Chair of Embedded Systems
Department of Computer Science
University of Augsburg, Germany

Martin Frieb

Universität Augsburg University

Wintersemester 2020/2021

Informatik 1

Kapitel 10 – Mathematische Konzepte in der Informatik

Inhaltsverzeichnis

10.1 Mathematik in der Informatik

Wozu braucht man Mathematik in der Informatik?
Die Informatik setzt sich aus vielen Gebieten zusammen, die Antworten auf verschiedene Fragen suchen, z.B.:

- **Rechnerarchitektur**: Wie kann man Rechner bauen?
- **Programmiersprachen und Compilerbau**: Wie kann man Rechner programmieren?
- **Software Engineering**: Wie kann man große und sehr komplexe Programme erstellen?
- **Künstliche Intelligenz**: Wie kann man Programme zu Problemen erstellen, die wir nicht gut verstehen?
- **Soziale Informatik**: Was muss man beachten, damit Rechner und Programme im Dienste des Menschen eingesetzt werden?
- **Sicherheitskritische Systeme**: Wie erstellt man Systeme aus Hard- und Software, von deren fehlerfreier Funktion Menschenleben abhängen?
- **Algorithmen und Datenstrukturen**: Wie erstellt und analysiert man besonders schnelle und speichereffiziente Programme?

Das Ziel der Informatik ist, die in den verschiedenen Gebieten der Informatik betrachteten Fragestellungen und deren zugrundeliegenden Gegebenheiten der realen Welt so präzise zu modellieren, dass man mathematisch gesicherte Aussagen darüber treffen kann und Rechner selbständig damit umgehen können.

Ein wichtiges Hilfsmittel dabei ist die Abstraktion, also das Weglassen von Details, die für die jeweils betrachtete Fragestellung unwichtig sind.

Zusammengefasst gründet die Informatik darauf, zu einer betrachteten Fragestellung ein passendes mathematisches Modell zu finden, mit dessen Hilfe die Fragestellung analysiert und rechnergestützt gelöst werden kann

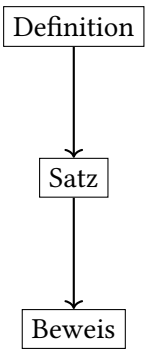
Mathematik ist also die Grundlage der Informatik.

Aufgaben eines Informatikers:

- Zu einer Fragestellung ein passendes bereits existierendes mathematisches Modell finden und anpassen, oder ein neues entwickeln
- Mathematische Modelle analysieren
- Mathematische Modelle implementieren

In Informatik-Vorlesungen lernen Sie dazu neben Programmiertechniken die für die Informatik grundlegenden mathematischen Strukturen und Methoden kennen und üben logisches Schließen und präzises und verständliches Formulieren

Vorgehen in der Mathematik:



Anwendung in der Informatik:

- **Definiere** neue mathematische Objekte zur Modellierung einer praktischen Fragestellung
- Ein **mathematischer Satz** formuliert nützliche Aussagen über die definierten Objekte
- Ein **mathematischer Beweis** ist eine logische Herleitung der Richtigkeit (oder Unrichtigkeit) einer Aussage aus Definitionen und anderen (bereits bewiesenen) Aussagen

10.2 Terme

10.2.1 Operationen

Definition: 10.1 Operation

Eine n - **stellige Operation** ist eine Abbildung

$$op : T_1 \times \ldots \times T_n \to T.$$

- Sie ordnet jedem Element $(t_1, \ldots, t_n) \in T_1 \times \ldots \times T_n$ durch eine **Rechenvorschrift** einen **Ergebniswert** $op(t_1, \ldots, t_n) \in T$ zu.
- n ist die **Stelligkeit** von op
- $t_1, \ldots, t_n$ heißen **Operanden** oder **Argumente**. Die Stelligkeit ist die Anzahl der Operanden.
- $T_1, ..., T_n$ heißen **Parametertypen** von op.
- T ist der **Resultattyp** von op.

Operationen sollten schon aus der Mathematik bekannt sein, eventuell jedoch nicht unter diesem Namen. Das Additionssymbol '$+$' beispielsweise ist eine 2-stellige Operation.
Es werden also ein- oder mehrere Elemente zu einem einzigen Element abgebildet.

