 3 stellen sicher, dass im Kino x_K ein Film läuft, in dem George Clooney mitspielt oder Regie führt.

Eine andere Sichtweise auf die Semantik

- Anstatt Wahrheitswerte in Interpretationen definieren Formeln der Logik der ersten Stufe auch Relationen in Strukturen.

- Junktoren und Quantoren entsprechen dann algebraischen Operatoren auf Relationen.

- Diese Sichtweise ist insbesondere in der Datenbanktheorie wichtig und bildet die Grundlage effizienter Algorithmen zur Auswertung von Datenbankanfragen.

Definition 3.37

Sei σ eine Signatur, sei $\varphi(x_1, \dots, x_n)$ eine FO[σ]-Formel und sei $\mathcal{A}$ eine σ-Struktur.
Die von $\varphi(x_1, \dots, x_n)$ in $\mathcal{A}$ definierte n-stellige Relation ist

$$[\![\varphi(x_1, \dots, x_n)]\!]^{\mathcal{A}} \quad := \quad \{\, (a_1, \dots, a_n) \in A^n \;:\; \mathcal{A} \models \varphi[a_1, \dots, a_n] \,\}.$$

Vorsicht: Die Relation $[\![\varphi(x_1, \dots, x_n)]\!]^{\mathcal{A}}$ hängt nicht nur von der Formel φ ab, sondern auch von dem Tupel $(x_1, \dots, x_n) \in \text{VAR}^n$.

Beispiel 3.38

Die FO[σ_{KINO}]-Formeln $\varphi_2(x_T)$ und $\varphi_3(x_K, x_{St})$ aus Beispiel 3.36 definieren in unserer Beispiel-Datenbank $\mathcal{D}$ die Relationen

$$[\![\varphi_2(x_T)]\!]^{\mathcal{D}} \quad = \quad \left\{ \begin{array}{l} (\,\text{Gravity}\,)\ , \\ (\,\text{Monuments Men}\,) \end{array} \right\}$$

und

$$[\![\varphi_3(x_K, x_{St})]\!]^{\mathcal{D}} \quad = \quad \left\{ \begin{array}{l} (\,\text{Babylon, Kreuzberg}\,)\ , \\ (\,\text{Moviemento, Kreuzberg}\,)\ , \\ (\,\text{Urania, Schöneberg}\,) \end{array} \right\}$$

Ändern der Variablen

Lemma 3.39

Sei σ eine Signatur, sei $\mathcal{A}$ eine σ-Struktur und sei $\varphi(x_1,\dots,x_n) \in \mathrm{FO}[\sigma]$.

(a) *Für jede Permutation*[2] π *von* $[n]$ *ist*

$$\llbracket\varphi(x_{\pi(1)},\dots,x_{\pi(n)})\rrbracket^{\mathcal{A}} \quad = \quad \{\,(a_{\pi(1)},\dots,a_{\pi(n)}) \ : \ (a_1,\dots,a_n) \in \llbracket\varphi(x_1,\dots,x_n)\rrbracket^{\mathcal{A}}\ \}.$$

(b) *Für jede Variable* $y \in \mathrm{VAR} \setminus \{x_1,\dots,x_n\}$ *ist*

$$\llbracket\varphi(x_1,\dots,x_n,y)\rrbracket^{\mathcal{A}} \quad = \quad \llbracket\varphi(x_1,\dots,x_n)\rrbracket^{\mathcal{A}} \ \times \ A.$$

(c) *Falls* $x_n \notin \mathrm{frei}(\varphi)$, *so ist*

$$\llbracket\varphi(x_1,\dots,x_{n-1})\rrbracket^{\mathcal{A}} \quad = \quad \{\,(a_1,\dots,a_{n-1}) \ : \ \text{es gibt (mind.) ein } a \in A \text{ so dass } (a_1,\dots,a_{n-1},a) \in \llbracket\varphi(x_1,\dots,x_n)\rrbracket^{\mathcal{A}}\,\}.$$

[2] Eine Permutation einer Menge M ist eine bijektive Abbildung von M nach M.

Rekursive Beschreibung von $[\![\varphi(x_1,\ldots,x_n)]\!]^{\mathcal{A}}$

Beobachtung 3.40

Ist σ eine Signatur und $\mathcal{A}$ eine σ-Struktur, so können wir für FO[σ]-Formeln φ und Variablentupel $(x_1,\ldots,x_n)$ mit $\text{frei}(\varphi) \subseteq \{x_1,\ldots,x_n\}$ die Relation $[\![\varphi(x_1,\ldots,x_n)]\!]^{\mathcal{A}} \subseteq A^n$ rekursiv wie folgt beschreiben:

- Falls φ von der Form $t_1 = t_2$ für σ-Terme t_1, t_2 ist, so ist

$$[\![\varphi(x_1,\ldots,x_n)]\!]^{\mathcal{A}} \;=\; \{\, (a_1,\ldots,a_n) \in A^n : \; t_1^{\mathcal{A}}[a_1,\ldots,a_n] \;=\; t_2^{\mathcal{A}}[a_1,\ldots,a_n] \,\}$$

  Zur Erinnerung: Für einen σ-Term $t(x_1,\ldots,x_n)$ schreiben wir $t^{\mathcal{A}}[a_1,\ldots,a_n]$ um das Element $[\![t]\!]^{(\mathcal{A},\beta)} \in A$ zu bezeichnen, wobei β eine Belegung mit $\beta(x_i) = a_i$, für alle $i \in [n]$, ist.

- Falls φ von der Form $R(t_1,\ldots,t_k)$ für ein $R \in \sigma$, für $k := \text{ar}(R)$ und für σ-Terme $t_1,\ldots,t_k$ ist, so ist

$$[\![\varphi(x_1,\ldots,x_n)]\!]^{\mathcal{A}} \;=\; \{\, (a_1,\ldots,a_n) \in A^n : \; \big(\, t_1^{\mathcal{A}}[a_1,\ldots,a_n],\; \ldots,\; t_k^{\mathcal{A}}[a_1,\ldots,a_n] \,\big) \in R^{\mathcal{A}} \,\}$$

- Falls φ von der Form $\neg\psi$ ist, so ist

$$[\![\varphi(x_1,\ldots,x_n)]\!]^{\mathcal{A}} \quad = \quad A^n \setminus [\![\psi(x_1,\ldots,x_n)]\!]^{\mathcal{A}}$$

- Falls φ von der Form $(\psi_1 \wedge \psi_2)$ ist, so ist

$$[\![\varphi(x_1,\ldots,x_n)]\!]^{\mathcal{A}} \quad = \quad [\![\psi_1(x_1,\ldots,x_n)]\!]^{\mathcal{A}} \cap [\![\psi_2(x_1,\ldots,x_n)]\!]^{\mathcal{A}}$$

- Falls φ von der Form $(\psi_1 \vee \psi_2)$ ist, so ist

$$[\![\varphi(x_1,\ldots,x_n)]\!]^{\mathcal{A}} \quad = \quad [\![\psi_1(x_1,\ldots,x_n)]\!]^{\mathcal{A}} \cup [\![\psi_2(x_1,\ldots,x_n)]\!]^{\mathcal{A}}$$

- Falls φ von der Form $(\psi_1 \rightarrow \psi_2)$ ist, so ist

$$[\![\varphi(x_1,\ldots,x_n)]\!]^{\mathcal{A}} \quad = \quad [\![\neg\psi_1(x_1,\ldots,x_n)]\!]^{\mathcal{A}} \cup [\![\psi_2(x_1,\ldots,x_n)]\!]^{\mathcal{A}}$$

- Falls φ von der Form $\exists y\ \psi$ ist, so ist

$$[\![\varphi(x_1,\ldots,x_n)]\!]^{\mathcal{A}} \;=\; \{\ (a_1,\ldots,a_n) \in A^n :\ \text{es gibt (mind.) ein } b \in A \text{ mit } (a_1,\ldots,a_n,b) \in [\![\psi(x_1,\ldots,x_n,y)]\!]^{\mathcal{A}}\ \}$$

Somit ist $[\![\varphi(x_1,\ldots,x_n)]\!]^{\mathcal{A}}$ die Projektion von $[\![\psi(x_1,\ldots,x_n,y)]\!]^{\mathcal{A}}$ auf die ersten n Stellen.

- Falls φ von der Form $\forall y\ \psi$ ist, so ist

$$[\![\varphi(x_1,\ldots,x_n)]\!]^{\mathcal{A}} \;=\; \{\ (a_1,\ldots,a_n) \in A^n :\ \text{für jedes } b \in A \text{ ist } (a_1,\ldots,a_n,b) \in [\![\psi(x_1,\ldots,x_n,y)]\!]^{\mathcal{A}}\ \}$$

Das Auswertungsproblem für FO

Eingabe:	Eine endliche Signatur σ, eine σ-Struktur $\mathcal{A}$, deren Universum A endlich ist, eine FO[σ]-Formel φ, eine Zahl $n \in \mathbb{N}$ und ein Variablentupel $(x_1, \ldots, x_n) \in \mathrm{VAR}^n$, so dass $\mathrm{frei}(\varphi) \subseteq \{x_1, \ldots, x_n\}$ ist.
Aufgabe:	Berechne $[\![\varphi(x_1, \ldots, x_n)]\!]^{\mathcal{A}}$.

Beobachtung 3.40 führt unmittelbar zu einem rekursiven Algorithmus, der das Auswertungsproblem für FO löst.

Eine Laufzeitanalyse zeigt, dass Folgendes gilt:

Satz 3.41

Es gibt einen Algorithmus, der das Auswertungsproblem für FO *bei Eingabe einer Signatur σ, eine σ-Struktur $\mathcal{A}$, einer* $\mathrm{FO}[\sigma]$*-Formel φ, einer Zahl n und eines Variablentupels $(x_1, \ldots, x_n)$ mit* $\mathrm{frei}(\varphi) \subseteq \{x_1, \ldots, x_n\}$ *in Zeit*

$$O\left(\|\varphi\| + \|\mathcal{A}\| + \|\varphi\| \cdot w \cdot \|\mathcal{A}\|^{w} \right)$$

löst, wobei gilt:

- $\|\varphi\|$ *ist die Länge von φ, aufgefasst als Wort über dem Alphabet* $A_{\mathrm{FO}[\sigma]}$
- *w ist die maximale Anzahl freier Variablen in Subformeln von φ — die so genannte Breite (engl.: width) von φ*
- $\|\mathcal{A}\|$ *ist ein Maß für die Größe einer geeigneten Repräsentation von $\mathcal{A}$ als Eingabe für einen Algorithmus; präzise:*

$$\|\mathcal{A}\| := |\sigma| + \sum_{R \in \sigma} |R^{\mathcal{A}}| \cdot \mathrm{ar}(R) + \sum_{f \in \sigma} |A|^{\mathrm{ar}(f)} \cdot (\mathrm{ar}(f)+1)$$

(Hier ohne Beweis)

Abschnitt 3.7:

Äquivalenz von Formeln der Logik erster Stufe

Äquivalenz

Definition 3.42

Sei σ eine Signatur.

(a) Zwei FO[σ]-Formeln φ und ψ heißen äquivalent (kurz: $\varphi \equiv \psi$), wenn für jede σ-Interpretation $\mathcal{I}$ gilt:

$$\mathcal{I} \models \varphi \iff \mathcal{I} \models \psi.$$

(b) Zwei Formelmengen $\Phi, \Psi \subseteq \mathrm{FO}[\sigma]$ heißen äquivalent (kurz: $\Phi \equiv \Psi$), wenn für jede σ-Interpretation $\mathcal{I}$ gilt:[3]

$$\mathcal{I} \models \Phi \iff \mathcal{I} \models \Psi.$$

[3] Zur Erinnerung: $\mathcal{I} \models \Phi$ bedeutet, dass $\mathcal{I} \models \varphi$ für jede Formel $\varphi \in \Phi$ gilt.

Beispiel 3.43

Welche der folgenden Formeln sind äquivalent, welche nicht?

- $\varphi_1 := \ \exists y\ E(x,y)$
- $\varphi_2 := \ \exists z\ E(x,z)$
- $\varphi_3 := \ \exists z\ E(y,z)$

Aussagenlogische Äquivalenzen

Lemma 3.44

Ersetzt man in äquivalenten aussagenlogischen Formeln alle Aussagenymbole durch FO[σ]*-Formeln, so erhält man äquivalente* FO[σ]*-Formeln.*

Beispiel

Aus der aussagenlogische Äquivalenz $(X \to Y) \equiv \neg X \lor Y$ folgt, dass

$$(\varphi \to \psi) \equiv \neg\varphi \lor \psi$$

für alle FO[σ]-Formeln φ und ψ gilt.

Quantoren und Negation

Man sieht leicht, dass Folgendes gilt:

Lemma 3.45

Für alle FO[σ]*-Formeln* φ *und alle Variablen* $x \in$ VAR *gilt:*

$$\neg\,\exists x\,\varphi \;\equiv\; \forall x\,\neg\varphi \qquad \textit{und} \qquad \neg\,\forall x\,\varphi \;\equiv\; \exists x\,\neg\varphi.$$

Beweis: Folgt direkt aus der Definition der Semantik (Details: Übung). $\square$

Das Ersetzungslemma

Lemma 3.46

Sei σ eine beliebige Signatur und sei φ eine FO[σ]*-Formel.*
Ist φ' eine FO[σ]*-Formel, die aus φ entsteht, indem man eine Subformel ψ von φ durch eine zu ψ äquivalente* FO[σ]*-Formel ψ' ersetzt, so ist $\varphi \equiv \varphi'$.*

Beweis: Übung.

Satz 3.47

Jede FO[σ]*-Formel ist äquivalent zu einer* FO[σ]*-Formel, in der*

(a) *keiner der Junktoren $\{\wedge, \rightarrow\}$ vorkommt*
(d.h., es kommen nur die Junktoren $\neg, \vee$ und die Quantoren $\exists, \forall$ vor).

(b) *nur Existenzquantoren und die Junktoren $\neg, \vee$ vorkommen.*

(c) *nur Existenzquantoren und die Junktoren $\neg, \wedge$ vorkommen.*

(d) *nur Allquantoren und die Junktoren $\neg, \vee$ vorkommen.*

(e) *nur Allquantoren und die Junktoren $\neg, \wedge$ vorkommen.*

Daher genügt es, bei Beweisen per Induktion über den Aufbau von Formeln von nun an im Induktionsschritt i.d.R. nur noch die Fälle für $\exists$, $\neg$, $\vee$ zu betrachten.

Abschnitt 3.8:

Ehrenfeucht-Fraïssé-Spiele

In diesem Abschnitt werden Ehrenfeucht-Fraïssé-Spiele (kurz: EF-Spiele) eingeführt. Diese liefern ein Werkzeug, mit dessen Hilfe man zeigen kann, dass bestimmte Anfragen oder Klassen von Strukturen nicht in Logik erster Stufe definiert werden können.

Der Einfachheit halber betrachten wir hier nur Signaturen, die keine Funktionssymbole und keine Konstantensymbole enthalten. Solche Signaturen werden im Folgenden relationale Signaturen genannt.

Außerdem werden wir im Folgenden bei zwei gegebenen Strukturen $\mathcal{A}$ und $\mathcal{B}$ immer o.B.d.A. annehmen, dass ihre Universen disjunkt sind, d.h. $A \cap B = \emptyset$.

Das m-Runden EF-Spiel

Sei σ eine relationale Signatur und seien $\mathcal{A}, \mathcal{B}$ zwei σ-Strukturen.

Für $k \in \mathbb{N}$ seien $\overline{a} := a_1, \ldots, a_k \in A$ und $\overline{b} := b_1, \ldots, b_k \in B$ Folgen der Länge k von Elementen aus A bzw. B.

Sei $m \in \mathbb{N}$.

Das m-Runden EF-Spiel auf $(\mathcal{A}, \overline{a})$ und $(\mathcal{B}, \overline{b})$ (bzw. auf $\mathcal{A}$ und $\mathcal{B}$, falls $k = 0$ ist) wird gemäß folgender Spielregeln gespielt:

Spielregeln des m-Runden EF-Spiels auf $(\mathcal{A}, \overline{a})$ und $(\mathcal{B}, \overline{b})$

- Es gibt 2 Spieler, genannt Spoiler (kurz: *Sp*) und Duplicator (kurz: *Dupl*).
- Das Spielbrett besteht aus $(\mathcal{A}, \overline{a})$ und $(\mathcal{B}, \overline{b})$.
- Eine Partie des Spiels besteht aus m Runden.

  In jeder Runde $i \in \{1, \ldots, m\}$ geschieht Folgendes:

  1. Zunächst wählt Spoiler entweder ein Element in A, das im Folgenden mit a_{k+i} bezeichnet wird, oder er wählt ein Element in B, das im Folgenden mit b_{k+i} bezeichnet wird.

     Beachte: Insbes. kann Spoiler in jeder Runde neu entscheiden, in welcher der beiden Strukturen er ein Element wählen möchte.

  2. Danach antwortet Duplicator mit einem Element aus dem Universum der anderen Struktur, d.h. er wählt ein $b_{k+i} \in B$, falls Spoiler ein $a_{k+i} \in A$ gewählt hat, bzw. ein Element $a_{k+i} \in A$, falls Spoiler ein $b_{k+i} \in B$ gewählt hat.

  Nach Runde m ist die Partie beendet und der Gewinner wird wie folgt ermittelt:

Gewinnbedingung

Duplicator hat gewonnen, falls die beiden folgenden Bedingungen erfüllt sind.

(1) Für alle $j, j' \in \{1, \ldots, k+m\}$ gilt: $a_j = a_{j'} \iff b_j = b_{j'}$.

(2) Die Abbildung $\pi : \{a_1, \ldots, a_{k+m}\} \to \{b_1, \ldots, b_{k+m}\}$ mit

$$\pi(a_j) := b_j, \qquad \text{für jedes } j \in \{1, \ldots, k+m\}$$

ist ein partieller Isomorphismus von $\mathcal{A}$ nach $\mathcal{B}$ (siehe Definition 3.48).

Spoiler hat gewonnen, falls mindestens eine der beiden obigen Bedingungen verletzt ist.

Definition 3.48 (partieller Isomorphismus)

Sei σ eine relationale Signatur, seien $\mathcal{A}, \mathcal{B}$ zwei σ-Strukturen, sei $X \subseteq A$. Eine Abbildung $\pi : X \to B$ heißt partieller Isomorphismus von $\mathcal{A}$ nach $\mathcal{B}$, falls gilt:

(1) π ist injektiv und

(2) für jedes $R \in \sigma$, für $r := \mathrm{ar}(R)$ und für alle $(x_1, \ldots, x_r) \in X^r$ gilt:

$$(x_1, \ldots, x_r) \in R^{\mathcal{A}} \iff (\pi(x_1), \ldots, \pi(x_r)) \in R^{\mathcal{B}}.$$

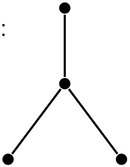
Beispiel 3.49

Sei $\sigma := \{ E/2 \}$ und sei $k := 0$.

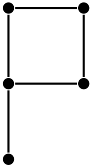
In den folgenden Darstellungen von Graphen repräsentiert jede ungerichtete Kante zwischen Knoten x, y die beiden gerichteten Kanten (x, y) und (y, x).

(a) Betrachte die folgenden beiden Graphen $\mathcal{A}, \mathcal{B}$.

$\mathcal{A}$:

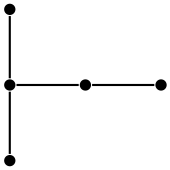


$\mathcal{B}$:

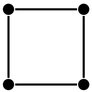


(b) Betrachte die beiden folgenden Graphen $\mathcal{A}, \mathcal{B}$.

$\mathcal{A}$:



$\mathcal{B}$:



Die Ziele von Spoiler und Duplicator

Die Gewinnbedingung im EF-Spiel ist so gewählt, dass die Ziele von Spoiler und Duplicator anschaulich folgendermaßen beschrieben werden können:

- Spoilers Ziel ist es, zu zeigen, dass die beiden Strukturen $(\mathcal{A}, \overline{a})$ und $(\mathcal{B}, \overline{b})$ verschieden sind.
- Duplicators Ziel ist es, einen etwaigen Unterschied zwischen den beiden Strukturen zu vertuschen.

Gewinnstrategien

Eine Strategie für einen der beiden Spieler im m-Runden EF-Spiel auf $(\mathcal{A}, \overline{a})$ und $(\mathcal{B}, \overline{b})$ ist eine Vorschrift, die ihm sagt, welchen Zug er als Nächstes machen soll. Formal:

- Eine Strategie für Spoiler ist eine Abbildung

$$f_{Sp} \ : \ \bigcup_{i=0}^{m-1} (A \times B)^i \ \longrightarrow \ A \cup B.$$

Sind $a_{k+1}, \ldots, a_{k+i} \in A$ und $b_{k+1}, \ldots, b_{k+i} \in B$ die in den ersten i Runden gewählten Elemente, so gibt

$$f_{Sp}(a_{k+1}, b_{k+1}, \ldots, a_{k+i}, b_{k+i})$$

an, welches Element Spoiler in der $(i{+}1)$-ten Runde wählen soll.

- Eine Strategie für Duplicator ist eine Abbildung

$$f_{Dupl} \ : \ \bigcup_{i=0}^{m-1} (A \times B)^i \times (A \cup B) \ \longrightarrow \ B \cup A,$$

so dass für alle $i \in \{0, \ldots, m-1\}$, alle $a_{k+1}, \ldots, a_{k+i} \in A$, alle $b_{k+1}, \ldots, b_{k+i} \in B$ und alle $c_{k+i+1} \in A \cup B$ gilt:

$$c_{k+i+1} \in A \quad \iff \quad f_{Dupl}(a_{k+1}, b_{k+1}, \ldots, a_{k+i}, b_{k+i}, c_{k+i+1}) \ \in \ B.$$

Sind $a_{k+1}, \ldots, a_{k+i} \in A$ und $b_{k+1}, \ldots, b_{k+i} \in B$ die in den ersten i Runden und ist $c_{k+i+1} \in A \cup B$ das von Spoiler in Runde $i+1$ gewählte Element, so gibt

$$f_{Dupl}(a_{k+1}, b_{k+1}, \ldots, a_{k+i}, b_{k+i}, c_{k+i+1})$$

an, welches Element Duplicator in der $(i+1)$-ten Runde wählen soll.

- Eine Gewinnstrategie ist eine Strategie für einen der beiden Spieler, mit der er jede Partie des m-Runden EF-Spiels auf $(\mathcal{A}, \overline{a})$ und $(\mathcal{B}, \overline{b})$ gewinnt.