Eine Operation besitzt bei uns folgende Notation: $op(t_1, \ldots, t_n)$. Also: Operationszeichen und anschließend in Klammern die benötigten Elemente.

Beispiel:

- $4 + 3$ wird zu: $+(4, 3)$
- $4 * 3$ wird zu: $*(4, 3)$
- $4/3$ wird zu: $/(4, 3)$

- -3 wird zu: $-(3)$
- $(4 \cdot 8)/10$ wird zu $/(\cdot(4,8),10))$

Beispiel:

Wie schon angesprochen ist die Addition $+$ eine 2-stellige Operation. Damit können zwei beliebige Werte aus einer beliebigen Zahlenmenge $\mathbb{L} \in \{\mathbb{N}, \mathbb{Q}, \mathbb{Z}, \mathbb{R}, \mathbb{C}\}$ auf die selbe Menge abgebildet werden.

$$+ : \mathbb{L} \times \mathbb{L} \to \mathbb{L}$$

- Sie ordnet jedem Element $(a, b) \in \mathbb{L} \times \mathbb{L}$ durch eine Rechenvorschrift einen Ergebniswert $+(a, b) \in \mathbb{L}$ zu (Es gilt: $a \in \mathbb{L}$ und $b \in \mathbb{L}$).
- $+$ besitzt eine Stelligkeit von 2 (da 2 Elemente für die Addition benötigt werden)
- a, b heißen Operanden oder Argumente.
- $\mathbb{L}$ sowie $\mathbb{L}$ sind Parametertypen von $+$.
- $\mathbb{L}$ ist der Resultattyp von $+$.

Häufig arbeiten wir mit 1- und 2-stelligen Operationen. Diese nennt man auch *unäre* sowie *binäre* Operationen.

Beispiel:

Unäre Operationen: 1-stellige Operationen heißen **unär**, z.B.:

- Negatives Vorzeichen: $- : \mathbb{R} \to \mathbb{R}$
- Positives Vorzeichen: $+ : \mathbb{R} \to \mathbb{R}$
- Abrundung: $\lfloor \cdot \rfloor : \mathbb{R} \to \mathbb{Z}$
- Aufrundung: $\lceil \cdot \rceil : \mathbb{R} \to \mathbb{Z}$
- Absolutbetrag: $|\cdot| : \mathbb{R} \to [0, \infty[$
- Wortlänge: $|\cdot| : A^* \to \mathbb{N}_0$
- Mächtigkeit von Mengen: $|\cdot| : P(A) \to \mathbb{N}_0$

Beispiel:

Binäre Operationen: 2-stellige Operationen heißen **binär**, z.B.:

- Addition: $+ : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$
- Subtraktion: $- : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$
- Multiplikation: $\cdot : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$
- Division: $/ : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$
- Modulo: $\text{mod} : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$
- Ganzzahlige Division: $\div : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$
- Mengenvereinigung: $\cup : P(A) \times P(A) \to P(A)$
- Mengendurchschnitt: $\cap : P(A) \times P(A) \to P(A)$

- Mengendifferenz: $\setminus : P(A) \times P(A) \to P(A)$

10.2.2 Terme

Terme sollten, ähnlich wie Operationen, schon aus der Mathematik bekannt sein. Darunter können unter anderem mathematische Gebilde verstanden werden, mit denen etwas ausgerechnet werden kann (bspw. ist $(x+y)^2$ ein Term). Wir können unter einem Term aber auch logische Operationen verstehen, die einen Wahrheitswert generieren (bspw. $p_1 \wedge p_2$).

Terme: Grundelemente
Wir wollen nun formal mathematisch **Terme** zur Darstellung von Rechnungen definieren.

Bevor wir jedoch einen Term definieren (oder beschreiben) können, müssen wir noch einige Begriffe definieren. Dazu geben wir uns eine Menge von **Operationen**, eine Menge von **Funktionen** und eine Menge von **Variablen** vor:

- Ein Element eines Parameter- oder Resultattyps T nennen wir **Konstante**. Wir sagen, eine Konstante $t \in T$ ist **vom Typ** T.
- Eine Variable hat einen Typ T und dient als Platzhalter für Konstanten vom Typ T.