Der Satz von Ehrenfeucht

Sei σ eine relationale Signatur, seien $\mathcal{A}, \mathcal{B}$ zwei σ-Strukturen, sei $m \in \mathbb{N}$, sei $k \in \mathbb{N}$, sei $\overline{a} = a_1, \ldots, a_k \in A$ und $\overline{b} = b_1, \ldots, b_k \in B$.

Der Satz von Ehrenfeucht besagt, dass die beiden folgenden Aussagen äquivalent sind:

(1) Duplicator hat eine Gewinnstrategie im m-Runden EF-Spiel auf $(\mathcal{A}, \overline{a})$ und $(\mathcal{B}, \overline{b})$.

(2) Für jede FO[σ]-Formel $\varphi(x_1, \ldots, x_k)$ der Quantorentiefe $\leqslant m$ gilt:

$$\mathcal{A} \models \varphi[a_1, \ldots, a_k] \iff \mathcal{B} \models \varphi[b_1, \ldots, b_k].$$

Anschaulich bedeutet dies, dass $(\mathcal{A}, \overline{a})$ und $(\mathcal{B}, \overline{b})$ aus Perspektive von FO[σ]-Formeln der Quantorentiefe $\leqslant m$ „gleich" aussehen, d.h. dass $(\mathcal{A}, \overline{a})$ und $(\mathcal{B}, \overline{b})$ von solchen Formeln nicht unterschieden werden können.

Die Quantorentiefe einer Formel φ ist dabei die maximale Anzahl von ineinander geschachtelten Quantoren, die in φ vorkommen:

Definition 3.50

Die Quantorentiefe (bzw. der Quantorenrang, engl.: quantifier rank) $\text{qr}(\varphi)$ einer FO$[\sigma]$-Formel φ ist rekursiv wie folgt definiert:

- Ist φ atomar, so ist $\text{qr}(\varphi) := 0$.
- Ist φ von der Form $\neg\psi$, so ist $\text{qr}(\varphi) := \text{qr}(\psi)$.
- Ist φ von der Form $(\psi_1 * \psi_2)$ mit $* \in \{\wedge, \vee, \rightarrow\}$, so ist $\text{qr}(\varphi) := \max\{\text{qr}(\psi_1),\ \text{qr}(\psi_2)\}$.
- Ist φ von der Form $\exists x\, \psi$ oder $\forall x\, \psi$, so ist $\text{qr}(\varphi) := \text{qr}(\psi) + 1$.

Beispiele:

- $\text{qr}\left(\exists x \forall y\ (x{=}y \vee E(x,y))\right) = 2$.
- $\text{qr}\left(\exists x\ (E(x,x) \vee \forall y\, \neg E(x,y))\right) = 2$.
- $\text{qr}\left((\exists x\, E(x,x) \vee \forall y\, \neg E(x,y))\right) = 1$.

Bemerkung 3.51

Gemäß Satz 3.47 ist jede FO$[\sigma]$-Formel φ äquivalent zu einer FO$[\sigma]$-Formel φ', in der nur Existenzquantoren und die Junktoren $\neg, \vee$ vorkommen (d.h.: in φ' kommt keins der Symbole $\forall, \wedge, \rightarrow$ vor). Man sieht leicht, dass φ' sogar so gewählt werden kann, dass gilt: $\text{qr}(\varphi') = \text{qr}(\varphi)$ und $\text{frei}(\varphi') = \text{frei}(\varphi)$.

Wir beweisen hier nur die Richtung „(1) $\Longrightarrow$ (2)“ des Satzes von Ehrenfeucht, deren Kontraposition in folgendem Satz formuliert wird.

Satz 3.52 (Satz von Ehrenfeucht, einfache Version)

Sei σ eine relationale Signatur und seien $\mathcal{A}, \mathcal{B}$ zwei σ-Strukturen, sei $m \in \mathbb{N}$, sei $k \in \mathbb{N}$, sei $\overline{a} = a_1, \ldots, a_k \in A$ und sei $\overline{b} = b_1, \ldots, b_k \in B$.

Falls es eine FO[σ]*-Formel* $\varphi(x_1, \ldots, x_k)$ *mit* $\text{frei}(\varphi) \subseteq \{x_1, \ldots, x_k\}$ *und* $\text{qr}(\varphi) \leqslant m$ *gibt, so dass*

$$\mathcal{A} \models \varphi[a_1, \ldots, a_k] \qquad \textit{und} \qquad \mathcal{B} \not\models \varphi[b_1, \ldots, b_k],$$

so hat Spoiler eine Gewinnstrategie im m-Runden EF-Spiel auf $(\mathcal{A}, \overline{a})$ und $(\mathcal{B}, \overline{b})$.

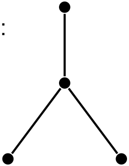
Beweisidee

Zunächst illustrieren wir die Beweisidee an einem Beispiel. Betrachte dazu die Formel

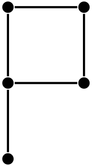
$$\varphi \ := \quad \exists x_1 \forall x_2 \ (\ x_1{=}x_2 \ \lor \ E(x_1, x_2)\ )$$

und die beiden Graphen $\mathcal{A}, \mathcal{B}$ aus Beispiel 3.49(a).

$\mathcal{A}$:



$\mathcal{B}$:



Es gilt: $\mathcal{A} \models \varphi$ und $\mathcal{B} \not\models \varphi$, d.h. $\mathcal{B} \models \neg\varphi$.

Beweis von Satz 3.52:

Per Induktion über den Aufbau von Formeln. Es seien eine relationale Signatur σ und zwei σ-Strukturen $\mathcal{A}$ und $\mathcal{B}$ gegeben. Die Aussage $\mathbb{A}(\varphi)$, die wir für alle FO[σ]-Formeln φ beweisen wollen, besagt Folgendes:

> *Für alle $m, k \in \mathbb{N}$, alle $\overline{a} = a_1, \ldots, a_k \in A$ und alle $\overline{b} = b_1, \ldots, b_k \in B$ gilt:*
>
> *Falls* $\mathrm{qr}(\varphi) \leqslant m$ *und* $|\,\mathrm{frei}(\varphi)| \leqslant k$ *und*
>
> $$\mathcal{A} \models \varphi[a_1, \ldots, a_k] \iff \mathcal{B} \not\models \varphi[b_1, \ldots, b_k],$$
>
> *so hat* Spoiler eine Gewinnstrategie im m-Runden EF-Spiel auf $(\mathcal{A}, \overline{a})$ und $(\mathcal{B}, \overline{b})$.

Um $\mathbb{A}(\varphi)$ für eine gegebene Formel φ zu beweisen, seien im Folgenden $m, k \in \mathbb{N}$, $\overline{a} = a_1, \ldots, a_k \in A$ und $\overline{b} = b_1, \ldots, b_k \in B$ beliebig gewählt. Es genügt, den Fall zu betrachten, in dem gilt:

$$(*): \quad m \geqslant \mathrm{qr}(\varphi), \quad k \geqslant |\,\mathrm{frei}(\varphi)| \quad \text{und} \quad \mathcal{A} \models \varphi[\overline{a}] \iff \mathcal{B} \not\models \varphi[\overline{b}],$$

denn andernfalls muss gemäß der Formulierung von $\mathbb{A}(\varphi)$ nichts gezeigt werden.

Folgerung aus dem Satz von Ehrenfeucht

Notation 3.53

Eine Klasse $\mathfrak{C}$ von σ-Strukturen heißt FO-definierbar, falls es einen FO[σ]-Satz φ gibt, der $\mathfrak{C}$ definiert.

Zur Erinnerung:

Für einen FO[σ]-Satz φ und eine Klasse $\mathfrak{C}$ von σ-Strukturen sagen wir „φ definiert $\mathfrak{C}$“, falls für jede σ-Struktur $\mathcal{A}$ gilt: $\mathcal{A} \in \mathfrak{C} \iff \mathcal{A} \models \varphi$.

Um für eine gegebene Klasse $\mathfrak{C}$ von σ-Strukturen zu zeigen, dass sie nicht FO-definierbar ist, können wir das folgende Korollar nutzen, das wir als eine einfache Folgerung aus Satz 3.52 erhalten.

Korollar 3.54

Sei σ eine relationale Signatur und sei $\mathfrak{C}$ eine Klasse von σ-Strukturen.
Falls es für jedes $m \geqslant 1$ zwei σ-Strukturen $\mathcal{A}_m$ und $\mathcal{B}_m$ gibt, so dass gilt:

1. $\mathcal{A}_m \in \mathfrak{C}$ *und*
2. $\mathcal{B}_m \notin \mathfrak{C}$ *und*
3. *Duplicator hat eine Gewinnstrategie im m-Runden EF-Spiel auf $\mathcal{A}_m$ und $\mathcal{B}_m$,*

dann ist $\mathfrak{C}$ nicht FO-definierbar.

Lineare Ordnungen gerader Kardinalität

Wir werden nun Korollar 3.54 anwenden, um folgenden Satz zu zeigen.

Satz 3.55

Die Klasse EVEN$_\leqslant$, die aus allen linearen Ordnungen $\mathcal{A} = (A, \leqslant^{\mathcal{A}})$ gerader Kardinalität besteht (d.h., A ist endlich und $|A|$ ist durch 2 teilbar), ist nicht FO-definierbar.

Um diesen Satz zu beweisen, genügt es gemäß Korollar 3.54, für jede Rundenzahl $m \geqslant 1$ eine lineare Ordnung $\mathcal{A}_m$ gerader Kardinalität und eine lineare Ordnung $\mathcal{B}_m$ ungerader Kardinalität anzugeben, für die wir zeigen können, dass Duplicator eine Gewinnstrategie im m-Runden EF-Spiel auf $\mathcal{A}_m$ und $\mathcal{B}_m$ hat.

Als Vorbereitung dazu betrachten wir zunächst ein Beispiel.

Beispiel 3.56

Betrachte die linearen Ordnungen $\mathcal{A} = (A, \leqslant^{\mathcal{A}})$ und $\mathcal{B} = (B, \leqslant^{\mathcal{B}})$ mit $A = \{1, \ldots, 8\}$ und $B = \{1, \ldots, 9\}$, wobei $\leqslant^{\mathcal{A}}$ und $\leqslant^{\mathcal{B}}$ die natürlichen linearen Ordnungen auf A und B sind.

Seien außerdem $k := 2$ und $\overline{a} := a_1, a_2$ und $\overline{b} := b_1, b_2$ mit $a_1 = b_1 = 1$ und $a_2 = 8$ und $b_2 = 9$ vorgegeben.

Frage: Was ist die größte Zahl m, so dass Duplicator eine Gewinnstrategie im m-Runden EF-Spiel auf $(\mathcal{A}, \overline{a})$ und $(\mathcal{B}, \overline{b})$ hat?

Die Gewinnstrategie für Duplicator lässt sich zu folgendem Resultat verallgemeinern.

Lemma 3.57

Seien $\mathcal{A}$ und $\mathcal{B}$ endliche lineare Ordnungen, sei $k := 2$, und sei $\overline{a} := a_1, a_2$ und $\overline{b} := b_1, b_2$, wobei a_1, b_1 die kleinsten und a_2, b_2 die größten Elemente in A und B bezüglich $\leqslant^{\mathcal{A}}$ und $\leqslant^{\mathcal{B}}$ sind.

Für jedes $m \geqslant 1$ gilt: Falls $|A|, |B| > 2^m$ oder $|A| = |B|$, so hat Duplicator eine Gewinnstrategie im m-Runden EF-Spiel auf $(\mathcal{A}, \overline{a})$ und $(\mathcal{B}, \overline{b})$.

Wir zeigen nun, dass Duplicator so spielen kann, dass für jedes $i \in \{0, 1, \ldots, m\}$ die folgende Invariante $(*)_i$ erfüllt ist:

$(*)_i$: Sind $a_{2+1}, \ldots, a_{2+i}$ und $b_{2+1}, \ldots, b_{2+i}$ die in den Runden $1, \ldots, i$ gewählten Elemente in A und B, so gilt für alle $j, j' \in \{1, \ldots, 2+i\}$:

1. $a_j \leqslant^{\mathcal{A}} a_{j'} \iff b_j \leqslant^{\mathcal{B}} b_{j'}$ und
2. $Dist(a_j, a_{j'}) = Dist(b_j, b_{j'})$ oder $Dist(a_j, a_{j'}), Dist(b_j, b_{j'}) \geqslant 2^{m-i}$.

Der Beweis folgt per Induktion nach i.

Satz 3.55 folgt nun direkt aus Korollar 3.54 und Lemma 3.57.

Beweis von Satz 3.55.

Um nachzuweisen, dass die Klasse $EVEN_{\leqslant}$ nicht FO-definierbar ist, genügt es laut Korollar 3.54, für jede Zahl $m \geqslant 1$ eine endliche lineare Ordnung $\mathcal{A}_m$ gerader Kardinalität und eine endliche lineare Ordnung $\mathcal{B}_m$ ungerader Kardinalität zu finden, so dass Duplicator eine Gewinnstrategie im m-Runden EF-Spiel auf $\mathcal{A}_m$ und $\mathcal{B}_m$ besitzt.

Wir wählen für $\mathcal{A}_m$ die natürliche lineare Ordnung mit Universum $A_m := \{1, \ldots, 2^m+2\}$, und für $\mathcal{B}_m$ die natürliche lineare Ordnung mit Universum $B_m := \{1, \ldots, 2^m+1\}$.

Gemäß Lemma 3.57 hat Duplicator eine Gewinnstrategie im m-Runden EF-Spiel auf $(\mathcal{A}_m, \overline{a})$ und $(\mathcal{B}_m, \overline{b})$, wobei $\overline{a} = a_1, a_2$ und $\overline{b} = b_1, b_2$ jeweils aus dem kleinsten und dem größten Element der beiden linearen Ordnungen bestehen.

Offensichtlicherweise ist diese Gewinnstrategie auch eine Gewinnstrategie für Duplicator im m-Runden EF-Spiel auf $\mathcal{A}_m$ und $\mathcal{B}_m$. $\square$

Bemerkung 3.58

Der obige Beweis zeigt nicht nur, dass die Klasse $EVEN_{\leqslant}$ nicht FO-definierbar ist, sondern sogar die etwas stärkere Aussage:

> *Es gibt keinen* FO[$\{\leqslant\}$]*-Satz* ψ*, so dass für jede* **endliche lineare Ordnung** $\mathcal{B}$ *gilt:* $\mathcal{B} \models \psi \iff |B|$ *ist gerade.*

Graph-Zusammenhang und Erreichbarkeit sind nicht FO-definierbar

Wir können die Aussage von Bemerkung 3.58 nutzen, um Folgendes zu zeigen.

Satz 3.59

Sei $\sigma := \{E/2\}$.

(a) *„Graph-Zusammenhang ist nicht* FO*-definierbar."*
D.h.: Es gibt keinen FO$[\sigma]$*-Satz* φ_{Conn}, *so dass für jeden endlichen ungerichteten Graphen* $\mathcal{G} = (V^{\mathcal{G}}, E^{\mathcal{G}})$ *und die zugehörige* σ*-Struktur* $\mathcal{A} = (A, E^{\mathcal{A}})$ *gilt:* $\mathcal{A} \models \varphi_{Conn} \iff \mathcal{G}$ *ist zusammenhängend.*

(b) *„Erreichbarkeit ist nicht* FO*-definierbar."*
D.h.: Es gibt keine FO$[\sigma]$*-Formel* $\varphi_{Reach}(x, y)$, *so dass für alle endlichen gerichteten Graphen* $\mathcal{A} = (A, E^{\mathcal{A}})$ *und alle Knoten* $a, b \in A$ *gilt:*
$\mathcal{A} \models \varphi_{Reach}[a, b] \iff$ *es gibt in* $\mathcal{A}$ *einen Weg von Knoten* a *zu Knoten* b*.*

(b) folgt direkt aus *(a)*, denn:

Angenommen $\varphi_{Reach}(x,y)$ wäre eine FO[σ]-Formel, so dass für alle gerichteten Graphen $\mathcal{A} = (A, E^{\mathcal{A}})$ und alle Knoten $a, b \in A$ gilt: $\mathcal{A} \models \varphi_{Reach}[a,b] \iff$ es gibt in $\mathcal{A}$ einen Weg von Knoten a zu Knoten b.

Dann ist

$$\varphi_{Conn} \ := \ \forall x \, \forall y \ \varphi_{Reach}(x,y)$$

ein FO[σ]-Satz, der in einem gerichteten Graphen $\mathcal{A}$ genau dann erfüllt ist, wenn $\mathcal{A}$ stark zusammenhängend ist.

Insbesondere gilt dann für jeden ungerichteten Graphen $\mathcal{G}$ und die zu $\mathcal{G}$ gehörende σ-Struktur $\mathcal{A}$: $\mathcal{A} \models \varphi_{Conn} \iff \mathcal{G}$ ist zusammenhängend.

Dies ist ein Widerspruch zu *(a)*.

Logische Reduktionen

Bemerkung 3.60

Die im Beweis von Satz 3.59 benutzte Vorgehensweise ist unter dem Begriff logische Reduktion (oder Transduktionen) bekannt.

Im Beweis von Teil *(b)* wurde gezeigt: Falls es eine FO[$\{E\}$]-Formel gibt, die ausdrückt, dass Knoten y von Knoten x aus erreichbar ist, dann gibt es auch eine FO[$\{E\}$]-Formel, die Graph-Zusammenhang definiert.

Somit wurde das Problem, einen FO[$\{E\}$]-Satz zu finden, der Graph-Zusammenhang definiert, auf das Problem reduziert, eine FO[$\{E\}$]-Formel zu finden, die ausdrückt, dass Knoten y von Knoten x aus erreichbar ist.

Im Beweis von Teil *(a)* wurde das Problem, einen FO[$\{\leqslant\}$]-Satz zu finden, der ausdrückt, dass eine endliche lineare Ordnung eine gerade Kardinalität besitzt, auf das Problem reduziert, einen FO[$\{E\}$]-Satz zu finden, der Graph-Zusammenhang definiert.

D.h. es wurde gezeigt: Falls Graph-Zusammenhang FO-definierbar ist, so ist auch die Aussage „eine endliche lineare Ordnung besitzt eine gerade Kardinalität“ FO-definierbar.

Dies wurde dadurch erreicht, dass man innerhalb einer linearen Ordnung einen geeigneten Graphen „simuliert“ (bzw. “interpretiert“), indem man die Kantenrelation des Graphen durch eine FO[$\{\leqslant\}$]-Formel beschreibt.

Generell ist diese Methode der logischen Reduktionen oft nützlich, um bereits bekannte Nicht-Definierbarkeits-Resultate auf neue Nicht-Definierbarkeits-Resultate zu übertragen.

Abschnitt 3.9:

Erfüllbarkeit, Allgemeingültigkeit und die Folgerungsbeziehung

Die im Folgenden eingeführten Begriffe der Erfüllbarkeit, Allgemeingültigkeit und der Folgerungsbeziehung sind für die Logik erster Stufe ähnlich definiert wie für die Aussagenlogik.

Im Folgenden sei σ stets eine beliebige Signatur.

Erfüllbarkeit und Allgemeingültigkeit

Definition 3.61

Eine FO[σ]-Formel φ (bzw. eine Formelmenge $\Phi \subseteq \text{FO}[\sigma]$) heißt erfüllbar, wenn es eine σ-Interpretation gibt, die φ (bzw. Φ) erfüllt.

Eine Formel oder Formelmenge, die nicht erfüllbar ist, nennen wir unerfüllbar.

Definition 3.62

Eine FO[σ]-Formel φ heißt allgemeingültig, wenn jede σ-Interpretation die Formel φ erfüllt.

Wir schreiben kurz $\models \varphi$ um auszudrücken, dass φ allgemeingültig ist.

Offensichtlicherweise gilt für alle FO[σ]-Formeln φ:

$$\varphi \text{ ist allgemeingültig} \iff \neg\varphi \text{ ist unerfüllbar.}$$

Verum ($\top$) und Falsum ($\bot$)

Beispiele:

- Die FO[σ]-Formel $\forall v_0\ v_0{=}v_0$ ist allgemeingültig.
- Die FO[σ]-Formel $\exists v_0\ \neg\ v_0{=}v_0$ ist unerfüllbar.

Notation 3.63

Wir schreiben $\top$ (in Worten: Verum), um die allgemeingültige FO-Formel $\forall v_0\ v_0{=}v_0$ zu bezeichnen.

Wir schreiben $\bot$ (in Worten: Falsum), um die unerfüllbare FO-Formel $\exists v_0\ \neg\ v_0{=}v_0$ zu bezeichnen.

Die Folgerungsbeziehung

Definition 3.64

Eine FO[σ]-Formel ψ folgt aus einer Formelmenge $\Phi \subseteq$ FO[σ] (wir schreiben: $\Phi \models \psi$), wenn für jede σ-Interpretation $\mathcal{I}$ gilt:
Falls $\mathcal{I} \models \Phi$, so gilt auch $\mathcal{I} \models \psi$.

Notation

Für zwei FO[σ]-Formeln φ, ψ schreiben wir kurz $\varphi \models \psi$ an Stelle von $\{\varphi\} \models \psi$ und sagen, dass die Formel ψ aus der Formel φ folgt.

Zusammenhänge

Es bestehen ähnliche Zusammenhänge wie bei der Aussagenlogik:

Lemma 3.65 (Allgemeingültigkeit, Unerfüllbarkeit und Folgerung)

Für jede FO[σ]*-Formel* φ *gilt:*

(a) φ *ist allgemeingültig* $\iff$ $\varphi \equiv \top$ $\iff$ $\top \models \varphi$.

(b) φ *ist unerfüllbar* $\iff$ $\varphi \equiv \bot$ $\iff$ $\varphi \models \bot$.

(c) $\models \varphi$ $\iff$ $\emptyset \models \varphi$.
D.h.: φ *ist allgemeingültig* $\iff$ φ *folgt aus der leeren Menge.*

Lemma 3.66 (Erfüllbarkeit und die Folgerungsbeziehung)

(a) *Für alle Formelmengen* $\Phi \subseteq$ FO[σ] *und alle* FO[σ]*-Formeln* ψ *gilt:*

$$\Phi \models \psi \quad \iff \quad \Phi \cup \{\neg\psi\} \text{ ist unerfüllbar.}$$

(b) *Für alle* FO[σ]*-Formeln* φ, ψ *gilt:* $\varphi \equiv \psi \iff\ \models (\varphi \leftrightarrow \psi)$.

Beweis der beiden Lemmas: Analog zu den Beweisen der entsprechenden Resultate in der Aussagenlogik. Details: Übung.

Abschnitt 3.10:

Normalformen

Negationsnormalform

Die Negationsnormalform für Formeln der Logik erster Stufe ist ähnlich definiert wie die Negationsnormalform der Aussagenlogik.

Definition 3.67

Sei σ eine beliebige Signatur. Eine FO[σ]-Formel φ ist in Negationsnormalform (kurz: NNF), wenn Negationszeichen in φ nur unmittelbar vor atomaren Subformeln auftreten und φ den Junktor „$\rightarrow$“ nicht enthält.

Satz 3.68

Jede FO[σ]*-Formel* φ *ist äquivalent zu einer Formel in NNF.*

Beweis.

Gemäß Satz 3.47 können wir o.B.d.A. annehmen, dass φ den Junktor „$\rightarrow$“ nicht enthält.
Ähnlich wie für die Aussagenlogik definieren wir per Induktion über den Aufbau zu jeder FO[σ]-Formel φ zwei FO[σ]-Formeln φ' und φ'' in NNF, so dass gilt:
$\varphi \equiv \varphi'$ und $\neg\varphi \equiv \varphi''$. Details: Übung. $\square$

Pränexe Normalform

Definition 3.69

Sei σ eine beliebige Signatur.

(a) Eine FO[σ]-Formel heißt quantorenfrei, falls in ihr keins der Symbole $\exists, \forall$ vorkommt.
Die Menge aller quantorenfreien FO[σ]-Formeln bezeichnen wir mit QF_σ.

(b) Eine FO[σ]-Formel φ ist in pränexer Normalform (bzw. Pränex-Normalform, kurz: PNF), wenn sie von der Form

$$Q_1 x_1 \cdots Q_n x_n \; \chi$$

ist, wobei $n \geqslant 0$, $Q_1, \ldots, Q_n \in \{\exists, \forall\}$, $x_1, \ldots, x_n \in \mathrm{VAR}$ und $\chi \in \mathrm{QF}_\sigma$.
$Q_1 x_1 \cdots Q_n x_n$ wird Quantoren-Präfix von φ genannt;
χ heißt Kern (bzw. Matrix) von φ.

Satz 3.70

Jede FO[σ]-*Formel* φ *ist äquivalent zu einer* FO[σ]-*Formel* φ' *in pränexer Normalform mit* $\mathrm{frei}(\varphi') = \mathrm{frei}(\varphi)$.

Bevor wir Satz 3.70 beweisen, betrachten wir zunächst ein Beispiel.

Beispiel 3.71

Sei

$$\varphi(y) \ := \quad \forall x \, \neg \big(\exists y \, E(x,y) \ \rightarrow \ \exists x \, E(x,y) \big).$$

Umformung in eine äquivalente Formel in Pränex-Normalform:

Beweis von Satz 3.70:

Wir zeigen zunächst drei Lemmas und schließen danach den Beweis ab.

Lemma 3.72

Sei $\psi := Q_1 x_1 \cdots Q_n x_n\, \chi$, *wobei* $n \geqslant 0$, $Q_1, \ldots, Q_n \in \{\exists, \forall\}$ *und* $\chi \in \mathrm{FO}[\sigma]$. *Für jedes* $Q \in \{\exists, \forall\}$ *sei*

$$\widetilde{Q} \ := \begin{cases} \forall & \textit{falls} \ \ Q = \exists\,, \\ \exists & \textit{falls} \ \ Q = \forall\,. \end{cases}$$

Dann gilt: $\neg\,\psi \quad \equiv \quad \widetilde{Q}_1 x_1 \cdots \widetilde{Q}_n x_n\, \neg\chi$.

Beweis.

Einfaches Nachrechnen per Induktion nach n unter Verwendung der Tatsache, dass $\neg\,\exists x\,\varphi \ \equiv\ \forall x\,\neg\varphi$ und $\neg\,\forall x\,\varphi \ \equiv\ \exists x\,\neg\varphi$ (Lemma 3.45).
Details: Übung. $\square$

Lemma 3.73

Für alle FO[σ]*-Formeln* φ *und* ψ *und für alle Variablen* $x \in \mathrm{VAR} \setminus \mathrm{frei}(\varphi)$ *gilt:*

$$\big(\varphi \ \land \ \exists x\, \psi\big) \ \equiv \ \exists x \,\big(\varphi \land \psi\big) \quad , \quad \big(\varphi \ \land \ \forall x\, \psi\big) \ \equiv \ \forall x \,\big(\varphi \land \psi\big) \ ,$$

$$\big(\varphi \ \lor \ \exists x\, \psi\big) \ \equiv \ \exists x \,\big(\varphi \lor \psi\big) \quad , \quad \big(\varphi \ \lor \ \forall x\, \psi\big) \ \equiv \ \forall x \,\big(\varphi \lor \psi\big) \ .$$

Beweis. Die Beweise aller vier Äquivalenzen sind ähnlich. Wir beweisen hier nur die erste:

$$\big(\varphi \ \land \ \exists x\, \psi\big) \ \equiv \ \exists x \,\big(\varphi \land \psi\big) \ . \tag{1}$$

Lemma 3.74

Seien

$$\psi_1 := Q_1 x_1 \cdots Q_\ell x_\ell \ \chi_1 \qquad \textit{und} \qquad \psi_2 := Q'_1 y_1 \cdots Q'_m y_m \ \chi_2 \ ,$$

wobei $\ell, m \geqslant 0$, $Q_1, \ldots, Q_\ell,\ Q'_1, \ldots, Q'_m \in \{\exists, \forall\}$,
$x_1, \ldots, x_\ell,\ y_1, \ldots, y_m \in \text{VAR}$, $\chi_1, \chi_2 \in \text{FO}[\sigma]$.

Es gelte: $\{x_1, \ldots, x_\ell\} \cap \text{frei}(\psi_2) = \emptyset$ *und* $\{y_1, \ldots, y_m\} \cap \text{frei}(\chi_1) = \emptyset$.

Dann gilt für $* \in \{\wedge, \vee\}$, *dass*

$$(\,\psi_1 \, * \, \psi_2\,) \quad \equiv \quad Q_1 x_1 \cdots Q_\ell x_\ell \ Q'_1 y_1 \cdots Q'_m y_m \ (\,\chi_1 \, * \, \chi_2\,).$$

Beweis.

Zwei Induktionen über ℓ bzw. m unter Verwendung von Lemma 3.73.
Details: Übung. □

Abschluss des Beweises von Satz 3.70:

Sei φ eine FO[σ]-Formel.

Gemäß Satz 3.47 können wir o.B.d.A. annehmen, dass φ den Junktor „$\rightarrow$“ nicht enthält.

Per Induktion über den Aufbau von φ zeigen wir, dass es eine zu φ äquivalente Formel φ' in PNF gibt mit $\text{frei}(\varphi') = \text{frei}(\varphi)$.

Induktionsanfang: Atomare Formeln sind quantorenfrei und daher insbesondere in PNF.

Induktionsschritt:

Kapitel 4:

Grundlagen des automatischen Schließens

Ziel: Automatisches Schließen

- In typischen Anwendungen der Logik beschreibt man mit Hilfe einer Formelmenge das Wissen über ein Anwendungsszenario und will aus diesem Wissen dann, möglichst automatisch, Folgerungen ziehen.
- In diesem Kapitel werden wir untersuchen, inwieweit sich für die Logik erster Stufe das Folgern automatiseren lässt.
- Wir werden einen syntaktischen Beweisbegriff einführen, der genau dem semantischen Folgerungsbegriff entspricht (Vollständigkeitssatz).
- Dadurch werden wir einen Algorithmus erhalten, der nach und nach alle allgemeingültigen Sätze der Logik erster Stufe aufzählt.
- Andererseits werden wir zeigen, dass es keinen Algorithmus gibt, der bei Eingabe eines beliebigen Satzes der Logik erster Stufe entscheidet, ob der Satz allgemeingültig ist.
- Als Folgerung aus dem Vollständigkeitssatz werden wir auch den Endlichkeitssatz für die Logik erster Stufe erhalten.

Abschnitt 4.1:

Kalküle und Ableitungen

Ableitungsregeln und Kalküle

Definition 4.1

Sei M eine beliebige Menge.

(a) Eine Ableitungsregel über M (kurz: Regel) hat die Form

$$\frac{a_1 \quad \cdots \quad a_n}{b}$$

wobei $n \geqslant 0$ und $a_1, \ldots, a_n, b \in M$.

Wir bezeichnen $a_1, \ldots, a_n$ als die Voraussetzungen der Regel und b als die Konsequenz.

Ableitungsregeln ohne Voraussetzungen (also mit $n = 0$) bezeichnen wir als Axiome.

(b) Ein Kalkül über M ist eine Menge von Ableitungsregeln über M.

Ableitungen

Definition 4.2

Sei $\mathfrak{K}$ ein Kalkül über einer Menge M, sei $V \subseteq M$ und sei $a \in M$.

(a) Eine Ableitung von a aus V in $\mathfrak{K}$ ist eine endliche Folge $(a_1, \ldots, a_\ell) \in M^\ell$, so dass $\ell \geqslant 1$, $a_\ell = a$ und für alle $i \in \{1, \ldots, \ell\}$ gilt:

- $a_i \in V$ oder
- $\overline{a_i}$ ist ein Axiom in $\mathfrak{K}$ oder
- es gibt in $\mathfrak{K}$ eine Ableitungsregel $\dfrac{b_1 \ \cdots \ b_n}{a_i}$ so dass $b_1, \ldots, b_n \in \{a_1, \ldots, a_{i-1}\}$.

Der besseren Lesbarkeit halber schreiben wir in konkreten Beispielen Ableitungen der Form $(a_1, \ldots, a_\ell)$ oft zeilenweise, also

$$
\begin{array}{ll}
(1) & a_1 \\
(2) & a_2 \\
\vdots & \\
(\ell) & a_\ell
\end{array}
$$

und geben am Ende jeder Zeile eine kurze Begründung an.

(b) Ein Element $a \in M$ ist aus V in $\mathfrak{K}$ ableitbar, wenn es eine Ableitung von a aus V in $\mathfrak{K}$ gibt.

(c) Wir schreiben $\mathrm{abl}_{\mathfrak{K}}(V)$, um die Menge aller aus V in $\mathfrak{K}$ ableitbaren Elemente zu bezeichnen.

(d) Für $V = \emptyset$ nutzen wir folgende Notationen:

Eine Ableitung von a in $\mathfrak{K}$ ist eine Ableitung von a aus $\emptyset$ in $\mathfrak{K}$.

Ein Element $a \in M$ heißt ableitbar aus $\mathfrak{K}$, falls es eine Ableitung von a in $\mathfrak{K}$ gibt.

Die Menge aller in $\mathfrak{K}$ ableitbaren Elemente bezeichnen wir mit $\mathrm{abl}_{\mathfrak{K}}$, d.h.: $\mathrm{abl}_{\mathfrak{K}} := \mathrm{abl}_{\mathfrak{K}}(\emptyset)$.

Wir werden Kalküle nutzen, um auf elegante Art rekursive Definitionen bestimmter Mengen anzugeben:

Um eine bestimmte Teilmenge A einer Menge M rekursiv zu definieren, genügt es, einen Kalkül $\mathfrak{K}$ über M anzugeben, für den gilt: $\mathrm{abl}_{\mathfrak{K}} = A$.

Beispiel: Mengen natürlicher Zahlen

Beispiel 4.3

Sei $\mathfrak{K}$ der Kalkül über $M := \mathbb{N}$ mit folgenden Ableitungsregeln:

- Axiom: $\dfrac{}{1}$
- Weitere Regeln: $\dfrac{n}{2n}$, für jedes $n \in \mathbb{N}$.

Fragen:

- Was ist $\mathrm{abl}_{\mathfrak{K}}$?
- Was ist $\mathrm{abl}_{\mathfrak{K}}(V)$ für $V := \{3\}$?

Beispiel: Aussagenlogik

Beispiel 4.4

Sei $\Sigma := A_{\mathsf{AL}}$ das Alphabet der Aussagenlogik, d.h.

$$\Sigma \;=\; \mathsf{AS} \,\cup\, \{\,\neg,\ \wedge,\ \vee,\ \rightarrow,\ \mathbf{0},\ \mathbf{1},\ (,\ )\,\}\ ,$$

wobei $\mathsf{AS} = \{A_i \ :\ i \in \mathbb{N}\}$ die Menge aller Aussagensymbole ist.

Gesucht: Ein Kalkül $\mathfrak{K}$ über $M := \Sigma^*$, aus dem genau die syntaktisch korrekten aussagenlogischen Formeln ableitbar sind, d.h. $\mathsf{abl}_{\mathfrak{K}} = \mathsf{AL}$.

Beispiel: Resolution

Die Kalkül-Schreibweise lässt sich auch dazu nutzen, eine elegante Darstellung der Resolutionswiderlegungen zu anzugeben.

Zur Erinnerung:

- Eine Klausel ist eine endliche Menge von Literalen.

  Ein Literal ist eine aussagenlogische Formel der Form X oder $\neg X$, wobei $X \in \text{AS}$.

- Wir haben in Satz 2.60 gezeigt, dass für jede Menge Γ von Klauseln gilt:

  $$\Gamma \text{ ist unerfüllbar} \quad \iff \quad \Gamma \vdash_R \emptyset.$$

  Hierbei ist $\emptyset$ die leere Klausel.

  „$\Gamma \vdash_R \emptyset$“ bedeutet, dass es eine Resolutionswiderlegung von Γ gibt.

  Zur Erinnerung hier die Definition des Begriffs der Resolutionswiderlegungen:

Resolutionsableitungen und -widerlegungen

Definition

Sei Γ eine Klauselmenge.

(a) Eine Resolutionsableitung einer Klausel δ aus Γ ist ein Tupel $(\delta_1, \ldots, \delta_\ell)$ von Klauseln, so dass gilt: $\ell \geqslant 1$, $\delta_\ell = \delta$, und für alle $i \in [\ell]$ ist

- $\delta_i \in \Gamma$, oder
- es gibt $j, k \in [i{-}1]$, so dass δ_i eine Resolvente von δ_j und δ_k ist.

(b) Eine Resolutionswiderlegung von Γ ist eine Resolutionsableitung der leeren Klausel aus Γ.

Zur Erinnerung:

Eine Klausel δ ist genau dann eine Resolvente zweier Klauseln γ_1 und γ_2, wenn es ein Literal λ gibt, so dass gilt:

$$\lambda \in \gamma_1, \qquad \overline{\lambda} \in \gamma_2 \qquad \text{und} \qquad \delta = \big(\gamma_1 \setminus \{\lambda\}\big) \cup \big(\gamma_2 \setminus \{\overline{\lambda}\}\big).$$

Der Resolutionskalkül der Aussagenlogik

Gesucht: Ein Kalkül $\mathfrak{K}_R$ über der Menge aller Klauseln, so dass für jede Klauselmenge Γ und jede Klausel δ gilt:

$$\delta \in \mathrm{abl}_{\mathfrak{K}_R}(\Gamma) \quad \Longleftrightarrow \quad \Gamma \vdash_R \delta$$

d.h.: δ ist genau dann aus Γ in $\mathfrak{K}_R$ ableitbar, wenn es eine Resolutionsableitung von δ aus Γ gibt.

Der Kalkül $\mathfrak{K}_R$ wird Resolutionskalkül der Aussagenlogik genannt.

Kalküle und abgeschlossene Mengen

Definition 4.5

Sei $\mathfrak{K}$ ein Kalkül über einer Menge M.
Eine Menge $A \subseteq M$ heißt abgeschlossen unter $\mathfrak{K}$, wenn für jede Ableitungsregel

$$\frac{a_1 \quad \cdots \quad a_n}{b}$$

in $\mathfrak{K}$ gilt: Falls $a_1, \ldots, a_n \in A$, so ist auch $b \in A$.

Satz 4.6

Sei $\mathfrak{K}$ ein Kalkül über einer Menge M und sei $V \subseteq M$.
Dann ist $\mathrm{abl}_{\mathfrak{K}}(V)$ die bzgl. „$\subseteq$“ kleinste unter $\mathfrak{K}$ abgeschlossene Menge, die V enthält.
D.h. es gilt:

(a) $V \subseteq \mathrm{abl}_{\mathfrak{K}}(V)$.

(b) $\mathrm{abl}_{\mathfrak{K}}(V)$ *ist abgeschlossen unter* $\mathfrak{K}$.

(c) *Für jede Menge A mit $V \subseteq A \subseteq M$ gilt:*
Falls A abgeschlossen ist unter $\mathfrak{K}$, so ist $\mathrm{abl}_{\mathfrak{K}}(V) \subseteq A$.

(d) $$\mathrm{abl}_{\mathfrak{K}}(V) \;=\; \bigcap_{\substack{V \subseteq A \subseteq M,\\ A \text{ abgeschlossen unter } \mathfrak{K}}} A\,.$$

Induktionsprinzip für die ableitbaren Elemente eines Kalküls

Sei $\mathfrak{K}$ ein Kalkül über einer Menge M und sei $V \subseteq M$. Um zu zeigen, dass eine bestimmte Aussage $\mathbb{A}(a)$ für alle aus V in $\mathfrak{K}$ ableitbaren Elemente a gilt, können wir das Induktionsprinzip nutzen und einfach Folgendes zeigen:

(1) Die Aussage $\mathbb{A}(a)$ gilt für jedes $a \in V$, und

(2) für jede Ableitungsregel

$$\frac{a_1 \quad \cdots \quad a_n}{b}$$

in $\mathfrak{K}$ gilt: Falls $\mathbb{A}(a_i)$ für jedes $i \in [n]$ gilt, so gilt auch $\mathbb{A}(b)$.

Daraus folgt laut dem nächsten Lemma dann, dass $\mathbb{A}(a)$ für jedes $a \in \mathrm{abl}_{\mathfrak{K}}(V)$ gilt.

Lemma 4.7

Sei $\mathfrak{K}$ ein Kalkül über einer Menge M und sei $V \subseteq M$. Falls

(1) *eine Aussage $\mathbb{A}(a)$ für jedes $a \in V$ gilt und*

(2) *für jede Ableitungsregel*

$$\frac{a_1 \quad \cdots \quad a_n}{b}$$

in $\mathfrak{K}$ gilt: falls $\mathbb{A}(a_i)$ für jedes $i \in [n]$ gilt, so gilt auch $\mathbb{A}(b)$,

dann gilt die Aussage $\mathbb{A}(a)$ für jedes $a \in \mathrm{abl}_{\mathfrak{K}}(V)$.

Beweis.

Es seien (1) und (2) erfüllt.

Betrachte die Menge

$$A \ := \ \{\, a \in M \ : \ \text{die Aussage } \mathbb{A}(a) \text{ gilt} \,\}\ .$$

Wegen (1) ist $V \subseteq A$.

Wegen (2) ist A abgeschlossen unter $\mathfrak{K}$.

Aus Satz 4.6 folgt daher: $\mathrm{abl}_{\mathfrak{K}}(V) \subseteq A$.

Somit gilt die Aussage $\mathbb{A}(a)$ für jedes $a \in \mathrm{abl}_{\mathfrak{K}}(V)$. $\square$

Abschnitt 4.2:

Ein Beweiskalkül für die Logik erster Stufe — der Vollständigkeitssatz

Notation

- In diesem Kapitel sei σ eine beliebige fest gewählte Signatur.
- Der Einfachheit halber werden wir o.B.d.A. in diesem Kapitel nur FO[σ]-Formeln betrachten, in denen das Symbol „$\rightarrow$“ nicht vorkommt.
- $t, u, t_1, t_2, t', u', u'', \ldots$ bezeichnen immer σ-Terme.
- $\varphi, \psi, \chi, \ldots$ bezeichnen immer FO[σ]-Formeln.
- $\Phi, \Psi, \Phi_1, \Phi_2, \Psi', \ldots$ bezeichnen immer Mengen von FO[σ]-Formeln.
- $\Gamma, \Delta, \Gamma', \Delta_1, \Delta_2, \ldots$ bezeichnen immer endliche Mengen von FO[σ]-Formeln.
- Für $\Phi \subseteq \text{FO}[\sigma]$ ist $\text{frei}(\Phi) := \bigcup_{\varphi \in \Phi} \text{frei}(\varphi)$.

  Manchmal schreiben wir auch $\text{frei}(\Phi, \varphi)$ an Stelle von $\text{frei}(\Phi \cup \{\varphi\})$.
- Ist M eine Menge, so schreiben wir $L \subseteq_e M$, um auszudrücken, dass L eine endliche Teilmenge von M ist.

Sequenzen

Definition 4.8

(a) Eine Sequenz ist ein Ausdruck der Form

$$\Gamma \vdash \psi$$

wobei $\psi \in \mathrm{FO}[\sigma]$ und $\Gamma \subseteq_e \mathrm{FO}[\sigma]$ (d.h., Γ ist eine endliche Menge von $\mathrm{FO}[\sigma]$-Formeln).

Wir bezeichnen Γ als das Antezedens und ψ als das Sukzedens der Sequenz $\Gamma \vdash \psi$.

(b) Wir schreiben M_S um die Menge aller Sequenzen zu bezeichnen, d.h.:

$$M_S \;:=\; \{\, \Gamma \vdash \psi \;:\; \Gamma \subseteq_e \mathrm{FO}[\sigma],\ \psi \in \mathrm{FO}[\sigma] \,\}.$$

Korrektheit einer Sequenz

Definition 4.9

Eine Sequenz $\Gamma \vdash \psi$ heißt korrekt, falls gilt: $\Gamma \models \psi$.

Zur Erinnerung: $\Gamma \models \psi$ bedeutet:

Für jede σ-Interpretation $\mathcal{I}$ gilt: Falls $\mathcal{I} \models \Gamma$, so auch $\mathcal{I} \models \psi$.

Beispiel:

Welche der folgenden Sequenzen sind korrekt für alle $\varphi, \psi \in \mathsf{FO}[\sigma]$ und alle $x, y \in \mathsf{VAR}$; welche sind nicht korrekt?

(1) $\{\, (\neg\varphi \vee \psi)\ ,\ \varphi \,\} \ \vdash\ \psi$

(2) $\emptyset \ \vdash\ (\varphi \vee \neg\varphi)$

(3) $\{\, \exists x \forall y\, \varphi \,\} \ \vdash\ \forall y \exists x\, \varphi$

(4) $\{\, \forall y \exists x\ x{=}y \,\} \ \vdash\ \exists x \forall y\ x{=}y$

Ziel

Wir wollen im Folgenden einen Kalkül $\mathfrak{K}$ über M_S angeben, so dass gilt:

(1) $\mathfrak{K}$ ist korrekt, d.h. jede in $\mathfrak{K}$ ableitbare Sequenz ist korrekt.

(2) $\mathfrak{K}$ ist vollständig, d.h. jede korrekte Sequenz ist in $\mathfrak{K}$ ableitbar.

(3) $\mathfrak{K}$ ist effektiv, d.h. es gibt einen Algorithmus, der nach und nach genau die aus $\mathfrak{K}$ ableitbaren Sequenzen aufzählt.