Anmerkung:
Das Vorgehen, welches gerade beschrieben wird, entspricht dem typischen Vorgehen in der Mathematik. Es werden einige Begriffe und mathematische Objekte als gegeben vorausgesetzt. Darauf basierend konstruiert (definiert) man neue Begriffe und mathematische Objekte. Auf diese Weise erstellt man Schritt für Schritt ein immer komplexeres mathematisches Modell.

In unserem Fall setzen wir Operationen, Funktionen und Variablen voraus. Darauf basierend definieren wir uns Konstanten.

Terme: Bildungsregeln

Definition: 10.2 Term

1. Jede **Variable** oder **Konstante** eines Typs T ist ein **Term vom Typ** T.
2. Ist $op : T_1 \times \ldots \times T_n \to T$ eine Operation und A_i jeweils ein Term vom Typ T_i für $i \in \{1, ..., n\}$, so ist
$$op(A_1, \ldots, A_n)$$
ein **Term vom Typ** T.
3. Ist $f : S \to T$ eine Funktion und A ein Term vom Typ S, so ist
$$f(A)$$
ein **Term vom Typ** T.

Beispiel:

Eine lineare Funktion $f(x) = m \cdot x + b$ kann beispielsweise in zwei Terme aufgeteilt werden:

- Einen linearen Term $m \cdot x$ (dieser besteht wiederum aus der Variable x und Konstante m)
- Einen konstanten Term mit der Konstante b

Beispiel:

Mit obiger Definition können wir unter Benutzung der eingeführten Funktionen und arithmetischen Rechenoperationen der Reihe nach Terme bilden. Dafür bauen wir die Terme nacheinander auf.

Das Ziel wird sein, den folgenden Term zu bekommen: $\div(+(-(x), 5), \log(\cdot(y, y)))$
x und y sind hier immer Variablen vom Typ $\mathbb{Z}$

1. Wir müssen zuerst auf den Term $+(-(x), 5)$ kommen:
   1.1. Wir definieren uns eine Variable x sowie die Konstante 5
   1.2. Auf x wird der unäre $-$ Operator angewendet: $-(x)$
   1.3. Wir kombinieren die Konstante und den Term aus Schritt 1.2. mit dem binären $+$ Operator: $+(-(x), 5)$
2. Nun wollen wir uns den Term $\log(\cdot(y, y))$ definieren:
   2.1. Wir beginnen mit der Variable y
   2.2. Auf diese Variable können wir den binären Multiplikationsoperator anwenden: $\cdot(y, y)$
   2.3. Abschließend wird nun der Logarithmus angewendet: $\log(\cdot(y, y))$
3. Mithilfe der Terme aus den Schritten 1.3. und 2.3. können wir nun den gesuchten Term definieren: $\div(+(-(x), 5), \log(\cdot(y, y)))$

Strukturelle Induktion
Terme bzw. die **Menge der Terme** werden per sogenannter struktureller Induktion definiert:
Zuerst werden **einfache Grundelemente** der Menge definiert. Also alle benötigten Variablen oder Konstanten eines Typs T (Unterpunkt 1 aus der Definition).
Dann werden **endliche viele Bildungsregeln** angegeben, mit denen aus existierenden Elementen der Menge zusätzliche Elemente konstruiert werden können (Unterpunkte 2 und 3 aus der Definition).

Solche Bildungsregeln dienen als Regel zur Konstruktion eines neuen Elements aus bereits vorhandenen Elementen. Es dürfen aber nur solche Elemente genutzt werden, die sich aus Grundelementen durch (ggf. wiederholte) Anwendung der Bildungsregeln konstruieren lassen.

Details zur strukturellen Induktion folgen im Kapitel 10.3.