Dies liefert dann insbesondere einen Algorithmus, der nach und nach alle allgemeingültigen FO[σ]-Formeln aufzählt: Dazu lasse den gemäß (3) existierenden Algorithmus laufen, und immer wenn dieser eine Sequenz der Form $\Gamma \vdash \psi$ mit $\Gamma = \emptyset$ ausgeben will, gib ψ aus.

Wegen (1) ist die Sequenz dann korrekt, d.h. es gilt $\emptyset \models \psi$, und daher ist ψ allgemeingültig.

Wegen (2) werden tatsächlich <u>alle</u> allgemeingültigen FO[σ]-Formeln aufgezählt.

Notationen für Sequenzen

Wir schreiben kurz

- $\Gamma, \varphi \vdash \psi$, um die Sequenz $\Gamma \cup \{\varphi\} \vdash \psi$ zu bezeichnen.
- $\varphi_1, \ldots, \varphi_n \vdash \psi$, um die Sequenz $\{\varphi_1, \ldots, \varphi_n\} \vdash \psi$ zu bezeichnen.
- $\vdash \psi$, um die Sequenz $\emptyset \vdash \psi$ zu bezeichnen.

Sequenzenregeln

Eine Sequenzenregel ist eine Ableitungsregel über M_S.

Sequenzenregeln der Form

$$\frac{a_1 \quad \cdots \quad a_n}{b}$$

schreiben wir meistens zeilenweise, als

$$\begin{array}{c} a_1 \\ \vdots \\ a_n \\ \hline b \end{array}$$

wobei jedes a_i eine Sequenz der Form $\Gamma_i \vdash \psi_i$ ist,
und b eine Sequenz der Form $\Delta \vdash \varphi$ ist.

Definition 4.10

Eine Sequenzenregel

$$\frac{\begin{array}{c}\Gamma_1 \vdash \psi_1 \\ \vdots \\ \Gamma_n \vdash \psi_n\end{array}}{\Delta \vdash \varphi}$$

heißt korrekt, wenn Folgendes gilt: Sind die Sequenzen $\Gamma_i \vdash \psi_i$ für alle $i \in \{1, \ldots, n\}$ korrekt, so ist auch die Sequenz $\Delta \vdash \varphi$ korrekt.

Aus dem Induktionsprinzip für Kalküle (Lemma 4.7) folgt direkt:

Lemma 4.11

Ein Kalkül $\mathfrak{K}$ über M_S ist korrekt, falls jede Sequenzenregel in $\mathfrak{K}$ korrekt ist.

Wir werden nun eine Reihe von korrekten Sequenzenregeln zusammentragen, die alle zusammen dann den von uns gesuchten korrekten, vollständigen und effektiven Kalkül über M_S bilden werden.

Grundregeln:

Für alle $\Gamma, \Gamma' \subseteq_e \text{FO}[\sigma]$ und alle $\varphi \in \text{FO}[\sigma]$ betrachten wir die folgenden Sequenzenregeln:

- Voraussetzungsregel (V):

$$\frac{}{\Gamma, \varphi \vdash \varphi}$$

- Erweiterungsregel (E):

$$\frac{\Gamma \vdash \varphi}{\Gamma' \vdash \varphi} \qquad \text{falls } \Gamma \subseteq \Gamma'$$

Lemma 4.12

Jede der Grundregeln (V) *bzw.* (E) *ist korrekt.*

Ausagenlogische Regeln:

Für alle $\Gamma \subseteq_e \mathsf{FO}[\sigma]$ und alle $\varphi, \psi, \chi \in \mathsf{FO}[\sigma]$ betrachten wir die folgenden Sequenzenregeln:

- Fallunterscheidungsregel (FU):

$$\frac{\begin{array}{ll} \Gamma, \psi & \vdash \varphi \\ \Gamma, \neg\psi & \vdash \varphi \end{array}}{\begin{array}{ll} \Gamma & \vdash \varphi \end{array}}$$

- Widerspruchsregel (W):

$$\frac{\begin{array}{l} \Gamma \vdash \psi \\ \Gamma \vdash \neg\psi \end{array}}{\Gamma \vdash \varphi} \qquad \text{(für alle } \varphi \in \mathsf{FO}[\sigma])$$

- $\wedge$-Einführung im Antezedens $(\wedge\mathsf{A}_1)$, $(\wedge\mathsf{A}_2)$:

$$\frac{\Gamma, \varphi \vdash \chi}{\Gamma, (\varphi \wedge \psi) \vdash \chi} \qquad \frac{\Gamma, \psi \vdash \chi}{\Gamma, (\varphi \wedge \psi) \vdash \chi}$$

- $\wedge$-Einführung im Sukzedens $(\wedge\mathsf{S})$:

$$\frac{\begin{array}{l}\Gamma \vdash \varphi \\ \Gamma \vdash \psi\end{array}}{\Gamma \vdash (\varphi \wedge \psi)}$$

- $\vee$-Einführung im Antezedens $(\vee\mathsf{A})$:

$$\frac{\begin{array}{l}\Gamma, \varphi \vdash \chi \\ \Gamma, \psi \vdash \chi\end{array}}{\Gamma, (\varphi \vee \psi) \vdash \chi}$$

- $\vee$-Einführung im Sukzedens $(\vee\mathsf{S}_1)$, $(\vee\mathsf{S}_2)$:

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash (\varphi \vee \psi)} \qquad \frac{\Gamma \vdash \psi}{\Gamma \vdash (\varphi \vee \psi)}$$

Lemma 4.13

Jede der aussagenlogischen Regeln (FU), (W), ($\wedge$A$_1$), ($\wedge$A$_2$), ($\wedge$S), ($\vee$A), ($\vee$S$_1$), ($\vee$S$_2$) *ist korrekt.*

Substitutionen

Um weitere wichtige Sequenzenregeln einführen zu können, benötigen wir eine Möglichkeit, für eine Variable $x \in \mathrm{VAR}$ und einen σ-Term $t \in \mathrm{T}_\sigma$ eine FO[σ]-Formel φ so zu einer FO[σ]-Formel $\varphi\frac{t}{x}$ abzuändern, dass gilt:

> *Die Formel $\varphi\frac{t}{x}$ sagt über den Term t dasselbe aus, wie die Formel φ über die Variable x.*

Präzise: Es soll für jede σ-Interpretation $\mathcal{I}$ gelten:

$$\mathcal{I} \models \varphi\tfrac{t}{x} \quad \iff \quad \mathcal{I}\tfrac{t}{x} \models \varphi. \tag{2}$$

Dabei ist die σ-Interpretation $\mathcal{I}\frac{t}{x}$ für $\mathcal{I} = (\mathcal{A}, \beta)$ wie folgt definiert:
$\mathcal{I}\frac{t}{x} := (\mathcal{A}, \beta\frac{a}{x})$, für $a := [\![t]\!]^{\mathcal{I}}$.

Außerdem soll gelten:

$$\varphi\tfrac{x}{x} \quad = \quad \varphi. \tag{3}$$

Um zu gewährleisten, dass (2) und (3) gilt, wählen wir zu gegebenem φ, t und x die Formel $\varphi\frac{t}{x}$ wie folgt:

- Falls $t = x$, so setze $\varphi\frac{t}{x} := \varphi$. Andernfalls gehe wie folgt vor:
- Sei $y_1, \ldots, y_\ell$ eine Liste aller Variablen aus $\text{var}(t) \cup \{x\}$, die gebundene Vorkommen in φ besitzen.
- Sei $z_1, \ldots, z_\ell$ eine Liste von Variablen $\neq x$, die *nicht* in φ oder t vorkommen.
- Sei φ' die Formel, die aus φ entsteht, indem für jedes $i \in \{1, \ldots, \ell\}$ jedes *gebundene* Vorkommen der Variablen y_i ersetzt wird durch die Variable z_i.
- Sei $\varphi\frac{t}{x}$ die Formel, die aus φ' entsteht, indem jedes Vorkommen der Variablen x durch den Term t ersetzt wird.

Lemma 4.14 (Substitutionslemma)

Für jede FO[σ]*-Formel* φ*, jeden* σ*-Term* t*, jede Variable* $x \in$ VAR *und jede* σ*-Interpretation* $\mathcal{I}$ *gilt:*

$$\mathcal{I} \models \varphi\frac{t}{x} \qquad \iff \qquad \mathcal{I}\frac{t}{x} \models \varphi.$$

Beweis.

Übung. $\square$

Wir können nun weitere wichtige Sequenzenregeln formulieren:

Quantorenregeln:

Für alle $\Gamma \subseteq_e \text{FO}[\sigma]$, alle $\varphi, \psi \in \text{FO}[\sigma]$, alle $x, y \in \text{VAR}$ und alle $t \in \text{T}_\sigma$ betrachten wir die folgenden Sequenzenregeln:

- $\forall$-Einführung im Antezedens ($\forall$A):

$$\frac{\Gamma, \varphi\frac{t}{x} \ \vdash \psi}{\Gamma, \forall x \varphi \vdash \psi}$$

- $\forall$-Einführung im Sukzedens ($\forall$S):

$$\frac{\Gamma \vdash \ \varphi\frac{y}{x}}{\Gamma \vdash \forall x \varphi} \qquad \text{falls} \quad y \notin \text{frei}(\Gamma, \forall x \varphi)$$

- $\exists$-Einführung im Antezedens ($\exists$A):

$$\frac{\Gamma, \varphi\frac{y}{x} \ \vdash \psi}{\Gamma, \exists x \varphi \vdash \psi} \qquad \text{falls} \quad y \notin \text{frei}(\Gamma,\ \exists x \varphi,\ \psi)$$

- $\exists$-Einführung im Sukzedens ($\exists$S):

$$\frac{\Gamma \vdash \ \varphi\frac{t}{x}}{\Gamma \vdash \exists x \varphi}$$

Lemma 4.15

Jede der Quantorenregeln $(\forall\text{A})$, $(\forall\text{S})$, $(\exists\text{A})$, $(\exists\text{S})$ *ist korrekt.*

Gleichheitsregeln:

Für alle $\Gamma \subseteq_e \mathrm{FO}[\sigma]$, alle $\varphi \in \mathrm{FO}[\sigma]$, alle $x \in \mathrm{VAR}$ und alle $t, u \in \mathrm{T}_\sigma$ betrachten wir die folgenden Sequenzenregeln:

- Reflexivität der Gleichheit (G):

$$\frac{}{\Gamma \vdash t{=}t}$$

- Substitutionsregel (S):

$$\frac{\Gamma \qquad\quad \vdash \varphi\frac{t}{x}}{\Gamma, t{=}u \vdash \varphi\frac{u}{x}}$$

Lemma 4.16

Jede der Gleichheitsregeln (G) *bzw.* (S) *ist korrekt.*

Der Sequenzenkalkül $\mathfrak{K}_S$ für die Logik erster Stufe

Definition 4.17

Der Sequenzenkalkül $\mathfrak{K}_S$ ist der Kalkül über der Menge M_S aller Sequenzen, der für alle $\Gamma, \Gamma' \subseteq_e \mathrm{FO}[\sigma]$, alle $\varphi, \psi, \chi \in \mathrm{FO}[\sigma]$, alle $t, u \in \mathrm{T}_\sigma$ und alle $x, y \in \mathrm{VAR}$ aus

- den Grundregeln (V), (E),
- den aussagenlogischen Regeln

  (FU), (W), ($\wedge$A$_1$), ($\wedge$A$_2$), ($\wedge$S), ($\vee$A), ($\vee$S$_1$), ($\vee$S$_2$),
- den Quantorenregeln ($\forall$A), ($\forall$S), ($\exists$A), ($\exists$S)
- und den Gleichheitsregeln (G), (S)

besteht.

Aus der Korrektheit der Regeln des Sequenzenkalküls (Lemmas 4.12, 4.13, 4.15, 4.16) folgt mit Lemma 4.11:

Satz 4.18

Der Sequenzenkalkül $\mathfrak{K}_S$ ist korrekt,
d.h. jede in $\mathfrak{K}_S$ ableitbare Sequenz ist korrekt.

Außerdem sieht man anhand der Definition der einzelnen Regeln leicht, dass es einen Algorithmus gibt, der bei Eingabe einer Zahl $\ell \geqslant 1$ und einer Folge $(a_1, \ldots, a_\ell) \in M_S^\ell$ entscheidet, ob $(a_1, \ldots, a_\ell)$ eine Ableitung in $\mathfrak{K}_S$ ist.

Für *abzählbare* Signaturen σ kann man außerdem einen Algorithmus angeben, der nach und nach alle Folgen in $\{(a_1, \ldots, a_\ell) \in M_S^\ell \ : \ \ell \geqslant 1\}$ ausgibt.

Beides zusammen liefert für abzählbare Signaturen σ, dass der Sequenzenkalkül $\mathfrak{K}_S$ effektiv ist.
Details: Übung.

Unser nächstes Ziel ist, zu zeigen, dass der Sequenzenkalkül $\mathfrak{K}_S$ auch vollständig ist, d.h. dass es für jede *korrekte* Sequenz eine Ableitung in $\mathfrak{K}_S$ gibt.

Dazu betrachten wir zunächst einige Beispiele für Ableitungen im Sequenzenkalkül $\mathfrak{K}_S$.

Darstellung von Ableitungen

Am Anfang des Kapitels haben wir bereits vereinbart, dass wir ähnlich wie bei Resolutionsableitungen auch allgemein für einen Kalkül $\mathfrak{K}$ über einer Menge M Ableitungen $(a_1, \ldots, a_\ell)$ der besseren Lesbarkeit halber oft zeilenweise schreiben, also

$$\begin{array}{ll} (1) & a_1 \\ (2) & a_2 \\ \vdots & \\ (\ell) & a_\ell \end{array}$$

und am Ende jeder Zeile eine kurze Begründung angeben.

Im Folgenden betrachten wir einige Beispiele für Ableitungen im Sequenzenkalkül $\mathfrak{K}_S$.

Beispiele 4.19

(a) Für jedes $\Gamma \subseteq_e \mathrm{FO}[\sigma]$ und jedes $\varphi \in \mathrm{FO}[\sigma]$ ist die Sequenz $\Gamma \vdash (\varphi \vee \neg\varphi)$ ableitbar in $\mathfrak{K}_S$:

Beweisbarkeit: $\Phi \vdash_{\mathfrak{K}_S} \varphi$

Definition 4.20

Sei $\Phi \subseteq \mathsf{FO}[\sigma]$ und sei $\varphi \in \mathsf{FO}[\sigma]$.
Die Formel φ heißt beweisbar aus Φ (kurz: $\Phi \vdash_{\mathfrak{K}_S} \varphi$), wenn es ein $\Gamma \subseteq_e \Phi$ gibt, so dass die Sequenz $\Gamma \vdash \varphi$ in $\mathfrak{K}_S$ ableitbar ist.

Ein Beweis von φ aus Φ ist eine Ableitung einer Sequenz $\Gamma \vdash \varphi$ in $\mathfrak{K}_S$, wobei $\Gamma \subseteq_e \Phi$ ist.

Notation

An Stelle von $\emptyset \vdash_{\mathfrak{K}_S} \varphi$ schreiben wir auch kurz: $\vdash_{\mathfrak{K}_S} \varphi$.

Aus der Korrektheit des Sequenzenkalküls $\mathfrak{K}_S$ (Satz 4.18) folgt:

Korollar 4.21

Für jede $\mathsf{FO}[\sigma]$*-Formel* φ *und für jede Formelmenge* $\Phi \subseteq \mathsf{FO}[\sigma]$ *gilt:*

$$\Phi \vdash_{\mathfrak{K}_S} \varphi \quad \Longrightarrow \quad \Phi \models \varphi.$$

Widerspruchsfreiheit

In der Mathematik nennen wir eine Menge von Aussagen widerspruchsvoll, falls sich daraus ein Widerspruch (d.h. eine bestimmte Aussage und deren Negat) herleiten lässt.
Wenn wir unter „herleiten" einen Beweis im Sequenzenkalkül $\mathfrak{K}_S$ verstehen, ergibt sich folgender Begriff:

Definition 4.22

Sei $\Phi \subseteq \text{FO}[\sigma]$.

(a) Φ heißt widerspruchsvoll, falls es eine $\text{FO}[\sigma]$-Formel φ gibt, so dass $\Phi \vdash_{\mathfrak{K}_S} \varphi$ und $\Phi \vdash_{\mathfrak{K}_S} \neg\varphi$.

(b) Φ heißt widerspruchsfrei, falls Φ nicht widerspruchsvoll ist.

Aus der Korrektheit des Sequenzenkalküls folgt, dass erfüllbare Formelmengen widerspruchsfrei sind:

Korollar 4.23

Für alle $\Phi \subseteq \text{FO}[\sigma]$ *gilt:* $\quad \Phi$ *erfüllbar* $\Longrightarrow$ Φ *widerspruchsfrei.*

Eigenschaften widerspruchsvoller Mengen

Lemma 4.24

Für jede Formelmenge $\Phi \subseteq \mathsf{FO}[\sigma]$ *sind folgende Aussagen äquivalent:*

(a) Φ *ist widerspruchsvoll.*

(b) *Für jede* $\mathsf{FO}[\sigma]$*-Formel* ψ *gilt:* $\Phi \vdash_{\mathfrak{K}_S} \psi$.

Der Vollständigkeitssatz

Satz 4.25

Für alle Signaturen σ, alle Formelmengen $\Phi \subseteq \mathrm{FO}[\sigma]$ und alle Formeln $\varphi \in \mathrm{FO}[\sigma]$ gilt:

(1) $\Phi \vdash_{\mathfrak{K}_S} \varphi \iff \Phi \models \varphi$.

(2) Φ *ist widerspruchsfrei* $\iff$ Φ *ist erfüllbar.*

Die Richtung „$\Longrightarrow$“ von (1) und die Richung „$\Longleftarrow$“ von (2) haben wir bereits in Korollar 4.21 und Korollar 4.23 bewiesen.

Die Richtung „$\Longrightarrow$“ von (2) wird von dem folgenden, schwer zu beweisenden Erfüllbarkeitslemma bereitgestellt:

Lemma 4.26 (Erfüllbarkeitslemma)

Jede widerspruchsfreie Menge $\Phi \subseteq \mathrm{FO}[\sigma]$ ist erfüllbar.

Beweis des Vollständigkeitssatzes unter Verwendung des Erfüllbarkeitslemmas:

Unter Verwendung des Erfüllbarkeitslemmas (Lemma 4.26) erhalten wir zusammen mit Korollar 4.23, dass Teil (2) des Vollständigkeitssatzes korrekt ist. D.h. für jede Formelmenge $\Phi \subseteq \mathrm{FO}[\sigma]$ gilt:

(2) Φ *ist widerspruchsfrei* $\iff$ Φ *ist erfüllbar.*

Die Richtung „$\Longrightarrow$“ von (1) haben wir bereits in Korollar 4.21 gezeigt.

Die Richtung „$\Longleftarrow$“ von Teil (1) des Vollständigkeitssatzes lässt sich wie folgt beweisen:

Zum Beweis des Erfüllbarkeitslemmas:

Zur Erinnerung: Das Erfüllbarkeitslemma besagt:

Jede widerspruchsfreie Formelmenge $\Phi \subseteq \mathrm{FO}[\sigma]$ *ist erfüllbar.*

Beweisidee:

Konstruiere eine σ-Interpretation $\mathcal{I}_\Phi = (\mathcal{A}, \beta)$, so dass gilt:

- Das Universum A von $\mathcal{A}$ ist die Menge T_σ aller σ-Terme.
- Für jeden σ-Term t gilt: $[\![t]\!]^{\mathcal{I}} = t$.
- Für jedes Relationssymbol $R \in \sigma$, für $k := \mathrm{ar}(R)$, und für alle σ-Terme $t_1, \ldots, t_k$ gilt:
$$(t_1, \ldots, t_k) \in R^{\mathcal{A}} \iff \Phi \vdash_{\mathfrak{K}_S} R(t_1, \ldots, t_k)$$

Diese Interpretation $\mathcal{I}_\Phi$ wird Terminterpretation von Φ genannt.
Gemäß Definition erfüllt $\mathcal{I}_\Phi$ alle atomaren Formeln der Form $R(t_1, \ldots, t_k)$ in Φ.
Im Allgemeinen gilt jedoch noch nicht $\mathcal{I}_\Phi \models \Phi$ (betrachte dazu beispielsweise die Formelmenge $\Phi := \{v_0{=}v_1\}$, die offensichtlicherweise erfüllbar ist, für die aber gilt: $\mathcal{I}_\Phi \not\models \Phi$).

Aber nach einigen anspruchsvollen Modifikationen von $\mathcal{I}_\Phi$ erhält man eine Interpretation $\mathcal{I}'_\Phi$ mit $\mathcal{I}'_\Phi \models \Phi$.

Abschnitt 4.3:

Der Endlichkeitssatz

Zur Erinnerung:

Wir haben bereits den Endlichkeitssatz der Aussagenlogik kennen gelernt, der besagt, dass Folgendes für jede Menge $\Phi \subseteq \mathsf{AL}$ und jede Formel $\psi \in \mathsf{AL}$ gilt:

(1) Φ ist erfüllbar $\iff$ Jede endliche Teilmenge von Φ ist erfüllbar.

(2) $\Phi \models \psi \iff$ Es gibt eine endliche Teilmenge Γ von Φ, so dass $\Gamma \models \psi$.

Der Endlichkeitssatz gilt auch für die Logik erster Stufe, d.h. die Aussagen (1) und (2) gelten auch für alle Mengen $\Phi \subseteq \mathsf{FO}[\sigma]$ und alle $\psi \in \mathsf{FO}[\sigma]$.

Zum Beweis der Endlichkeitssatzes der Logik erster Stufe nutzen wir den Vollständigkeitssatz sowie das folgende Lemma.

Das syntaktische Endlichkeitslemma

Lemma 4.27

Für jede Signatur σ und jede Formelmenge $\Phi \subseteq \mathsf{FO}[\sigma]$ gilt:

Φ *ist widerspruchsfrei* $\iff$ *Jede endliche Teilmenge von* Φ *ist widerspruchsfrei.*

Der Endlichkeitssatz (auch bekannt als Kompaktheitssatz)

Satz 4.28

Für jede Signatur σ, jede Formelmenge $\Phi \subseteq \mathsf{FO}[\sigma]$ und jede Formel $\psi \in \mathsf{FO}[\sigma]$ gilt:

(1) Φ *ist erfüllbar* $\iff$ *Jede endliche Teilmenge von* Φ *ist erfüllbar.*

(2) $\Phi \models \psi \iff$ *Es gibt eine endliche Teilmenge* Γ *von* Φ*, so dass* $\Gamma \models \psi$.

Beachte: Die Aussage des Endlichkeitssatzes ist nur für unendliche Formelmengen Φ interessant (für endliche Mengen Φ ist sie trivial).