Teilterme

Definition: 10.4 Teilterm

- Jeder Term A ist ein **Teilterm** von sich selbst.
- Ist $\mathrm{op}(A_1, \ldots, A_n)$ ein Term, so sind $A_1, ..., A_n$ und Teilterme von $A_1, ..., A_n$ **Teilterme** von $\mathrm{op}(A_1, \ldots, A_n)$
- Ist $f(A)$ ein Term, so sind A und Teilterme von A **Teilterme** von $f(A)$

Beispiel:

Der Term $+(-(x), 5)$ hat die folgenden Teilterme:

- $+(-(x), 5)$
- $-(x)$
- x
- 5

10.2.3 Notationsarten

Mathematische Ausdrücke besitzen verschiedene Notationen. Sie können, wie bereits aus der Schule bekannt ist, folgendermaßen geschrieben werden: Operation und anschließend Wert bzw. Wert-Operation-Wert (bspw: -5, $2 + 3$).
Aber es gibt auch eine andere Schreibweise, bei der die Operationen **immer** vorne stehen (bspw: $(-\ 5), (+\ 2\ \ 3)$)

Infixnotation
Unter der **Infixnotation** versteht man die bereits aus der Schule bekannte Schreibweise *Wert-Operation-Wert*. Es ist auch die übliche Schreibweise in mathematischen Formeln sowie im C-Code. Das Operationszeichen kommt immer **zwischen** den Operanden bzw. davor bei unären Operationen.

Formaler beschrieben schreibt man in einem Ausdruck in Infixnotation

- $(A\ \mathrm{op}\ B)$ für jede binäre Operation op.
- $(\mathrm{op}\ A)$ für jede unäre Operation op.

Beispiel:

- (-5) statt $-(5)$
- $(2 \cdot (3 + 5))$ statt $\cdot(2, +(3, 5))$
- $((2 \cdot 3) + 5)$ statt $+(\cdot(2, 3), 5)$

Präfixnotation

Die **Präfixnotation** (oder auch polnische Notation genannt), ist eine komplett klammerfreie Schreibweise. Sie wird überwiegend in der Logik und in einigen Programmiersprachen (bspw. LISP) verwendet. Das Operationszeichen kommt hier **immer vor** dem Operanden.

Formaler beschrieben schreibt man in einem Ausdruck in Präfixnotation

- $(\mathrm{op}\ \ A_1\ \ A_2\ \ \ldots\ \ A_n)$ statt $\mathrm{op}(A_1, \ldots, A_n)$
- $(f\ \ A)$ statt $f(A)$

> Beispiel:
>
> - $-(5)$ schreibt man in der Präfixnotation $(-\ \ 5)$ (negatives Vorzeichen)
> - $-(5,3)$ schreibt man in der Präfixnotation $(-\ \ 5\ \ 3)$ (Subtraktion)
> - $\cdot(-(5,6),7)$ schreibt man in der Präfixnotation $(\cdot\ \ -\ \ 5\ \ 6\ \ 7)$
>   (bekannte Schreibweise aus der Schule: $(5-6)\cdot 7$)
> - $-(5, \cdot(6,7))$ schreibt man in der Präfixnotation $(-\ \ 5\ \ \cdot\ \ 6\ \ 7)$
>   (bekannte Schreibweise aus der Schule: $5-(6\cdot 7)$)
>
> Wie wir bereits sehen können, unterscheidet sich die Präfixnotation nicht groß zur Schreibweise der Operationen. Für uns ist der einzige Unterschied das Weglassen der Klammern.

Wir verwenden diese Schreibweise nicht weiter, sollte aber einmal angesprochen worden sein.

Auswertung von Termen

Bei der Ausführung eines Programms haben die Variablen zu einem bestimmten Zeitpunkt einen konkreten Wert, so dass sich damit der Wert eines Ausdrucks ausrechnen lässt. Ebenso verhält es sich bei den Termen: Der Wert eines Terms hängt davon ab, welchen Wert die Variablen annehmen.

Wenn wir einen Term A gegeben haben und einen konkreten Wert für diesen Term ausrechnen wollen, legen wir eine konkrete **Variablen-Belegung** fest, die wir mit β bezeichnen. Dabei handelt es sich um eine Abbildung, die jeder Variable einen konkreten Wert (also eine Konstante) zuweist (z.B. $\beta := (x \to 5, y \to 2, ...)$). Um den konkreten Wert zu bezeichnen, den der Term A bezüglich dieser Belegung β annimmt, schreiben wir A'. Sofern ein Term aus Operationen bzw. Funktionen besteht, so ist die Belegung erst auf deren Argumenten anzuwenden und danach sind die Operationen bzw. Funktionen auszuführen. Terme werden also von **innen nach außen** ausgewertet. Die Reihenfolge der Auswertung der Operanden einer Operation ist dabei nicht definiert.