Erststufige Axiomatisierbarkeit

Definition 4.29

Eine Klasse $\mathfrak{C}$ von σ-Strukturen heißt erststufig axiomatisierbar, falls es eine Menge Φ von FO[σ]-Sätzen gibt, so dass gilt: $\mathfrak{C} = \text{MOD}_\sigma(\Phi)$.

Zur Erinnerung:
$\text{MOD}_\sigma(\Phi)$ ist die Klasse aller σ-Strukturen $\mathcal{A}$, für die gilt: $\mathcal{A} \models \Phi$.

Definition 4.30

Die Mächtigkeit einer σ-Struktur ist die Mächtigkeit ihres Universums.
Eine σ-Struktur heißt endlich, unendlich, abzählbar, bzw. überabzählbar, wenn ihr Universum die entsprechende Mächtigkeit besitzt.

Beispiel 4.31

Die Klasse aller unendlichen σ-Strukturen ist erststufig axiomatisierbar.

Wir können den Endlichkeitssatz anwenden, um zu zeigen, dass bestimmte Klassen von Strukturen nicht erststufig axiomatisierbar sind.
Im Folgenden betrachten wir dazu zwei Beispiele: die Nicht-Axiomatierbarkeit der „Endlichkeit“ von Strukturen und die Nicht-Axiomatisierbarkeit von „Graph-Zusammenhang“.

Nicht-Axiomatisierbarkeit der „Endlichkeit" von Strukturen

Lemma 4.32

Sei Φ eine Menge von FO[σ]*-Sätzen. Falls Φ beliebig große endliche Modelle besitzt (d.h. für jedes $n \in \mathbb{N}$ gibt es eine endliche σ-Struktur $\mathcal{A}$ mit $|A| \geqslant n$ und $\mathcal{A} \models \Phi$), so besitzt Φ ein unendliches Modell.*

Satz 4.33

Die Klasse aller <u>endlichen</u> σ-Strukturen ist <u>nicht</u> erststufig axiomatisierbar.

Korollar 4.34

Es gibt keine <u>endliche</u> Menge von FO[σ]*-Sätzen, die die Klasse aller <u>unendlichen</u> σ-Strukturen erststufig axiomatisiert.*

Nicht-Axiomatisierbarkeit von „Graph-Zusammenhang“

Satz 4.35

Die Klasse aller zusammenhängenden Graphen ist nicht erststufig axiomatisierbar.

Der Satz von Löwenheim und Skolem

Unter Verwendung von Teilergebnissen, die beim (in dieser Vorlesung nicht im Detail behandelten) Beweis des Erfüllbarkeitslemmas anfallen, erhält man das folgende Resultat.

Satz 4.36 (Der Satz von Löwenheim und Skolem)

Sei σ eine abzählbare Signatur. Dann hat jede erfüllbare Formelmenge $\Phi \subseteq \mathsf{FO}[\sigma]$ ein höchstens abzählbares Modell.

(Hier ohne Beweis)

Als direkte Folgerung aus dem Satz von Löwenheim und Skolem erhalten wir:

Korollar 4.37

Sei σ eine abzählbare Signatur. Dann ist die Klasse aller <u>überabzählbaren</u> σ-Strukturen <u>nicht</u> erststufig axiomatisierbar.

Abschnitt 4.4:

Die Grenzen der Berechenbarkeit

Zur Erinnerung:

Einige Begriffe zum Thema (Un)Entscheidbarkeit

Entscheidungsprobleme sind Probleme, die mit „ja“ oder „nein“ beantwortet werden können. Genauer:

- Sei M eine abzählbar unendliche Menge, zum Beispiel
  - die Menge Σ^* aller Worte über einem endlichen Alphabet Σ, oder
  - die Menge aller Graphen, deren Knotenmenge eine endliche Teilmenge der natürlichen Zahlen ist.

- Das Entscheidungsproblem für eine Menge $L \subseteq M$ ist das folgende Berechnungsproblem:

Das Entscheidungsproblem für $L \subseteq M$

Eingabe: Ein Element $m \in M$.

Frage: Ist $m \in L$?

Beispiele für Entscheidungsprobleme

- Graphzusammenhang ist das Entscheidungsproblem für $L \subseteq M$, wobei
  - M die Menge aller ungerichteten Graphen ist, deren Knotenmenge eine endliche Teilmenge von $\mathbb{N}$ ist und
  - L die Menge aller zusammenhängenden Graphen aus M ist.

- Das Halteproblem ist das Entscheidungsproblem für $L \subseteq M$, wobei
  - M die Menge aller Worte $w\#x$ mit $w, x \in \{0,1\}^*$ ist und
  - L die Menge aller Worte $w\#x$ ist, so dass w eine deterministische Turingmaschine beschreibt, die bei Eingabe x nach endlich vielen Schritten anhält.

Entscheidungsprobleme für die Logik erster Stufe

Allgemeingültigkeitsproblem für FO[σ]

Eingabe: Eine FO[σ]-Formel φ

Frage: Ist φ allgemeingültig?

Formal:

M ist die Menge aller Worte über dem Alphabet $A_{\mathsf{FO}[\sigma]}$ und

L ist die Menge $\{\varphi \in \mathsf{FO}[\sigma] \ : \ \varphi \text{ ist allgemeingültig}\}$

Erfüllbarkeitsproblem für FO[σ]

Eingabe: FO[σ]-Formel φ

Frage: Ist φ erfüllbar?

Unerfüllbarkeitsproblem für FO[σ]

Eingabe: FO[σ]-Formel φ

Frage: Ist φ unerfüllbar?

Folgerungsproblem für FO[σ]

Eingabe: Zwei FO[σ]-Formeln φ, ψ

Frage: Gilt $\varphi \models \psi$?

Entscheidbarkeit und Semi-Entscheidbarkeit

Definition 4.38

Sei M eine abzählbar unendliche Menge.

(a) Eine Menge $L \subseteq M$ heißt entscheidbar, falls es einen Algorithmus gibt, der bei Eingabe eines $m \in M$ nach endlich vielen Schritten anhält und
   - „ja“ ausgibt, falls $m \in L$
   - „nein“ ausgibt, falls $m \notin L$.

(b) $L \subseteq M$ heißt semi-entscheidbar, falls es einen Algorithmus gibt, der bei Eingabe eines $m \in M$
   - nach endlich vielen Schritten anhält und „ja“ ausgibt, falls $m \in L$
   - nie anhält, falls $m \notin L$.

Beispiele:

- Graphzusammenhang ist entscheidbar (z.B. durch Tiefen- oder Breitensuche).
- Das Halteproblem ist semi-entscheidbar (bei Eingabe von $w\#x$ konstruiere die von w repräsentierte deterministische Turingmaschine und lasse diese mit Eingabe x laufen).
  Ist es auch entscheidbar? Nein! — Das Halteproblem ist das Paradebeispiel eines nicht entscheidbaren Problems.

Einfache Beobachtungen

- Jede entscheidbare Menge $L \subseteq M$ ist auch semi-entscheidbar (anstatt „nein“ auszugeben und anzuhalten, gehen wir einfach in eine Endlosschleife)

- Für jede entscheidbare Menge $L \subseteq M$ ist auch die Menge $\overline{L} := (M \setminus L) \subseteq M$ entscheidbar (vertausche einfach die Antworten „ja“ und „nein“)

- Wenn sowohl $L \subseteq M$ als auch $\overline{L} := (M \setminus L) \subseteq M$ semi-entscheidbar sind, dann ist $L \subseteq M$ sogar entscheidbar.

Semi-Entscheidbarkeit einiger Logik-Probleme

Satz 4.39

Sei σ eine höchstens abzählbare Signatur.
Jedes der folgenden Probleme ist semi-enscheidbar:

(a) *das Allgemeingültigkeitsproblem für* $\mathsf{FO}[\sigma]$,

(b) *das Unerfüllbarkeitsproblem für* $\mathsf{FO}[\sigma]$,

(c) *das Folgerungsproblem für* $\mathsf{FO}[\sigma]$.

Unentscheidbarkeit einiger Logik-Probleme

Unser nächstes Ziel ist, zu zeigen, dass für bestimmte Signaturen σ gilt:

- Das Allgemeingültigkeitsproblem für FO[σ],
- das Unerfüllbarkeitsproblem für FO[σ],
- das Erfüllbarkeitsproblem für FO[σ] und
- das Folgerungsproblem für FO[σ]

ist nicht entscheidbar.

Wir werden dazu wie folgt vorgehen:

1. Wir nutzen das bekannte Resultat, das besagt, dass das Postsche Korrespondenzproblem unentscheidbar ist.
2. Wir zeigen, wie das Postsche Korrespondenzproblem unter Zuhilfenahme eines Entscheidungs-Algorithmus für das Allgemeingültigkeitsproblem für FO[σ] (für eine geeignete Signatur σ) gelöst werden könnte.
   Dadurch erhalten wir, dass das Allgemeingültigkeitsproblem für FO[σ] unentscheidbar ist.
3. Die Unentscheidbarkeit des Unerfüllbarkeitsproblems, des Erfüllbarkeitsproblems und des Folgerungsproblems für FO[σ] folgen dann leicht aus der Unentscheidbarkeit des Allgemeingültigkeitsproblems für FO[σ].

Das Postsche Korrespondenzproblem

Das Postsche Korrespondenzproblem (PKP)

Eingabe: Eine Zahl $k \geqslant 1$ und k Paare (x_1, y_1), (x_2, y_2), ..., (x_k, y_k) mit $x_1, y_1, \ldots, x_k, y_k \in \{0,1\}^*$.

Frage: Gibt es ein $n \geqslant 1$ und Indizes $i_1, \ldots, i_n \in \{1, \ldots, k\}$, so dass gilt: $x_{i_1} x_{i_2} \cdots x_{i_n} \ = \ y_{i_1} y_{i_2} \cdots y_{i_n}$?

Beispiel:

Das PKP mit Eingabe $k = 3$ und

$$(x_1, y_1) \ = \ (\,1,\ 111\,), \qquad (x_2, y_2) \ = \ (\,10111,\ 10\,), \qquad (x_3, y_3) \ = \ (\,10,\ 0\,).$$

hat eine Lösung mit $n = 4$ und $i_1 = 2$, $i_2 = 1$, $i_3 = 1$, $i_4 = 3$, denn:

$$\begin{aligned} x_2\, x_1\, x_1\, x_3 \quad &= \quad 10111\ \ 1\ \ 1\ \ 10 \\ y_2\, y_1\, y_1\, y_3 \quad &= \quad 10\ \ 111\ \ 111\ \ 0. \end{aligned}$$

Bekannt:

- Das PKP ist semi-entscheidbar. (Dies sieht man leicht.)
- Das PKP ist <u>nicht</u> entscheidbar.
  (Dies wurde in der „Einführung in die Theoretische Informatik“ bewiesen.)

Die Unentscheidbarkeit der Logik erster Stufe

Satz 4.40

Sei $\sigma := \{R, f_0, f_1, c\}$, *wobei* c *ein Konstantensymbol,* R *ein 2-stelliges Relationssymbol und* f_0, f_1 *zwei 1-stellige Funktionssymbole sind.*

Das Allgemeingültigkeitsproblem für $\mathrm{FO}[\sigma]$ *ist nicht entscheidbar.*

Beweis: Auf Grund der Unentscheidbarkeit des PKP reicht es, eine Reduktion vom PKP zum Allgemeingültigkeitsproblem für $\mathrm{FO}[\sigma]$ anzugeben. D.h. wir zeigen, dass bei Eingabe eines Tupels $I = \big(k,\ (x_1, y_1),\ \ldots,\ (x_k, y_k) \big)$, das eine Eingabe für's PKP repräsentiert, eine $\mathrm{FO}[\sigma]$-Formel φ_I konstruiert werden kann, die genau dann allgemeingültig ist, wenn I eine „ja"-Instanz für's PKP ist (d.h. es gibt $n \geqslant 1$ und $i_1, \ldots, i_n \in [k]$, so dass $x_{i_1} \cdots x_{i_n} = y_{i_1} \cdots y_{i_n}$).

Wenn das Allgemeingültigkeitsproblem für $\mathrm{FO}[\sigma]$ entscheidbar wäre, wäre daher auch das PKP entscheidbar.

Zur Konstruktion der Formel φ_I gehen wir in mehreren Schritten vor.

Schritt 1: Für jede Eingabe $I = \big(k,\ (x_1, y_1),\ \ldots,\ (x_k, y_k) \big)$ für das PKP definiere eine σ-Struktur $\mathcal{A}_I$, so dass gilt:

$\mathcal{A}_I \models \exists z\, R(z,z) \iff$ I ist eine „ja"-Instanz für's PKP, d.h. es gibt $n \geqslant 1$ und $i_1, \ldots, i_n \in [k]$, so dass $x_{i_1} \cdots x_{i_n} = y_{i_1} \cdots y_{i_n}$.

Dazu wählen wir $\mathcal{A}_I$ wie folgt:

Schritt 2: Konstruiere FO[σ]-Formeln ψ_I^{Start} und $\psi_I^{Schritt}$, die $\mathcal{A}_I$ hinreichend genau beschreiben.

Die Formel ψ_I^{Start} soll besagen, dass die Relation $R^{\mathcal{A}_I}$ die Tupel (x_j, y_j) für alle $j \in [k]$ enthält.

Die Formel $\psi_I^{Schritt}$ soll besagen, dass die Relation $R^{\mathcal{A}_I}$ abgeschlossen ist unter Konkatenation mit (x_j, y_j); d.h.: Ist $(u, v) \in R^{\mathcal{A}_I}$ und $j \in [k]$, so ist auch $(ux_j, vy_j) \in R^{\mathcal{A}_I}$.

Um dies durch FO[σ]-Formeln zu formulieren, nutzen wir folgende Schreibweisen:

Schritt 3: Setze $\varphi_I \ := \ \Big(\big(\psi_I^{Start} \wedge \psi_I^{Schritt}\big) \to \ \exists z\, R(z,z) \Big)$

Klar: Es gibt einen Algorithmus, der bei Eingabe von I die Formel φ_I konstruiert.

Behauptung 1:

φ_I ist allgemeingültig $\iff$ I ist eine „ja"-Instanz für's PKP.

Behauptung 2: Für alle $(u,v) \in R^{\mathcal{A}_I}$ gilt: $\big(h(u), h(v)\big) \in R^{\mathcal{B}}$.

Aus Satz 4.39, Satz 4.40 und den bekannten Zusammenhängen zwischen semi-entscheidbaren und entscheidbaren Problemen, sowie den Korrespondenzen zwischen Allgemeingültigkeit, (Un)Erfüllbarkeit und logischer Folgerung, erhält man leicht:

Korollar 4.41

Sei σ die Signatur aus Satz 4.40. Dann gilt:

(a) *Das Allgemeingültigkeitsproblem für* $\text{FO}[\sigma]$ *ist semi-entscheidbar aber nicht entscheidbar.*

(b) *Das Folgerungsproblem für* $\text{FO}[\sigma]$ *ist semi-entscheidbar aber nicht entscheidbar.*

(c) *Das Unerfüllbarkeitsproblem für* $\text{FO}[\sigma]$ *ist semi-entscheidbar aber nicht entscheidbar.*

(d) *Das Erfüllbarkeitsproblem für* $\text{FO}[\sigma]$ *ist nicht semi-entscheidbar.*

Beweis: Übung.

Bemerkung 4.42

Man kann zeigen, dass

(1) Korollar 4.41 für jede Signatur σ gilt, die mindestens ein Relationssymbol der Stelligkeit $\geqslant 2$ enthält

(2) für Signaturen σ, die ausschließlich aus Konstantensymbolen und Relationssymbolen der Stelligkeit 1 bestehen, jedes der in Korollar 4.41 betrachteten Probleme entscheidbar ist.

(Hier ohne Beweis)

Abschnitt 4.5:

Der Satz von Herbrand

- Im letzten Abschnitt haben wir gesehen, dass es keinen Algorithmus gibt, der das Erfüllbarkeitsproblem und das Allgemeingültigkeitsproblem der Logik erster Stufe löst und stets terminiert.

- Trotzdem möchte man für verschiedene Anwendungsbereiche Verfahren haben, die das Erfüllbarkeits- oder das Allgemeingültigkeitsproblem der Logik erster Stufe „so gut wie möglich“ lösen.

- Einen Ansatz für die Entwicklung solcher, in der Praxis nutzbarer, Verfahren liefert die Herbrand-Theorie, die nach dem französischen Logiker Jacques Herbrand (1908–1931) benannt ist.

- Ziel dieses Abschnitts ist, den Satz von Herbrand vorzustellen, der das Allgemeingültigkeits- bzw. das Erfüllbarkeitsproblem der Logik erster Stufe auf das entsprechende Problem der Aussagenlogik zurückführt.

Notationen

- In diesem Abschnitt bezeichnet σ stets eine endliche oder abzählbare Signatur, die mindestens ein Konstantensymbol enthält.
- Die Menge aller quantorenfreien FO[σ]-Formeln bezeichnen wir mit QF_σ.
- Ein Grundterm über σ ist ein variablenfreier σ-Term, d.h., ein σ-Term, der keine Variable enthält.
  Die Menge aller Grundterme über σ bezeichnen wir mit GT_σ.

Beispiele:

(a) Sei $\sigma := \{ c,\ f/1,\ g/2,\ R/2 \}$.
Grundterme über σ sind dann z.B.

$$c, \quad f(c), \quad g(c,c), \quad f(f(c)), \quad f(g(c,c)), \quad g(c,f(c)), \quad g(f(c),c), \quad \dots$$

(b) Sei $\sigma := \{ c,\ R/2 \}$.
Dann ist c der einzige Grundterm über σ. D.h.

$$\mathrm{GT}_\sigma \;=\; \{ c \}.$$

Herbrandstrukturen

Definition 4.43

Sei σ eine Signatur, die mindestens ein Konstantensymbol enthält.
Eine σ-Herbrandstruktur ist eine σ-Struktur $\mathcal{A}$ mit folgenden Eigenschaften:

- Das Universum A von $\mathcal{A}$ ist genau die Menge GT_σ aller Grundterme über σ (d.h. aller variablenfreien σ-Terme).
- Für jedes Konstantensymbol $c \in \sigma$ ist $c^{\mathcal{A}} = c$.
- Für jedes Funktionssymbol $f \in \sigma$, für $k := \mathrm{ar}(f)$, und für alle variablenfreien σ-Terme $t_1, \ldots, t_k \in A$ ist

$$f^{\mathcal{A}}(t_1, \ldots, t_k) \quad = \quad f(t_1, \ldots, t_k).$$

Beachte: Alle σ-Herbrandstrukturen haben dasselbe Universum und dieselbe Interpretation der Konstanten- und Funktionssymbole.

Lediglich die Interpretation der Relationssymbole kann in σ-Herbrandstrukturen frei gewählt werden.

Zur Angabe einer konkreten σ-Herbrandstruktur $\mathcal{A}$ genügt es also, die Interpretation der Relationssymbole anzugeben, d.h. für jedes Relationssymbol $R \in \sigma$ die Relation $R^{\mathcal{A}}$ anzugeben.

Beispiel

Sei $\sigma := \{ c,\ R/2 \}$.

Frage: Wie sehen σ-Herbrandstrukturen aus?

Antwort: Für jede σ-Herbrandstruktur $\mathcal{A}$ gilt:

- Universum: $A = \{ c \}$
- $c^{\mathcal{A}} = c$
- $R^{\mathcal{A}} \subseteq \{c\}^2$, d.h.

$$R^{\mathcal{A}} = \emptyset \qquad \text{oder} \qquad R^{\mathcal{A}} = \{ (c,c) \}.$$

Somit gibt es genau 2 verschiedene σ-Herbrandstrukturen.

Bemerkung 4.44

Sei $\mathcal{A}$ eine σ-Herbrandstruktur.
Man sieht leicht, dass Folgendes gilt:

- Für jeden variablenfreien σ-Term t (d.h. für jedes $t \in \mathrm{GT}_\sigma = A$) gilt:

$$[\![t]\!]^{\mathcal{A}} \quad = \quad t\,.$$

- Für jede quantorenfreie FO[σ]-Formel ψ gilt:
  Ist $\mathrm{var}(\psi) \subseteq \{x_1, \ldots, x_n\}$ und sind $t_1, \ldots, t_n \in \mathrm{GT}_\sigma$, so gilt:

$$\mathcal{A} \models \psi[t_1, \ldots, t_n] \qquad \Longleftrightarrow \qquad \mathcal{A} \models \psi\frac{t_1, \ldots, t_n}{x_1, \ldots, x_n}$$

  Dabei ist $\psi\frac{t_1, \ldots, t_n}{x_1, \ldots, x_n}$ die Formel, die aus ψ entsteht, indem für jedes $i \in [n]$ jedes Vorkommen von x_i ersetzt wird durch den Grundterm t_i.

Herbrand-Modelle und gleichheitsfreie Formeln in Skolemform

Definition 4.45

(a) Ein Herbrand-Modell eines FO[σ]-Satzes φ ist eine σ-Herbrandstruktur, die φ erfüllt.

(b) Eine FO[σ]-Formel φ heißt gleichheitsfrei, falls das Symbol „$=$" nicht in φ vorkommt.

(c) Eine FO[σ]-Formel ist in Skolemform (auch: Skolem-Normalform), falls sie von der Form

$$\forall x_1 \cdots \forall x_n \ \psi$$

ist, wobei gilt: $n \geqslant 0$, $x_1, \ldots, x_n$ sind paarweise verschiedene Variablen, und ψ ist eine quantorenfreie FO[σ]-Formel.

Satz 4.46

Sei σ eine Signatur, die mindestens ein Konstantensymbol besitzt.
Für jeden gleichheitsfreien FO[σ]-*Satz φ in Skolemform gilt:*

$$\varphi \text{ ist erfüllbar} \quad \iff \quad \varphi \text{ besitzt ein Herbrand-Modell.}$$

Die Herbrand-Expansion eines Satzes in Skolemform

Definition 4.47

Sei φ ein gleichheitsfreier FO[σ]-Satz in Skolemform, d.h. φ ist von der Form $\forall x_1 \cdots \forall x_n\, \psi$, wobei ψ quantorenfrei und gleichheitsfrei ist.
Die Herbrand-Expansion von φ ist die Formelmenge

$$\text{HE}(\varphi) \quad := \quad \left\{ \ \psi\,\frac{t_1,\ldots,t_n}{x_1,\ldots,x_n} \ : \ t_1,\ldots,t_n \in \text{GT}_\sigma \ \right\}$$

D.h.: Jede Formel in HE(φ) entsteht, indem in der quantorenfreien Formel ψ jede Variable x_i ersetzt wird durch einen Grundterm t_i.