> Beispiel:
>
> Gegeben sei folgender Term:
> $A := +(-(\cdot(5,x), /(y,2)), -(z))$
> Nun wollen wir den konkreten Wert des Terms A bzgl. einer Belegung β ausrechnen.
> Diese legen wir fest als $\beta := (x \to 3, y \to 8, z \to 4)$.
> Damit ergibt sich:

$$A' := +(-(\cdot(5,3),/(8,2)),-(4)) = +(-(15,4),-(4)) = +(11,-4) = 7$$

Formal aufgeschrieben funktioniert die Auswertung von Termen also wie folgt:

Definition: 10.8 Auswertung von Termen

Sei β eine Variablen-Belegung. Wird definieren wie folgt induktiv den Wert A' eines Terms A bzgl. β:

- Der Wert k' einer Konstante k ist $k' := k$.
- Der Wert x' einer Variable x ist $x' := \beta(x)$.
- Der Wert $op(A_1, \ldots, A_n)'$ eines Terms $op(A_1, \ldots, A_n)$ ist $op(A_1, \ldots, A_n)' := op(A_1', \ldots, A_n')$
- Der Wert $f(A)'$ eines Terms $f(A)$ ist $f(A)' := f(A')$

10.3 Strukturelle Induktion

Aus Mathematik für Informatiker sollte schon die vollständige Induktion bekannt sein. Hierbei wird für eine ab $n \in \mathbb{N}_0$ beginnende Zahlenfolge bewiesen, dass eine Formel o.ä. gilt. Die vollständige Induktion folgt dabei immer dem gleichen Prinzip: Im *Induktionsanfang* zeigt man, dass die zu beweisende Aussage für einen Startwert gilt. Danach zeigt man im *Induktionsschritt* unter Verwendung der *Induktionsvoraussetzung*, dass die Aussage für jeden nächsten Wert gilt (i.d.R. $n \to n + 1$).

Bei der *strukturellen Induktion* befassen wir uns jedoch nicht mehr ausschließlich mit Zahlenfolgen, sondern unter anderem mit Mengen und Termen. Ihr liegt dabei im Induktionsschritt ein anderes Grundprinzip zu Grunde: Man prüft, ob die zu beweisende Aussage nach der Anwendung weiterer (Aufbau-)Regeln gilt, mit denen die jeweiligen Mengen/Terme/... gebildet werden.

Ein Beweisprinzip für induktiv definierte Mengen

Sei M eine induktiv definierte Menge und E eine Eigenschaft, für die wir beweisen wollen, dass sie für alle Elemente von M erfüllt ist.

Allgemeines Schema der **strukturellen Induktion**:

1. **Induktionsanfang**:
   Zu zeigen: E gilt für alle Grundelemente von M
2. **Induktionsschritt**:
   Zu zeigen: Angenommen E gilt die Elemente $m_1, \ldots, m_n$ (**Induktionsvoraussetzung**), so gilt E auch für jedes Element, das aus $m_1, \ldots, m_n$ mit einer der Bildungsregeln von M konstruiert wird.

Die strukturelle Induktion stützt sich dabei immer auf Bildungsregeln. Wir gehen diese Regeln durch und prüfen für jede Regel, ob nach Anwendung dieser Regel die zu beweisende Aufgabe weiterhin erfüllt ist.

Beispiel: Strukturelle Induktion über einer Menge

Sei M die wie folgt definierte Menge:

1. 3 ist ein Element von M (d.h. das einzige Grundelement ist 3)
2. Ist x ein Element von M, dann ist auch $x + 3$ ein Element von M (Bildungsregel für neue Elemente)

Beispiel:

Die oben definierte Menge mag anfangs abschrecken, ist jedoch nicht besonders kompliziert:

1. Anfangs besitzen wir in der Menge: $\{3\}$
2. Es befindet sich $\{3\}$ in der Menge. Durch die Bildungsregel befindet sich auch $\{3, 3+3\} = \{3, 6\}$ in der Menge
3. Es befindet sich $\{3, 6\}$ in der Menge. Durch die Bildungsregel befindet sich auch $\{3, 6, 3+3, 6+3\} = \{3, 6, 9\}$ in der Menge
4. Es befindet sich $\{3, 6, 9\}$ in der Menge. Durch die Bildungsregel befindet sich auch $\{3, 6, 9, 3+3, 6+3, 9+3\} = \{3, 6, 9, 12\}$ in der Menge
5. ...