Beispiel 4.48

Sei $\sigma = \{\, c,\ f/1,\ g/2,\ R/3 \,\}$ und sei $\varphi \ := \ \forall x\, \forall y\, \forall z\ R\big(x, f(y), g(z,x)\big)$.
Dann gehören z.B. die folgenden Formeln zur Herbrand-Expansion HE(φ):

- $R(c, f(c), g(c,c))$ (dies erhält man, indem jede der Variablen x, y, z durch den Grundterm c ersetzt wird)
- $R(f(c), f(c), g(c, f(c)))$ (dies erhält man, indem x durch den Grundterm $f(c)$ und jede der Variablen y, z durch den Grundterm c ersetzt wird)
- $R(g(c,c), f(f(c)), g(c, g(c,c)))$ (dies erhält man, indem Variable x durch den Grundterm $g(c,c)$, Variable y durch den Grundterm $f(c)$ und Variable z durch den Grundterm c ersetzt wird)

Die aussagenlogische Version der Herbrand-Expansion

Für jeden gleichheitsfreien FO[σ]-Satz φ in Skolemform gilt:

Jede Formel $\xi \in \text{HE}(\varphi)$ ist quantorenfrei, gleichheitsfrei und variablenfrei, und jede atomare Subformel von ξ ist von der Form $R(t_1, \ldots, t_k)$, wobei $R \in \sigma$, $k = \text{ar}(R)$ und $t_1, \ldots, t_k \in \text{GT}_\sigma$.

Für jede solche atomare Formel stellen wir ein Aussagensymbol $X_{R(t_1,\ldots,t_k)} \in$ AS bereit.

Für jedes $\xi \in \text{HE}(\varphi)$ sei $\text{al}(\xi)$ die aussagenlogische Formel, die aus ξ entsteht, indem jede atomare Subformel der Form $R(t_1, \ldots, t_k)$ ersetzt wird durch das Aussagensymbol $X_{R(t_1,\ldots,t_k)}$.

Die aussagenlogische Version der Herbrand-Expansion von φ ist die Menge

$$\text{AHE}(\varphi) \quad := \quad \{ \, \text{al}(\xi) \ : \ \xi \in \text{HE}(\varphi) \, \}.$$

Der Satz von Herbrand

Satz 4.49 (Satz von Gödel-Herbrand-Skolem)

Sei σ eine Signatur, die mindestens ein Konstantensymbol enthält.
Für jeden gleichheitsfreien $\mathrm{FO}[\sigma]$*-Satz φ in Skolemform gilt:*
φ *ist erfüllbar* $\iff$ *die aussagenlogische Formelmenge* $\mathrm{AHE}(\varphi)$ *ist erfüllbar.*

In Verbindung mit dem Endlichkeitssatz der Aussagenlogik erhalten wir:

Satz 4.50 (Satz von Herbrand)

Sei σ eine Signatur, die mindestens ein Konstantensymbol enthält. Sei ψ eine gleichheitsfreie und quantorenfreie $\mathrm{FO}[\sigma]$*-Formel und sei* $\{x_1, \dots, x_n\} = \mathrm{frei}(\psi)$*.*
Dann gilt für die $\mathrm{FO}[\sigma]$*-Sätze* $\varphi := \forall x_1 \cdots \forall x_n\, \psi$ *und* $\varphi' := \exists x_1 \cdots \exists x_n\, \psi$ *:*

(a) φ *ist erfüllbar* $\iff$ *jede endliche Teilmenge von* $\mathrm{AHE}(\varphi)$ *ist erfüllbar.*

(b) φ *ist unerfüllbar* $\iff$ *es gibt eine endliche Teilmenge von* $\mathrm{AHE}(\varphi)$*, die unerfüllbar ist.*

(c) φ' *ist allgemeingültig* $\iff$ *es gibt eine Zahl $m \in \mathbb{N}$ und Grundterme $t_{i,1}, \dots, t_{i,n}$ für alle $i \in [m]$, so dass die folgende Formel allgemeingültig ist:*

$$\bigvee_{i=1}^{m} \psi \frac{t_{i,1}, \dots, t_{i,n}}{x_1, \dots, x_n}$$

Anwendung des Satzes von Herbrand

Um nachzuweisen, dass ein gleichheitsfreier FO[σ]-Satz φ in Skolemform unerfüllbar ist, kann man auf Grund des Satzes von Herbrand wie folgt vorgehen:

Für $i = 1, 2, 3, \ldots$ tue Folgendes:

(1) Sei ξ_i die i-te Formel in AHE(φ)
(2) Teste, ob die aussagenlogische Formel $(\xi_1 \wedge \cdots \wedge \xi_i)$ unerfüllbar ist.
(3) Falls ja, halte an mit Ausgabe „φ ist unerfüllbar“

Man sieht leicht, dass dies ein Semi-Entscheidungsverfahren ist, das eine gegebene Formel φ auf Unerfüllbarkeit testet.

Durch die Einschränkung auf gleichheitsfreie FO[σ]-Sätze in Skolemform scheint dieses Verfahren auf den ersten Blick nur sehr eingeschränkt anwendbar zu sein.

Im Folgenden zeigen wir jedoch, dass jede FO[σ]-Formel in eine zu ihr erfüllbarkeitsäquivalente Formel der richtigen Form transformiert werden kann.

Definition 4.51

Seien σ_1, σ_2 Signaturen und φ_i eine FO[σ_i]-Formel, für jedes $i \in \{1, 2\}$.
Die Formel φ_2 heißt erfüllbarkeitsäquivalent zu φ_1, falls gilt:

$$\varphi_2 \text{ ist erfüllbar} \quad \Longleftrightarrow \quad \varphi_1 \text{ ist erfüllbar.}$$

Satz 4.52 (Skolemisierung)

Zu jeder Signatur σ gibt es eine Signatur $\hat{\sigma}$, so dass jede FO[σ]-Formel φ in einen zu φ erfüllbarkeitsäquivalenten gleichheitsfreien FO[$\hat{\sigma}$]-Satz $\hat{\varphi}$ in Skolemform transformiert werden kann.

Beispiel 4.53

Die Formel $\forall x\, \exists y\, \forall z\, \exists u\ R(x, y, z, u)$ ist erfüllbarkeitsäquivalent zum folgenden gleichheitsfreien Satz in Skolemform:

$$\forall x\, \forall z\ R\big(x, f(x), z, g(x, z)\big)$$

Abschnitt 4.6:

Automatische Theorembeweiser

Einfaches Verfahren (ohne Unifikation)

Seien φ und ψ zwei FO[σ]-Formeln.

Ziel: Automatischer Beweis, dass $\varphi \models \psi$ gilt.

Dazu reicht es, zu zeigen, dass die Formel $(\varphi \wedge \neg\psi)$ unerfüllbar ist.

Verfahren:

1. Erzeuge einen zu $(\varphi \wedge \neg\psi)$ erfüllbarkeitsäquivalenten gleichheitsfreien FO[$\hat{\sigma}$]-Satz χ in Skolemform (über der erweiterten Signatur $\hat{\sigma}$). Nutze dazu das im Beweis von Satz 4.52 vorgestellte Verfahren.

2. Verwende das auf Folie 364 beschriebene Semi-Entscheidungsverfahren, um zu herauszufinden, ob χ unerfüllbar ist.

Beispiel 4.54

Sei $\sigma := \{R/1, c, f/1\}$,

$$\begin{aligned} \varphi \quad &:= \quad R(c) \ \wedge \ \forall x \, \exists y \, \big((\, R(x) \to R(f(f(y))) \,) \vee R(f(x)) \, \big) \\ \psi \quad &:= \quad \exists x \, R(f(f(x))). \end{aligned}$$

Dann ist $(\varphi \wedge \neg\psi) \ =$

$$R(c) \ \wedge \ \forall x \, \exists y \, \big((\, R(x) \to R(f(f(y))) \,) \vee R(f(x)) \, \big) \ \wedge \ \neg\exists x \, R(f(f(x)))$$

ein gleichheitsfreier Satz. Eine Umformung in Pränex-Normalform liefert den dazu äquivalenten Satz

$$\forall x \, \exists y \, \Big(\, R(c) \ \ \wedge \ \ (\, \neg R(x) \ \vee \ R(f(f(y))) \ \vee \ R(f(x)) \,) \ \ \wedge \ \ \neg R(f(f(x))) \, \Big).$$

Wir erweitern die Signatur um ein 1-stelliges Funktionssymbol g und erhalten den dazu erfüllbarkeitsäquivalenten gleichheitsfreien Satz in Skolemform $\chi \ =$

$$\forall x \, \Big(\, R(c) \ \ \wedge \ \ (\, \neg R(x) \ \vee \ R(f(f(g(x)))) \ \vee \ R(f(x)) \,) \ \ \wedge \ \ \neg R(f(f(x))) \, \Big)$$

über der Signatur $\hat{\sigma} = \{R, c, f, g\}$.

$$\chi \;=\; \forall x \Big(R(c) \;\wedge\; \big(\neg R(x) \vee R(f(f(g(x)))) \vee R(f(x)) \big) \;\wedge\; \neg R(f(f(x))) \Big).$$

Für jeden Grundterm $t \in \mathrm{GT}_{\hat{\sigma}}$ enthält die aussagenlogische Variante $\mathrm{AHE}(\chi)$ der Herbrand-Expansion von χ die aussagenlogische Formel

$$\xi_t \quad := \quad X_{R(c)} \wedge \Big(\neg X_{R(t)} \vee X_{R(f(f(g(t))))} \vee X_{R(f(t))} \Big) \wedge \neg X_{R(f(f(t)))}.$$

Wir zählen die Grundterme in $\mathrm{GT}_{\hat{\sigma}}$ in der folgenden Reihenfolge auf

$$t_1 = c, \quad t_2 = f(c), \quad t_3 = g(c), \quad t_4 = f(f(c)), \quad t_5 = g(f(c)), \quad \ldots$$

und zählen die Formeln in $\mathrm{AHE}(\chi)$ in derselben Reihenfolge auf, also

$$\xi_1 = \xi_{t_1}, \quad \xi_2 = \xi_{t_2}, \quad \xi_3 = \xi_{t_3}, \quad \ldots$$

Bei dem auf Folie 364 beschriebenen Verfahren wird dann beispielsweise im Schleifendurchlauf für $i = 5$ getestet, ob die aussagenlogische Formel

$$\big(\xi_1 \wedge \xi_2 \wedge \xi_3 \wedge \xi_4 \wedge \xi_5 \big)$$

unerfüllbar ist. Dazu können wir beispielsweise das Resolutionsverfahren oder den DPLL-Algorithmus anwenden.

In unserem Beispiel entspricht die Formel $(\xi_1 \wedge \cdots \wedge \xi_5)$ der Klauselmenge

$$\begin{aligned}\Gamma := \Big\{ & \{ X_{R(c)} \} ,\\ & \{ \neg X_{R(c)} , X_{R(f(f(g(c))))} , X_{R(f(c))} \} , \{ \neg X_{R(f(f(c)))} \} ,\\ & \{ \neg X_{R(f(c))} , X_{R(f(f(g(f(c)))))} , X_{R(f(f(c)))} \} , \{ \neg X_{R(f(f(f(c))))} \} ,\\ & \{ \neg X_{R(g(c))} , X_{R(f(f(g(g(c)))))} , X_{R(f(g(c)))} \} , \{ \neg X_{R(f(f(g(c))))} \}\\ & \{ \neg X_{R(f(f(c)))} , X_{R(f(f(g(f(f(c))))))} , X_{R(f(f(f(c))))} \} , \{ \neg X_{R(f(f(f(f(c)))))} \}\\ & \{ \neg X_{R(g(f(c)))} , X_{R(f(f(g(g(f(c))))))} , X_{R(f(g(f(c))))} \} , \{ \neg X_{R(f(f(g(f(c)))))} \} \Big\}\end{aligned}$$

Wir konstruieren eine Resolutionswiderlegung für Γ:

(1)	$\{ X_{R(c)} \}$	in Γ
(2)	$\{ \neg X_{R(c)} , X_{R(f(f(g(c))))} , X_{R(f(c))} \}$	in Γ
(3)	$\{ X_{R(f(f(g(c))))} , X_{R(f(c))} \}$	Resolvente aus 1,2
(4)	$\{ \neg X_{R(f(f(g(c))))} \}$	in Γ
(5)	$\{ X_{R(f(c))} \}$	Resolvente aus 3,4
(6)	$\{ \neg X_{R(f(c))} , X_{R(f(f(g(f(c)))))} , X_{R(f(f(c)))} \}$	in Γ
(7)	$\{ X_{R(f(f(g(f(c)))))} , X_{R(f(f(c)))} \}$	Resolvente aus 5,6
(8)	$\{ \neg X_{R(f(f(c)))} \}$	in Γ
(9)	$\{ X_{R(f(f(g(f(c)))))} \}$	Resolvente aus 7,8
(10)	$\{ \neg X_{R(f(f(g(f(c)))))} \}$	in Γ
(11)	$\emptyset$	Resolvente aus 9,10

Somit ist Γ unerfüllbar (gemäß Satz 2.60). Das auf Folie 364 angegebene Verfahren hält daher (spätestens) im Schleifendurchlauf für $i = 5$ mit der Ausgabe „χ ist unerfüllbar" an. Da χ erfüllbarkeitsäquivalent zur Formel $(\varphi \wedge \neg\psi)$ ist, wissen wir also, dass $\varphi \models \psi$ gilt.
Dies beendet Beispiel 4.54.

Kapitel 5:

Logik-Programmierung

Abschnitt 5.1:

Einführung

Logik-Programmierung

Logik-Programmierung bezeichnet die Idee, Logik direkt als Programmiersprache zu verwenden.

Logik-Programmierung (in Sprachen wie Prolog) und die verwandte funktionale Programmierung (in Sprachen wie LISP, ML, Haskell) sind deklarativ, im Gegensatz zur imperativen Programmierung (in Sprachen wie Java, C, Perl).

Die Idee der deklarativen Programmierung besteht darin, dem Computer lediglich sein Wissen über das Anwendungsszenario und sein Ziel mitzuteilen und dann die Lösung des Problems dem Computer zu überlassen.

Bei der imperativen Programmierung hingegen gibt man dem Computer die einzelnen Schritte zur Lösung des Problems vor.

Prolog

- ist die wichtigste logische Programmiersprache,
- geht zurück auf Kowalski und Colmerauer
  (Anfang der 1970er Jahre, Marseilles),
- steht für (franz.) Programmation en logique.
- Mitte/Ende der 1970er Jahre: effiziente Prolog-Implementierung durch den von Warren (in Edinburgh) entwickelten Prolog-10 Compiler.

Prolog ist eine voll entwickelte und mächtige Programmiersprache, die vor allem für symbolische Berechnungsprobleme geeignet ist.

Aus Effizienzgründen werden in Prolog die abstrakten Ideen der logischen Programmierung nicht in Reinform umgesetzt, Prolog hat auch „nichtlogische“ Elemente.

Dieses Kapitel

- setzt voraus, dass Sie bereits Grundkenntnisse der Programmiersprache *Prolog* besitzen, die beispielsweise im Buch „Learn Prolog Now!“ von P. Blackburn, J. Bos und K. Striegnitz vermittelt werden, und die während des Semesters bereits im Übungsbetrieb behandelt wurden.

- gibt eine Einführung in die Grundlagen der Logik-Programmierung — *keine* Einführung in die Programmiersprache Prolog!

Auf einige der Hauptunterschiede zwischen allgemeiner Logik-Programmierung und Prolog werden wir im Laufe dieses Kapitels eingehen.

Alle in diesem Kapitel enthaltenen Beispiele von Logikprogrammen sind voll lauffähige Prologprogramme, aber in einigen Fällen unterscheidet sich die Semantik des Programms im Sinne der Logik-Programmierung von der Semantik des Programms im Sinne von Prolog.

Zunächst zwei Beispiele für Logikprogramme

Beispiel 5.1

Ein Logikprogramm zur Repräsentation natürlicher Zahlen in Unärdarstellung und der zugehörigen Arithmetik und der Kleiner-Relation.

Programm: `unat.pl`

```
unat(null).
unat(s(X)) :- unat(X).

plus(null, Y, Y).
plus(s(X), Y, s(Z)) :- plus(X, Y, Z).

minus(X, Y, Z) :- plus(Y, Z, X).

mal(null, Y, null).
mal(s(X), Y, Z) :- mal(X, Y, Z1), plus(Z1, Y, Z).

less(null, s(_)).
less(s(X), s(Y)) :- less(X, Y).
```

Beispiel 5.2

Abschnitt 5.2:

Syntax und deklarative Semantik von Logikprogrammen

Logikprogramme

Logikprogramme sind „Wissensbasen“, bestehend aus einer endlichen Menge von Fakten und Regeln.

Eine Berechnung eines Logikprogramms besteht aus der Ableitung der Konsequenzen, die aus den Fakten und den Regeln des Programms hergeleitet werden können.

Man führt ein Programm aus, indem man Anfragen an die Wissensbasis stellt.

Fakten beschreiben Relationen zwischen Objekten.

Beispiele: `vater(gerald,scarlett)`, `maennlich(rhett)`, `party`, `plus(s(null),s(s(null)),s(s(s(null))))`.

Relationen haben eine Stelligkeit $k \geqslant 0$.
Nullstellige Relationen sind einfach Aussagen (z.B. besagt „`party`“, dass die Party stattfindet).

Eine **Anfrage** ist eine durch Kommas getrennte Liste von Fakten; gefragt wird, ob diese Fakten in der Wissenbasis gelten, d.h., ob sie aus der Wissensbasis ableitbar sind.

Beispiele: Die Anfrage `?- schwester(scarlett, suellen)` fragt, ob Scarlett eine Schwester von Suellen ist.
Die Anfrage `?- mutter(scarlett, X), vater(ashley, X)` fragt, ob Scarlett und Ashley ein gemeinsames Kind haben.

Die Rolle der Terme

Terme sind in Logikprogrammen die universelle Datenstruktur.

Je nach Kontext spielen sie die Rolle von Fakten
oder von Objekten, über die die Fakten sprechen.

Die einfachste Art von Termen in Logikprogrammen sind die im Folgenden definierten Konstanten und Variablen.

Atome, Zahlen, Konstanten und Variablen der Logik-Programmierung

Definition 5.3

(a) Atome sind die Grundbausteine von Logikprogrammen. Sie werden bezeichnet durch Zeichenketten, die keins der Symbole „`(`“ und „`)`“ enthalten und die mit einem Kleinbuchstaben beginnen oder in einfachen Hochkommata stehen. Atome repräsentieren Individuen.

Beispiele: `scarlett`, `'Scarlett'`, `logikInDerInformatik`

(b) Zahlen in Logikprogrammen sind entweder ganze Zahlen oder reelle Zahlen in Gleitkommadarstellung.

Beispiele: `42`, `1.2e-3`

(c) Konstanten der Logik-Programmierung sind Atome oder Zahlen.

Definition 5.4

Variablen der Logik-Programmierung werden durch Zeichenketten bezeichnet, die mit einem Großbuchstaben oder einem Unterstrich beginnen und keins der Symbole „(" und „)" enthalten.

Eine Variable repräsentiert in einem Logikprogramm (ähnlich wie in der Logik erster Stufe) ein nicht-spezifiziertes Individuum.

Man beachte den Gegensatz zur imperativen Programmierung, bei der eine Variable für eine „Speicherzelle" steht, in der Werte gespeichert und verändert werden können.

Beispiele: `X`, `Mutter`, `_mutter`, `RUD26`

Terme der Logik-Programmierung

Definition 5.5

(a) Ein einfacher Term der Logik-Programmierung ist eine Konstante oder eine Variable (d.h., ein Atom, eine Zahl oder eine Variable der Logik-Programmierung).

(b) Die Menge T_{LP} der Terme der Logik-Programmierung ist rekursiv wie folgt definiert:

  (1) Jeder einfache Term ist ein Term.
  (2) Ist `f` ein Atom, ist $k \in \mathbb{N}$ mit $k \geqslant 1$ und sind $t_1, \ldots, t_k \in \mathrm{T}_{\mathrm{LP}}$ Terme, so ist

$$\mathtt{f}(t_1, \ldots, t_k)$$

  ein Term in T_{LP}.

(c) Terme in T_{LP}, die keine einfachen Terme sind, heißen *zusammengesetzte Terme* der Logik-Programmierung.

In einem zusammengesetzten Term der Form $\mathtt{f}(t_1, \ldots, t_k)$ spielt das Atom `f` die Rolle eines k-stelligen Funktors, den wir mit `f/k` bezeichnen.

Spezialfall $k = 0$: Jedes Atom g wird als ein 0-stelliger Funktor betrachtet, der mit g/0 bezeichnet wird, und der ein (einfacher) Term ist.

Beispiele: `party`, `vater(gerald,scarlett)`, `s(s(s(null)))`,

```
vorlesung(name(logikInDerInformatik),
          zeit(Mi,9,11),
          ort(gebaeude(RUD26),raum(0110))).
```

Gleichheit von Termen

Zwei Terme t und t' der Logik-Programmierung werden nur dann als *gleich* bezeichnet, wenn sie syntaktisch, d.h. als Zeichenketten betrachtet, identisch sind.

Beispiel:
Die beiden Terme `plus(null,X,X)` und `plus(null,Y,Y)` sind nicht gleich.

Substitutionen

Notation

Für eine partielle Funktion f schreiben wir $\mathrm{Def}(f)$ und $\mathrm{Bild}(f)$ um den Definitionsbereich und den Bildbereich von f zu bezeichnen.
D.h. $\mathrm{Def}(f)$ ist die Menge aller Objekte x, für die der Wert $f(x)$ definiert ist, und $\mathrm{Bild}(f) = \{f(x) \ : \ x \in \mathrm{Def}(f)\}$.

Definition 5.6

Eine Substitution ist eine partielle Abbildung von der Menge der Variablen auf die Menge der Terme.

Eine Substitution für eine Menge V von Variablen der Logik-Programmierung ist eine Substitution S mit $\mathrm{Def}(S) \subseteq V$.

Beispiel:

$$S := \left\{ \mathtt{X} \mapsto \mathtt{c}, \ \ \mathtt{Y} \mapsto \mathtt{f(X,g(c))}, \ \ \mathtt{Z} \mapsto \mathtt{Y} \right\}$$

bezeichnet die Substitution mit Definitionsbereich $\mathrm{Def}(S) = \{\mathtt{X}, \mathtt{Y}, \mathtt{Z}\}$, für die gilt:
$S(\mathtt{X}) = \mathtt{c}, \ \ S(\mathtt{Y}) = \mathtt{f(X,g(c))}, \ \ S(\mathtt{Z}) = \mathtt{Y}$.