Wir wollen die Eigenschaft beweisen: **Jedes Element von M ist durch 3 teilbar**

Beweis mit struktureller Induktion:

1. **Induktionsanfang**:
   Im Induktionsanfang müssen wir für jedes Grundelement aus unserer Menge zeigen, dass die Eigenschaft gilt. Da anfänglich die 3 das einzige Element in unserer Menge ist, müssen wir nun Folgendes tun:
   Zu zeigen: 3 *ist durch* 3 *teilbar*
   Beweis: $(3 \bmod 3) = 0$
2. **Induktionsschritt**:
   Im Induktionsschritt müssen wir für alle Bildungsregeln beweisen, dass die geforderte Eigenschaft gilt.
   Zu zeigen: Angenommen x *ist durch* 3 *teilbar (aufgrund der* ***Induktionsvoraussetzung*** *können wir das annehmen), dann ist auch* $x + 3$ *durch* 3 *teilbar*
   Beweis:
   $$\begin{aligned}(x+3) \bmod 3 &= ((x \bmod 3) + (3 \bmod 3)) \bmod 3 \\ &= (0 + 0) \bmod 3 \\ &= 0\end{aligned}$$

Nachdem wir im Induktionsanfang gezeigt haben, dass wir annehmen können, dass die Induktionsvoraussetzung gilt (Beweis der Aussage für die Grundmenge), können wir das Ergebnis im Induktionsschritt verwenden. Durch den Beweis des Induktionsschritts haben wir auch die Eigenschaft gezeigt, dass alle Elemente von M durch 3 teilbar sind.

Beispiel: Strukturelle Induktion mit Nudeln

Die strukturelle Induktion kann man auch auf ein Nudelgericht anwenden. Hierbei haben wir das folgende Rezept definiert:

(A) Eine Nudel (10g) hinzufügen
(B) Einen Klecks Soße (15g) hinzufügen
(C) Einen Löffel geriebenen Käse (5g) hinzufügen

Damit auch von einem Nudelgericht die Rede sein kann, fordern wir, dass Regel (A) mindestens einmal angewendet wird. Ansonsten können wir diesem Gericht beliebig viele Nudeln, Soße oder Käse hinzufügen. Aber immer nur in den Mengenschritten, die oben definiert wurden.

Beispiel:

Ein Beispielgericht könnte folgendermaßen aussehen:

- 10g Nudeln
- 15g Soße
- 5g Käse

Oder auch so:

- 100g Nudeln
- 30g Soße
- 2000g Käse

Beweisen Sie:
Das Gewicht des Nudelgerichts ist immer ein Vielfaches von 5g.

Die Notation, die wir für das Gewicht des Nudelgerichts verwenden, ist der Einfachheit halber das Betragssymbol:

$$\text{Gewicht-Nudelgericht} = |\text{Nudelgericht}|$$

Somit ergibt sich folgende Eigenschaft, die bewiesen werden muss:

$$|\text{Nudelgericht}| \bmod 5 = 0$$

Beweis durch strukturelle Induktion:

Induktionsanfang:
Für den Induktionsanfang müssen wir zeigen, dass die Eigenschaft für das Grundelement der Menge gilt. Das kleinste Nudelgericht erhalten wir durch einmalige Anwendung der Regel (A). Da diese Regel mindestens einmal angewendet werden muss, können wir davon ausgehen, dass wir damit das Grundelement erhalten. Also prüfen wir,

ob die Eigenschaft für das Grundelement gilt:
$|\text{Nudel}| = 10 \Rightarrow 10 \bmod 5 = 0$

Wir sehen: Der Induktionsanfang ist gültig und wir können die Induktionsvoraussetzung definieren.