Anwendung von Substitutionen

Durch Anwenden einer Substitution S auf einen Term $t \in \mathrm{T}_{\mathrm{LP}}$ erhalten wir den Term $tS \in \mathrm{T}_{\mathrm{LP}}$, der aus t durch simultanes Ersetzen jeder Variablen $X \in \mathrm{Def}(S)$ durch den Term $S(X)$ entsteht.

Beispiel: Sei

$$t \quad := \quad \mathtt{h(f(X,X),\ Y,\ f(Y,g(Z)))}$$

und

$$S \;:=\; \{\, \mathtt{X} \mapsto \mathtt{c},\;\; \mathtt{Y} \mapsto \mathtt{f(X,g(c))},\;\; \mathtt{Z} \mapsto \mathtt{Y} \,\}.$$

Dann ist

$$tS \quad = \quad \mathtt{h(f(c,c),\ f(X,g(c)),\ f(f(X,g(c)),\ g(Y)))}.$$

Definition 5.7

Ein Term t' ist eine Instanz eines Terms t, wenn es eine Substitution S gibt, so dass $t' = tS$.

Grundterme

Definition 5.8

Ein Grundterm der Logik-Programmierung ist ein Term, der keine Variable(n) enthält.

Eine Grundinstanz eines Terms $t \in \mathrm{T}_{\mathrm{LP}}$ ist eine Instanz von t, die ein Grundterm ist.

Eine Grundinstanz eines Terms t entsteht also, indem jede in t vorkommende Variable durch einen Grundterm ersetzt wird.

Beispiele: `h(c,c,f(c))` und `h(f(f(c,c),g(d)),d,f(g(g(c))))` sind Grundinstanzen des Terms `h(X,Y,f(Z))`.

Bemerkung

Grundterme sind wichtig, weil sie in dem Modell, das dem Logikprogramm zu Grunde liegt, eine unmittelbare Bedeutung haben. Variablen hingegen haben keine direkte Bedeutung, sondern sind nur Platzhalter für Objekte.

Fakten der Logik-Programmierung

Definition 5.9

Ein Faktum der Logik-Programmierung ist ein Atom oder ein zusammengesetzter Term der Logik-Programmierung.

Fakten beschreiben Tatsachen bzw. Relationen zwischen Objekten.

Beispiele: Das Faktum `party` beschreibt, dass eine Party stattfindet.
Das Faktum `unat(s(s(null)))` beschreibt, dass der Term `s(s(null))` die Unärdarstellung einer natürlichen Zahl ist.
Das Faktum `mutter(scarlett, bonnie)` beschreibt, dass Scarlett die Mutter von Bonnie ist.

Fakten dürfen auch Variablen enthalten. Eine Variable in einem Faktum bedeutet, dass die entsprechende Aussage für *alle* Objekte, durch die die Variable ersetzt werden kann, gilt.
Beispiel: `plus(null,Y,Y)`

Regeln

Definition 5.10

Eine Regel der Logik-Programmierung besteht aus

- einem Faktum (dem so genannten Kopf der Regel),
- gefolgt von `:-`
  (in der Literatur wird an Stelle von „`:-`“ oft auch „$\leftarrow$“ geschrieben) und
- einer durch Kommas getrennten Liste von Fakten (dem so genannten Rumpf der Regel).

Wir interpretieren die Regel als Implikation:

Wenn alle Fakten im Rumpf gelten, dann gilt auch das Faktum im Kopf.

Beispiele:

```
minus(X,Y,Z) :- plus(Y,Z,X)
grossmutter(X,Z) :- mutter(X,Y), elternteil(Y,Z)
```

Logikprogramme

Definition 5.11

Ein Logikprogramm ist eine endliche Menge von Fakten und Regeln der Logik-Programmierung.

Es ist oft bequem, Fakten als spezielle Regeln mit leerem Rumpf aufzufassen. Dann besteht ein Logikprogramm nur aus Regeln.

In konkreten Beispielen stellen wir Logikprogramme meistens als Liste der in ihnen enthaltenen Fakten und Regeln dar, wobei das Ende jedes Eintrags dieser Liste durch einen Punkt markiert wird.

Beispiele: Das Programm `unat.pl` aus Beispiel 5.1 ist ein Logikprogramm im Sinne von Definition 5.11. Das Programm `vomWindeVerweht.pl` aus Beispiel 5.2 nicht, da dort Ungleichheitsprädikate der Form `X \== Y` vorkommen, die gemäß Definition 5.10 nicht im Rumpf von Regeln vorkommen können, da sie keine Fakten gemäß Definition 5.9 sind.

Ableitungen aus Logikprogrammen

Definition 5.12

Eine Ableitung aus einem Logikprogramm Π ist ein Tupel $(t_1, \ldots, t_\ell)$ von Termen, so dass $\ell \in \mathbb{N}$ mit $\ell \geqslant 1$ ist und für jedes $i \in [\ell]$ (mindestens) eine der beiden folgenden Aussagen zutrifft:

- t_i ist eine Instanz eines Faktums in Π.
- Es gibt eine Regel

  $$\varphi \;\; \texttt{:-} \;\; \psi_1, \ldots, \psi_m$$

  in Π, eine Substitution S und Indizes $i_1, \ldots, i_m \in \{1, \ldots, i{-}1\}$, so dass gilt: $t_i = \varphi S \quad$ und $\quad t_{i_j} = \psi_j S$ für jedes $j \in [m]$.

Eine Ableitung eines Terms t aus Π ist eine Ableitung $(t_1, \ldots, t_\ell)$ aus Π mit $t_\ell = t$.

Ein Term t ist ableitbar aus Π, wenn es eine Ableitung von t aus Π gibt.

Die im Kapitel über Automatisches Schließen eingeführte Kalkül-Schreibweise lässt sich dazu nutzen, eine elegante Darstellung des Begriffs der Ableitungen aus Logikprogrammen anzugeben.

Verwendung der Kalkül-Schreibweise für Ableitungen in Logikprogrammen

Sei Π ein Logikprogramm.

Gesucht: Ein Kalkül $\mathfrak{K}_\Pi$ über der Menge T_{LP}, so dass $\mathsf{abl}_{\mathfrak{K}_\Pi}$ genau die Menge aller aus Π ableitbaren Terme ist.

Darstellung von Ableitungen

- An Stelle von $(t_1, \ldots, t_\ell)$
  schreiben wir Ableitungen der besseren Lesbarkeit halber oft zeilenweise, also

$$\begin{array}{ll} (1) & t_1 \\ (2) & t_2 \\ \vdots & \\ (\ell) & t_\ell \end{array}$$

  und geben am Ende jeder Zeile eine kurze Begründung an.

- Ableitungen werden oft auch als Bäume dargestellt; man bezeichnet diese als Beweisbäume.

Beispiel

Betrachte das Programm `vomWindeVerweht1.pl`

Beispiel 5.13

Ableitung von `tante(suellen,bonnie)` aus dem Programm `vomWindeVerweht1.pl`:

(1)	`mutter(ellen,scarlett)`	Faktum in Zeile 3
(2)	`elternteil(ellen,scarlett)`	Regel in Zeile 25 und (1)
(3)	`mutter(ellen,suellen)`	Faktum in Zeile 3
(4)	`elternteil(ellen,suellen)`	Regel in Zeile 25 und (3)
(5)	`ungleich(suellen,scarlett)`	Regel in Zeile 32
(6)	`weiblich(suellen)`	Faktum in Zeile 16
(7)	`schwester(suellen,scarlett)`	Regel in Zeile 27 und (4),(2),(6),(5)
(8)	`mutter(scarlett,bonnie)`	Faktum in Zeile 4
(9)	`elternteil(scarlett,bonnie)`	Regel in Zeile 25 und (8)
(10)	`tante(suellen,bonnie)`	Regel in Zeile 27 und (9),(7)

Beweisbäume

Definition 5.14

Sei Π ein Logikprogramm und sei t ein Term.
Ein Beweisbaum für t aus Π ist ein endlicher Baum, dessen Knoten mit Termen beschriftet sind, so dass gilt:

- die Wurzel ist mit dem „Ziel“ t beschriftet,
- jedes Blatt ist mit einer Instanz eines Faktums in Π beschriftet, und
- für jeden inneren Knoten u und dessen Kinder $v_1, \ldots, v_m$ gilt:
  Es gibt eine Regel

  $$\varphi \ \texttt{:-}\ \psi_1, \ldots, \psi_m$$

  in Π und eine Substitution S, so dass für die Beschriftung t_u von u und die Beschriftungen $t_{v_1}, \ldots, t_{v_m}$ der Knoten $v_1, \ldots, v_m$ gilt:

  $$t_u = \varphi S, \quad t_{v_1} = \psi_1 S, \quad t_{v_2} = \psi_2 S, \quad \ldots, \quad t_{v_m} = \psi_m S.$$

Man sieht leicht, dass es genau dann einen Beweisbaum für t aus Π gibt, wenn t aus Π ableitbar ist (Details: Übung).

Deklarative Semantik von Logikprogrammen

Definition 5.15

Sei Π ein Logikprogramm.

Die Bedeutung von Π ist die Menge $\mathcal{B}(\Pi)$ aller Grundterme, die aus Π ableitbar sind.

Beispiel 5.16

Sei Π das folgende Logikprogramm `unat1.pl`.

Programm: `unat1.pl`

```
unat(null).
unat(s(X)) :- unat(X).
less(null, s(X)) :- unat(X).
less(s(X), s(Y)) :- less(X, Y).
```

Die Bedeutung von Π ist die Menge $\mathcal{B}(\Pi)$, und diese enthält u.a. die Terme

Beispiel: Wege in Digraphen (d.h., gerichteten Graphen)

Wir repräsentieren einen gerichteten Graphen G durch die Auflistung `node(v)` für alle Knoten `v` von G und `edge(v,w)` für alle Kanten `(v,w)` von G.

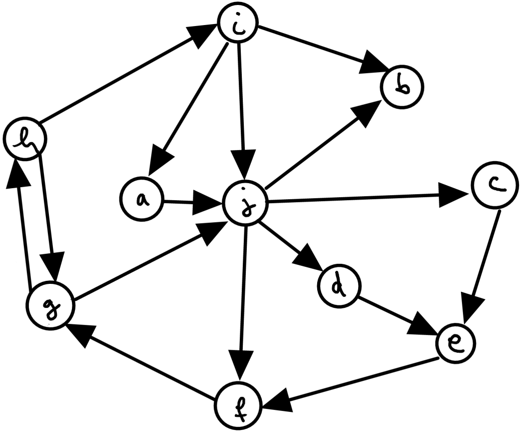
Ziel: `path(X,Y)` soll besagen, dass es in G einen Weg von Knoten `X` zu Knoten `Y` gibt.

Lösung:

```
path(X,X).
path(X,Y) :- edge(X,Z), path(Z,Y).
```

Im folgenden Programm `digraph.pl` ist dies zusammen mit einem Beispiel-Graphen gegeben.

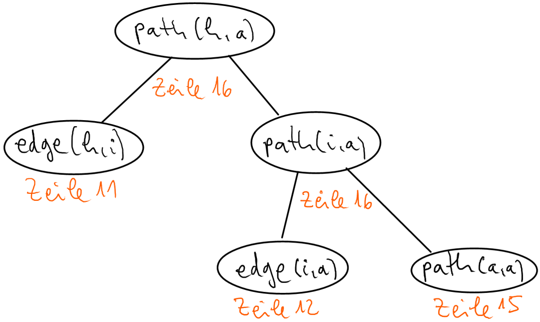
Der in `digraph.pl` angegebene Graph sieht wie folgt aus:



Ein Beweisbaum für `path(a,g)` aus `digraph.pl`:

Ein Beweisbaum für `path(h,a)` aus `digraph.pl`:

Idee: Wähle den Pfad $h \to i \to a$.



Und was tut Prolog bei Eingabe von

```
?- consult(digraph).
?- path(a,g).
```

und bei Eingabe von

```
?- path(h,a).
```

?

Auf die Frage, ob `path(a,g)` gilt, antwortet Prolog mit „`true`“.

Auf die Frage, ob `path(h,a)` gilt, antwortet Prolog mit „`ERROR: Out of local stack`“.

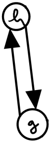
Was passiert hier?

Die Details zur Berechnung, die Prolog hier durchführt, können wir mit uns mit

```
?- trace.
?- path(h,a).
```

anschauen.

Dies zeigt, dass die Prolog-Suche nach einem Beweisbaum im Kreis



stecken bleibt.

Unterschied zwischen Theorie und Praxis

In der Theorie funktioniert die Pfadsuche aus `digraph.pl` für alle endlichen gerichteten Graphen.

In der Praxis funktioniert sie aber nur für azyklische Graphen.

Die operationelle Semantik von Prolog entspricht also nicht genau der deklarativen Semantik von Logikprogrammen!

Anfragen an Logikprogramme

Definition 5.17

Eine Anfrage der Logik-Progammierung besteht aus den Symbolen `?-` gefolgt von einem Faktum oder aus einer durch Kommas getrennten Liste von Fakten der Logik-Programmierung.

Die Antwort auf eine Anfrage α der Form

$$\texttt{?-}\ \alpha_1, \ldots, \alpha_n$$

an ein Logikprogramm Π ist definiert als die Menge $[\![\alpha]\!]^{\Pi}$ aller Substitutionen S für die in α vorkommenden Variablen, so dass gilt:
$\alpha_1 S$, ..., $\alpha_n S$ sind Grundterme, die aus Π ableitbar sind.

Hier repräsentiert die leere Menge $\emptyset$ die Antwort „falsch“.

Beachte: Eine Variable `X` in einer Anfrage fragt also nach einem bzw. allen Objekten, die die Anfrage erfüllen.

Beispiel 5.18

Betrachte die Anfrage

```
?- vater(gerald,X), mutter(ellen,X)
```

angewendet auf das Logikprogramm `vomWindeVerweht1.pl`.

Die Antwort auf diese Anfrage besteht aus den drei Substitutionen

$$\begin{aligned} S_1 &:= \{\, \mathtt{X} \mapsto \mathtt{scarlett} \,\}, \\ S_2 &:= \{\, \mathtt{X} \mapsto \mathtt{suellen} \,\}, \\ S_3 &:= \{\, \mathtt{X} \mapsto \mathtt{carreen} \,\}. \end{aligned}$$

Beispiele von Anfragen an das Logikprogramm `unat.pl`:

```
?- plus(s(null),s(s(null)),X).
?- plus(X,Y,s(s(s(null)))).
```

Abschnitt 5.3:

Operationelle Semantik

Deklarative vs. Operationelle Semantik

- Die in Definition 5.15 festgelegte deklarative Semantik von Logikprogrammen beruht auf einer logischen Interpretation von Programmen (Regeln als Implikationen) und logischer Deduktion.

- Jetzt werden wir dieser deklarativen Semantik eine operationelle Semantik gegenüberstellen, indem wir einen Algorithmus angeben, der Programme ausführt (auf einem abstrakten, nichtdeterministischen Maschinenmodell). Dadurch legen wir ebenfalls die Antworten auf die Anfragen fest und weisen somit Programmen eine Bedeutung zu.

- Wir werden sehen, dass die deklarative Bedeutung von Logikprogrammen mit der operationellen übereinstimmt.

Semantik von Programmiersprachen im Allgemeinen

Generell unterscheidet man zwischen zwei Wegen, die Semantik von Programmiersprachen zu definieren:

- Die deklarative oder denotationelle Semantik ordnet Programmen Objekte in abstrakten mathematischen Räumen zu, in der Regel partielle Funktionen, oder im Fall von Logikprogrammen Mengen von Grundtermen.

  Zur Erinnerung: Die Bedeutung $\mathcal{B}(\Pi)$ eines Logikprogramms Π ist gemäß Definition 5.15 die die Menge aller Grundterme, die aus Π ableitbar sind.

- Die operationelle Semantik legt fest, wie Programme auf abstrakten Maschinenmodellen ausgeführt werden.

Notation

- LP := die Menge aller Logikprogramme

- A_{LP} := die Menge aller Atome der Logik-Programmierung

  V_{LP} := die Menge aller Variablen der Logik-Programmierung

  K_{LP} := die Menge aller Konstanten der Logik-Programmierung

  T_{LP} := die Menge aller Terme der Logik-Programmierung

  F_{LP} := die Menge aller Anfragen der Logik-Programmierung

  R_{LP} := die Menge aller Regeln der Logik-Programmierung

- Für jedes ξ aus $\mathsf{T}_{\mathsf{LP}} \cup \mathsf{F}_{\mathsf{LP}} \cup \mathsf{R}_{\mathsf{LP}} \cup \mathsf{LP}$ bezeichnet $\mathsf{Var}(\xi)$ die Menge aller Variablen, die in ξ vorkommen.

  Beispiel: Ist ρ die Regel `path(X,Y) :- edge(X,Z), path(Z,Y).` dann ist $\mathsf{Var}(\rho) = \{\mathtt{X}, \mathtt{Y}, \mathtt{Z}\}$.

- Ist S eine Substitution und $\alpha \in \mathsf{F}_{\mathsf{LP}}$ eine Anfrage der Form `?-` $\alpha_1, \ldots, \alpha_m$ ist, so bezeichnet αS die Anfrage `?-` $\alpha_1 S, \ldots, \alpha_m S$.
  Entsprechend definieren wir für jede Regel $\rho \in \mathsf{R}_{\mathsf{LP}}$ die Regel ρS.

Mehr über Substitutionen

- Zur Erinnerung: Eine Substitution ist eine partielle Abbildung S von $\mathrm{V_{LP}}$ nach $\mathrm{T_{LP}}$. Den Definitionsbereich von S bezeichnen wir mit $\mathrm{Def}(S)$, den Bildbereich mit $\mathrm{Bild}(S)$.

- Die Verkettung zweier Substitutionen S und T ist die Substitution ST mit $\mathrm{Def}(ST) = \mathrm{Def}(S) \cup \mathrm{Def}(T)$ und $\mathtt{X}(ST) := (\mathtt{X}S)T$ für alle $\mathtt{X} \in \mathrm{Def}(ST)$.

- Die Einschränkung einer Substitution S auf eine Menge V von Variablen ist die Substitution $S|_V$ mit $\mathrm{Def}(S|_V) = \mathrm{Def}(S) \cap V$ und $\mathtt{X}S|_V := \mathtt{X}S$ für alle $\mathtt{X} \in \mathrm{Def}(S) \cap V$.

- Die leere Substitution bezeichnen wir mit I. Es gilt:
  - $tI = t$ für alle Terme $t \in \mathrm{T_{LP}}$, und
  - $IS = SI = S$ für alle Substitutionen S.

Beispiel 5.19

Für die Substitutionen

$$\begin{aligned} S &:= \{\, \mathtt{X} \mapsto \mathtt{good(c,Y)},\ \ \mathtt{Y} \mapsto \mathtt{rainy(d)} \,\}, \\ T &:= \{\, \mathtt{Y} \mapsto \mathtt{sunny(d)},\ \ \ \mathtt{Z} \mapsto \mathtt{humid(e)} \,\}. \end{aligned}$$

gilt:

$$\begin{aligned} ST &= \{\, \mathtt{X} \mapsto \mathtt{good(c,sunny(d))},\ \ \mathtt{Y} \mapsto \mathtt{rainy(d)},\ \ \mathtt{Z} \mapsto \mathtt{humid(e)} \,\} \\ TS &= \{\, \mathtt{X} \mapsto \mathtt{good(c,Y)},\ \ \mathtt{Y} \mapsto \mathtt{sunny(d)},\ \ \mathtt{Z} \mapsto \mathtt{humid(e)} \,\}. \end{aligned}$$

Umbennungen

- Eine Umbenennung ist eine injektive partielle Abbildung von V_{LP} nach V_{LP}.

  Wegen $\mathsf{V}_{\mathsf{LP}} \subseteq \mathsf{T}_{\mathsf{LP}}$, sind Umbenennungen spezielle Substitutionen.

- Eine Umbenennung für eine Menge V von Variablen ist eine Umbennung U mit $\mathrm{Def}(U) = V$.

- Ist U eine Umbennung, so bezeichnet U^{-1} ihre Umkehrung.

Beispiel: $U := \{\mathtt{X} \mapsto \mathtt{Y},\ \mathtt{Y} \mapsto \mathtt{Z}\}$ ist eine Umbenennung für $\{\mathtt{X}, \mathtt{Y}\}$.
$U^{-1} = \{\mathtt{Y} \mapsto \mathtt{X},\ \mathtt{Z} \mapsto \mathtt{Y}\}$ ist die Umkehrung von U.

Ein einfacher Interpreter für Logikprogramme

Algorithmus $\textsc{Antwort}(\Pi, \alpha)$

% Eingabe: Programm $\Pi \in \mathsf{LP}$, *Anfrage* ?- $\alpha \in \mathsf{F_{LP}}$ *mit* $\alpha = \alpha_1, \ldots, \alpha_m$

% Ausgabe: eine Substitution S *für* $\mathsf{Var}(\alpha)$ *oder das Wort „gescheitert".*

1. Wähle ein $i \in [m]$ *% α_i ist das nächste „Ziel"*
2. Wähle eine Regel ρ aus Π. Sei φ :- $\psi_1, \ldots, \psi_n$ die Form von ρ.
   % Fakten fassen wir als Regeln ohne Rumpf auf
3. Sei U eine Umbennung für $\mathsf{Var}(\rho)$, so dass $\mathsf{Var}(\rho U) \cap \mathsf{Var}(\alpha) = \emptyset$.
4. Wähle eine Substitution T, so dass $\alpha_i T = \varphi U T$. Wenn dies nicht möglich ist, gib „gescheitert" aus und halte an.
5. Wenn $m = 1$ und $n = 0$, gib $T|_{\mathsf{Var}(\alpha)}$ aus und halte an.
6. Setze $\alpha' := \alpha_1 T, \ldots, \alpha_{i-1} T, \psi_1 U T, \ldots, \psi_n U T, \alpha_{i+1} T, \ldots, \alpha_m T$.
7. Setze $T' := \textsc{Antwort}(\Pi, \alpha')$
8. Wenn T' eine Substitution ist, gib $(TT')|_{\mathsf{Var}(\alpha)}$ aus und halte an.
9. Gib „gescheitert" aus und halte an.

Zum Nichtdeterminismus des Interpreters

- Das Programm ANTWORT ist nichtdeterministisch. Wir sprechen von verschiedenen Läufen des Programms, die durch die Auswahlen in den Zeilen 1–4 bestimmt sind.

- Ein Lauf heißt akzeptierend, wenn die Ausgabe eine Substitution ist.

- Von den nichtdeterministischen Auswahlschritten in den Zeilen 1–4 ist die Wahl der Substitution in Zeile 4 am problematischsten, weil hier ein Element einer unendlichen Menge ausgewählt wird, und weil nicht klar ist, wie man so ein Element überhaupt finden kann.

- Die Wahl der Umbennung in Zeile 3 hingegen ist unwesentlich. Jede Umbennung U, für die $\text{Var}(\rho U) \cap \text{Var}(\alpha) = \emptyset$ gilt, führt zum gleichen Ergebnis, und es ist leicht, eine solche Umbennung zu finden.

Korrektheit und Vollständigkeit des Interpreters

Satz 5.20

Seien $\Pi \in \mathsf{LP}$ *ein Logikprogramm, sei* `?-` $\alpha \in \mathsf{F}_{\mathsf{LP}}$ *eine Anfrage mit* $\alpha = \alpha_1, \ldots, \alpha_m$*, und sei* S *eine Substitution für* $\mathrm{Var}(\alpha)$*. Dann sind folgende Aussagen äquivalent:*

(a) *Die Terme* $\alpha_1 S, \ldots, \alpha_m S$ *sind aus* Π *ableitbar.*

(b) *Es gibt einen Lauf von* $\textsc{Antwort}(\Pi, \alpha)$*, der* S *ausgibt.*

Die Richtung *„(b)* $\Longrightarrow$ *(a)"* wird *Korrektheit des Interpreters* genannt; die Richtung *„(a)* $\Longrightarrow$ *(b)" Vollständigkeit*.

Für den Spezialfall, dass $m = 1$ und α ein Grundterm ist, erhalten wir das folgende Korollar.

Korollar 5.21

Sei $\Pi \in \mathsf{LP}$ *ein Programm und sei* α *ein Grundterm. Dann gilt:*
$\alpha \in \mathcal{B}(\Pi) \iff$ *es gibt einen akzeptierenden Lauf von* $\textsc{Antwort}(\Pi, \alpha)$*.*

Nächstes Ziel:
Auflösen des Nichtdeterminismus in Zeile 4

Als ein Hauptproblem des nichtdeterministischen Interpreters ANTWORT haben wir die Wahl der Substitution T in Zeile 4 identifiziert.

Mit Hilfe der im Folgenden vorgestellten Unifikatoren können die richtigen Substitutionen auf deterministische Art gefunden werden.

Unifikation

Definition 5.22

Seien $t, s \in \mathrm{T}_{\mathrm{LP}}$ Terme der Logik-Programmierung.

(a) Ein Unifikator für t und s ist eine Substitution S, so dass $tS = sS$.

(b) t und s sind unifizierbar, wenn es einen Unifikator für t und s gibt.

Beispiel 5.23

$t := \texttt{mal(s(X), Y, s(Z))}$ und $s := \texttt{mal(s(s(null)), Y, Y)}$ sind unifizierbar.
Ein Unifikator ist

$$S \;:=\; \{\, \texttt{X} \mapsto \texttt{s(null)},\;\; \texttt{Y} \mapsto \texttt{s(Z)} \,\}.$$

Die entstehende gemeinsame Instanz ist
$tS \;=\; \texttt{mal(s(s(null)), s(Z), s(Z))} \;=\; sS.$

Ein weiterer Unifikator für t und s ist

$$S' \;:=\; \{\, \texttt{X} \mapsto \texttt{s(null)},\;\; \texttt{Y} \mapsto \texttt{s(null)},\;\; \texttt{Z} \mapsto \texttt{null} \,\}.$$

Die entstehende gemeinsame Instanz ist
$tS' \;=\; \texttt{mal(s(s(null)), s(null), s(null))} \;=\; sS'.$

Eine Ordnung auf den Substitutionen

Definition 5.24

Zwei Substitutionen S und T sind äquivalent (kurz: $S \equiv T$), wenn für alle Variablen $\mathtt{X} \in \mathrm{V_{LP}}$ gilt: $\mathtt{X}S = \mathtt{X}T$.

Beobachtung:

S und T sind genau dann äquivalent, wenn $\mathtt{X}S = \mathtt{X}T$ für alle $\mathtt{X} \in \mathrm{Def}(S) \cap \mathrm{Def}(T)$ und $\mathtt{X}S = \mathtt{X}$ für alle $\mathtt{X} \in \mathrm{Def}(S) \setminus \mathrm{Def}(T)$ und $\mathtt{X}T = \mathtt{X}$ für alle $\mathtt{X} \in \mathrm{Def}(T) \setminus \mathrm{Def}(S)$.

Definition 5.25

Seien S und T Substitutionen. S ist allgemeiner als T (wir schreiben $S \leqq T$), wenn es eine Substitution S' gibt, so dass $SS' \equiv T$.

Beobachtung:

I ist eine allgemeinste Substitution, d.h. für jede Substitution T gilt $I \leqq T$.

Allgemeinste Unifikatoren (kurz: mgu, für „most general unifier“)

Definition 5.26

Seien $t, s \in \mathrm{T}_{\mathrm{LP}}$. Ein allgemeinster Unifikator für t und s ist ein Unifikator S für t und s, so dass gilt: $S \leqq T$ für alle Unifikatoren T für t und s.

Das folgende Lemma besagt, dass allgemeinste Unifikatoren bis auf Umbenennung von Variablen eindeutig sind.

Lemma 5.27

Seien $t, s \in \mathrm{T}_{\mathrm{LP}}$, und seien S, T allgemeinste Unifikatoren für t und s. Dann gibt es eine Umbennenung U, so dass $SU \equiv T$.

Ein Unifikationsalgorithmus

Algorithmus $\mathrm{MGU}(t, s)$

% Eingabe: zwei Terme $t, s \in \mathsf{T}_{\mathsf{LP}}$.

% Ausgabe: eine Substitution S oder die Worte „nicht unifizierbar"

1. Wenn $t = s$, dann gib I aus und halte an.
2. Wenn $t = \mathtt{X} \in \mathsf{V}_{\mathsf{LP}}$
3. Wenn $\mathtt{X} \in \mathsf{Var}(s)$, dann gib „nicht unifizierbar" aus und halte an.
4. Gib $\{\mathtt{X} \mapsto s\}$ aus und halte an.
5. Wenn $s = \mathtt{X} \in \mathsf{V}_{\mathsf{LP}}$
6. Wenn $\mathtt{X} \in \mathsf{Var}(t)$ dann gib „nicht unifizierbar" aus und halte an.
7. Gib $\{\mathtt{X} \mapsto t\}$ aus und halte an.
8. Wenn $t = \mathtt{f}(t_1, \ldots, t_k)$ und $s = \mathtt{f}(s_1, \ldots, s_k)$ für ein Atom $\mathtt{f} \in \mathsf{A}_{\mathsf{LP}}$ und eine Stelligkeit $k \in \mathbb{N}$ mit $k \geqslant 1$
9. Setze $S_1 := I$.
10. Für $i = 1, \ldots, k$ tue Folgendes:
11. Setze $T_i := \mathrm{MGU}(t_i S_i, s_i S_i)$.
12. Wenn $T_i =$ „nicht unifizierbar" dann gib „nicht unifizierbar" aus und halte an.
13. Setze $S_{i+1} := S_i T_i$.
14. Gib S_{k+1} aus und halte an.
15. Gib „nicht unifizierbar" aus und halte an.

Korrektheit des Unifikationsalgorithmus

Satz 5.28

Für alle Terme $t, s \in \mathsf{T}_{\mathsf{LP}}$ gilt:

(a) *Sind t und s unifizierbar, so gibt $\mathrm{MGU}(t, s)$ einen allgemeinsten Unifikator für t und s aus.*

(b) *Sind t und s nicht unifizierbar, so gibt $\mathrm{MGU}(t, s)$ die Worte „nicht unifizierbar" aus.*

(Hier ohne Beweis)

Korollar 5.29

Sind zwei Terme unifizierbar, so gibt es für diese Terme einen allgemeinsten Unifikator.

Beispiele 5.30

(a) Ein allgemeinster Unifikator für

$$t \;:=\; \texttt{g(f(X,Y),f(V,W))} \qquad \text{und} \qquad s \;:=\; \texttt{g(V,f(Z,g(X,Y)))}$$

ist

$$\begin{aligned} S \;&:=\; \{\, \texttt{V} \mapsto \texttt{f(X,Y)},\;\; \texttt{Z} \mapsto \texttt{f(X,Y)},\;\; \texttt{W} \mapsto \texttt{g(X,Y)} \,\} \\ &=\; \{\texttt{V} \mapsto \texttt{f(X,Y)}\}\,\{\texttt{Z} \mapsto \texttt{f(X,Y)}\}\,\{\texttt{W} \mapsto \texttt{g(X,Y)}\}, \end{aligned}$$

und es gilt $tS = sS = \texttt{g(f(X,Y),f(f(X,Y),g(X,Y)))}$.

(b) $\texttt{g(f(X,Y),Y)}$ und $\texttt{g(c,Y)}$ sind nicht unifizierbar.

(c) Seien $n \geqslant 1$ und seien $X_0, \ldots, X_n \in \mathrm{V}_{\mathrm{LP}}$ paarweise verschieden. Sei

$$\begin{aligned} t_n \;&:=\; \texttt{f(}X_1\texttt{,}X_2\texttt{...,}X_n\texttt{)} \\ s_n \;&:=\; \texttt{f(g(}X_0\texttt{,}X_0\texttt{),g(}X_1\texttt{,}X_1\texttt{),...,g(}X_{n-1}\texttt{,}X_{n-1}\texttt{))}\,. \end{aligned}$$

Dann sind t_n und s_n unifizierbar durch einen allgemeinsten Unifikator S, für den gilt: – siehe Tafel –

Es gilt: Für jeden Unifikator T für t_n und s_n ist der Term $T(X_n)$ exponentiell groß in n, und jede gemeinsame Instanz von t_n und s_n ist exponentiell lang in n.

Auflösen des Nichtdeterminismus in Zeile 4

Wir können nun den Nichtdeterminismus in Zeile 4 unseres einfachen Interpreters für Logikprogramme, $\text{ANTWORT}(\Pi, \alpha)$, auflösen, indem wir als Substitution T einen allgemeinsten Unifikator von α_i und φU wählen, und zwar den allgemeinsten Unifikator, der vom Algorithmus $\text{MGU}(\alpha_i, \varphi U)$ ausgegeben wird.

Dadurch erhalten wir den folgenden Algorithmus $\text{UANTWORT}(\Pi, \alpha)$.

Interpreter für Logikprogramme mit allgemeinsten Unifikatoren

Algorithmus $\textsc{UAntwort}(\Pi, \alpha)$

% Eingabe: Programm $\Pi \in \mathsf{LP}$, Anfrage $\texttt{?-}\ \alpha \in \mathsf{F}_{\mathsf{LP}}$ mit $\alpha = \alpha_1, \ldots, \alpha_m$

% Ausgabe: eine Substitution $\tilde{S}$ für $\mathrm{Var}(\alpha)$ oder das Wort „gescheitert“.

1. Wähle ein $i \in [m]$ *% α_i ist das nächste „Ziel“*
2. Wähle eine Regel ρ aus Π. Sei $\varphi\ \texttt{:-}\ \psi_1, \ldots, \psi_n$ die Form von ρ.
   % Fakten fassen wir als Regeln ohne Rumpf auf
3. Sei U eine Umbennung für $\mathrm{Var}(\rho)$, so dass $\mathrm{Var}(\rho U) \cap \mathrm{Var}(\alpha) = \emptyset$.
4. Setze $\tilde{T} := \mathrm{MGU}(\alpha_i, \varphi U)$
   % $\tilde{T}$ soll ein allgemeinster Unifikator von α_i und φU sein
5. Wenn $\tilde{T} =$ „nicht unifizierbar“, gib „gescheitert“ aus und halte an.
6. Wenn $m = 1$ und $n = 0$, gib $\tilde{T}|_{\mathrm{Var}(\alpha)}$ aus und halte an.
7. Setze $\tilde{\alpha}' := \alpha_1\tilde{T}, \ldots, \alpha_{i-1}\tilde{T}, \psi_1 U\tilde{T}, \ldots, \psi_n U\tilde{T}, \alpha_{i+1}\tilde{T}, \ldots, \alpha_m\tilde{T}$.
8. Setze $\tilde{T}' := \textsc{UAntwort}(\Pi, \tilde{\alpha}')$
9. Wenn $\tilde{T}'$ eine Substitution ist, gib $(\tilde{T}\tilde{T}')|_{\mathrm{Var}(\alpha)}$ aus und halte an.
10. Gib „gescheitert“ aus und halte an.

Korrektheit und Vollständigkeit des Interpreters

Satz 5.31

Sei $\Pi \in \mathsf{LP}$ *ein Logikprogramm, sei* ?- α $\in \mathsf{F}_{\mathsf{LP}}$ *eine Anfrage mit* $\alpha = \alpha_1, \ldots, \alpha_m$*, und sei* S *eine Substitution für* $\mathrm{Var}(\alpha)$*. Dann sind folgende Aussagen äquivalent:*

(a) *Die Terme* $\alpha_1 S, \ldots, \alpha_m S$ *sind aus* Π *ableitbar.*

(b) *Es gibt einen Lauf von* $\textsc{UAntwort}(\Pi, \alpha)$*, der eine Substitution* $\tilde{S}$ *für* $\mathrm{Var}(\alpha)$ *mit* $\tilde{S} \leqq S$ *ausgibt.*

Korollar 5.32

Sei $\Pi \in \mathsf{LP}$ *ein Logikprogramm und sei* α *ein Grundterm. Dann gilt:*
$\alpha \in \mathcal{B}(\Pi) \iff$ *es gibt einen akzeptierenden Lauf von* $\textsc{UAntwort}(\Pi, \alpha)$*.*

Für den Beweis der Richtung „*(a)* $\Longrightarrow$ *(b)*“ von Satz 5.31 verwenden wir:

Lemma 5.33

Sei $\Pi \in \mathsf{LP}$ *und sei* ?- α $\in \mathsf{F}_{\mathsf{LP}}$ *mit* $\alpha = \alpha_1, \ldots, \alpha_m \in \mathsf{F}_{\mathsf{LP}}$*, und sei* S' *eine Substitution für* α*. Dann gibt es zu jedem Lauf von* $\textsc{Antwort}(\Pi, \alpha S')$*, der eine Substitution* S *ausgibt, einen Lauf von* $\textsc{UAntwort}(\Pi, \alpha)$*, der eine Substitution* $\tilde{S}$ *mit* $\tilde{S} \leqq S'S$ *ausgibt.*

Bemerkungen

- Indem wir das nichtdeterministische Auswählen einer Substitution im Algorithmus Antwort im Algorithmus UAntwort durch das deterministische Berechnen eines allgemeinsten Unifikators ersetzt haben, sind wir einen entscheidenden Schritt in Richtung „praktische Ausführbarkeit“ gegangen.

- Es bleiben aber immer noch die nichtdeterministischen Auswahlschritte eines Ziels in Zeile 1 und einer Regel in Zeile 2. Diese müssen bei einer praktischen Implementierung durch eine systematische Suche durch alle Möglichkeiten ersetzt werden.

  (Die Wahl der Umbennenung in Zeile 3 unproblematisch.)

- Verschiedene logische Programmiersprachen unterscheiden sich in den verwendeten Suchstrategien.

- Prolog verwendet Tiefensuche.

Abschnitt 5.4:

Logik-Programmierung und Prolog

Reines Prolog

Reines Prolog ist das Fragment der Programmiersprache Prolog, dessen Programme gerade die Logikprogramme in LP sind.

Insbesondere enthält reines Prolog keine speziellen Prolog-Operatoren wie Cut „!“, arithmetische Prädikate oder Ein-/Ausgabe-Prädikate (d.h. Prädikate mit Seiteneffekten).

Die Semantik von reinem Prolog stimmt nicht mit der deklarativen Semantik der Logik-Programmierung überein.

Die erste vom Prolog-Interpreter ausgegebene Antwort wird gemäß dem folgenden Interpreter $\textsc{PErsteAntwort}$ ermittelt.

Ein Prolog-Interpreter

Algorithmus $\textsc{PErsteAntwort}(\Pi, \alpha)$

% Eingabe: Programm $\Pi \in \mathsf{LP}$*, Anfrage* `?-` $\alpha \ \in \ \mathsf{F}_{\mathsf{LP}}$ *mit* $\alpha = \alpha_1, \ldots, \alpha_m$

% Ausgabe: eine Substitution S *für* $\mathrm{Var}(\alpha)$ *oder das Wort „false“*

1. Betrachte alle Regeln ρ in Π in der Reihenfolge ihres Vorkommens in Π und tue Folgendes: *% Fakten fassen wir als Regeln ohne Rumpf auf*
2. Sei φ `:-` $\psi_1, \ldots, \psi_n$ die Form von ρ
3. Sei U eine Umbennung für $\mathrm{Var}(\rho)$, so dass $\mathrm{Var}(\rho U) \cap \mathrm{Var}(\alpha) = \emptyset$
4. Setze $T := \mathrm{MGU}(\alpha_1, \varphi U)$
5. Wenn T eine Substitution ist
6. Wenn $m = 1$ und $n = 0$, gib $T|_{\mathrm{Var}(\alpha)}$ aus und halte an
7. Setze $\alpha' := \psi_1 UT, \ldots, \psi_n UT, \alpha_2 T, \ldots, \alpha_m T$
8. Setze $T' := \textsc{PErsteAntwort}(\Pi, \alpha')$
9. Wenn T' eine Substitution ist, gib $(TT')|_{\mathrm{Var}(\alpha)}$ aus und halte an
10. Gib „false“ aus und halte an

Vergleich zur deklarativen Semantik

$\textsc{PErsteAntwort}(\Pi, \alpha)$ gibt *höchstens eine* Substitution aus, kann u.U. aber auch in eine Endlosschleife gelangen und nicht terminieren.

Der folgende Satz besagt, dass im Falle der Terminierung die ausgegebene Antwort korrekt ist.

Satz 5.34

Sei $\Pi \in \mathsf{LP}$ ein Logikprogramm und sei ?- $\alpha \in \mathsf{F}_{\mathsf{LP}}$ *mit $\alpha = \alpha_1, \ldots, \alpha_m$ eine Anfrage. Dann gilt:*

(a) *Wenn $\textsc{PErsteAntwort}(\Pi, \alpha)$ eine Substitution S ausgibt, dann sind die Terme $\alpha_1 S, \ldots, \alpha_m S$ aus Π ableitbar.*

(b) *Wenn $\textsc{PErsteAntwort}(\Pi, \alpha)$ das Wort „false" ausgibt, dann gibt es keine Substitution S, so dass die Terme $\alpha_1 S, \ldots, \alpha_m S$ aus Π ableitbar sind.*

(Hier ohne Beweis)

Terminierung

Intuitiv besagt Satz 5.34, dass im Falle der Terminierung die vom Prolog-Interpreter bei Eingabe eines Logikprogramms Π und einer Anfrage `?-` α gegebene erste Antwort korrekt ist.

Möglicherweise hält der Prolog-Interpreter aber gar nicht an, obwohl es laut Definition der deklarativen Semantik korrekte Antworten gibt.

Es ist Aufgabe des Programmierers, dies zu verhindern!

Typische Probleme dabei sind Dummheit und linksrekursive Regeln.

Beispiel: `vorfahre(X,Y) :- vorfahre(X,Z), elternteil(Z,Y)`

Unterschied zwischen Theorie und Praxis

Beispiel 5.35

Die folgenden Logikprogramme `myplus1.pl`, `myplus2.pl`, `myplus3.pl` haben die gleiche Bedeutung hinsichtlich der deklarativen Semantik im folgenden Sinne:

Aus allen drei Programmen können genau dieselben Grundterme der Form `myplus(...)` abgeleitet werden.

Alle drei Programme erzeugen jedoch unterschiedliche Ausgaben in Prolog.

Programm: `myplus1.pl`

```
myplus(X,Y,Z) :- myplus(Y,X,Z).
myplus(0,X,X).
                myplus(1,1,2).  myplus(1,2,3).  myplus(1,3,4).
                                myplus(2,2,4).  myplus(2,3,5).
                                                myplus(3,3,6).
```

Programm: `myplus2.pl`

```
myplus(0,X,X).
                myplus(1,1,2).  myplus(1,2,3).  myplus(1,3,4).
                                myplus(2,2,4).  myplus(2,3,5).
                                                myplus(3,3,6).
myplus(X,Y,Z) :- myplus(Y,X,Z).
```

Programm: `myplus3.pl`

```
myplusH(0,X,X).
                myplusH(1,1,2).  myplusH(1,2,3).  myplusH(1,3,4).
                                 myplusH(2,2,4).  myplusH(2,3,5).
                                                  myplusH(3,3,6).
myplus(X,Y,Z) :- myplusH(X,Y,Z).
myplus(X,Y,Z) :- myplusH(Y,X,Z).
```

Aus Sicht des Prolog-Interpreters (und des Interpreters PERSTEANTWORT) ist das Programm `myplus1.pl` idiotisch und liefert auf keine Anfrage der Form „`myplus(...)`“ eine Antwort, da die Auswertung des Programms stets mit der ersten Regel in eine Endlosschleife gerät.

Das Programm `myplus2.pl` ist besser, hält aber auch bei „falschen“ Anfragen wie z.B. „`myplus(1,1,3)`“ nicht an, da die Auswertung des Programms dann mit der letzten Regel in eine Endlosschleife gerät.

Das Programm `myplus3.pl` leistet das, was es soll.

Beweisbäume vs. Suchbäume

Beweisbäume

sind in Definition 5.14 definiert. Ein Beweisbaum ist eine graphische Darstellung einer Ableitung eines Terms $t \in \mathsf{T}_{\mathsf{LP}}$ aus einem Logikprogramm $\Pi \in \mathsf{LP}$.

Somit stellt ein Beweisbaum eine einzelne Ableitung dar. Diese entspricht einem erfolgreichen Lauf unseres nichtdeterministischen Interpreters $\textsc{Antwort}$.

Suchbäume

stellen die vollständige Suche des Prolog-Interpreters bei Eingabe eines Logikprogramms Π und einer Anfrage ?- α dar. Insbesondere enthält der Suchbaum Informationen über <u>alle</u> erfolgreichen Läufe des nichtdeterministischen Interpreters $\textsc{Antwort}$.

Unifikation in Prolog

In Prolog testet der Ausdruck $t = s$ testet nicht, ob die Terme t und s gleich sind, sondern ob sie unifizierbar sind.

Der in den meisten Prologimplementierungen verwendete Unifikationsalgorithmus testet aus Effizienzgründen bei der Unifikation einer Variablen X mit einem Term t nicht, ob X in t vorkommt.

Diesen Test bezeichnet man als Occurs-Check, er findet in den Zeilen 3 und 6 unseres Unifikationsalgorithmus MGU statt.

In Prolog ist es eine Aufgabe des Programmierers, sicherzustellen, dass niemals eine Variable mit einem Term unifiziert wird, der diese Variable enthält.