Induktionsvoraussetzung:
Für ein bestehendes Nudelgericht gilt: $|\text{Nudelgericht}| \bmod 5 = 0$

Im **Induktionsschritt** müssen wir für alle Bildungsregeln beweisen, dass die geforderte Eigenschaft gilt. Also ist zu zeigen:

- nach jeder Anwendung von (A) gilt $|\text{Nudelgericht}| \bmod 5 = 0$
- nach jeder Anwendung von (B) gilt $|\text{Nudelgericht}| \bmod 5 = 0$
- nach jeder Anwendung von (C) gilt $|\text{Nudelgericht}| \bmod 5 = 0$

Beweis:
zu (A): Es wird eine Portion Nudeln zum Nudelgericht hinzugegeben:

$$\begin{aligned}(10 + |\text{Nudelgericht}|) \bmod 5 &= (10 \bmod 5 + |\text{Nudelgericht}| \bmod 5) \bmod 5 \\ &= (0 + |\text{Nudelgericht}| \bmod 5) \bmod 5 \\ &\overset{\text{nach IV}}{=} (0 + 0) \bmod 5 = 0\end{aligned}$$

zu (B): Es wird eine Portion Soße zum Nudelgericht hinzugegeben:

$$\begin{aligned}(15 + |\text{Nudelgericht}|) \bmod 5 &= (15 \bmod 5 + |\text{Nudelgericht}| \bmod 5) \bmod 5 \\ &= (0 + |\text{Nudelgericht}| \bmod 5) \bmod 5 \\ &\overset{\text{nach IV}}{=} (0 + 0) \bmod 5 = 0\end{aligned}$$

zu (C): Es wird eine Portion Käse zum Nudelgericht hinzugegeben:

$$\begin{aligned}(5 + |\text{Nudelgericht}|) \bmod 5 &= (5 \bmod 5 + |\text{Nudelgericht}| \bmod 5) \bmod 5 \\ &= (0 + |\text{Nudelgericht}| \bmod 5) \bmod 5 \\ &\overset{\text{nach IV}}{=} (0 + 0) \bmod 5 = 0\end{aligned}$$

Damit haben wir mittels struktureller Induktion bewiesen, dass das Gewicht eines Nudelgerichts, das unserem Rezept folgt, immer ein Vielfaches von 5g ist.

Beispiel: Aussagen über Terme beweisen

Wir wollen für die Definition 10.2 auf Seite 6 beweisen:
Jeder Term A enthält eine Konstante oder eine Variable

Möchte man eine strukturelle Induktion auf Termen durchführen, muss man sich auch hier auf die Aufbauregeln begrenzen und entlang hangeln.

Beweis mit struktureller Induktion:

1. **Induktionsanfang** (E gilt für die Grundelemente der Menge der Terme):
   Das stimmt aufgrund der Definition eines Terms, denn jedes Grundelement ist entweder eine Variable oder eine Konstante
2. **Induktionsschritt** (Durch Anwendung einer Bildungsregel bleibt E erhalten):
   Induktionsvoraussetzung: $A, A_1, \ldots, A_n$ sind Terme, die jeweils eine Variable oder eine Konstante enthalten
   Zu zeigen: $op(A_1, \ldots, A_n)$ und $f(A)$ enthalten eine Variable oder eine Konstante
   Beweis: Wenn jeder Term $A, A_1, \ldots, A_n$ eine Konstante oder Variable enthält, dann enthalten auch $op(A_1, \ldots, A_n)$ und $f(A)$ eine Variable oder Konstante, da $n \geqslant 1$ ist.

Beispiel: Beweis durch Gegenbeispiel

Wir wollen beweisen:
Nicht jeder Term A enthält eine Operation

Beweis durch Angabe eines Gegenbeispiels:
Jede Variable und jede Konstante ist ein Term, der keine Operation enthält

Möchte man zeigen, dass **nicht** alle Elemente einer Menge eine Eigenschaft E erfüllen, so reicht es ein **Gegenbeispiel** zu finden, d.h. ein Element aus der Menge zu finden, das E **nicht** erfüllt. Genaueres zum Gegenbeispiel findet sich am Ende dieses Skriptes, auf Seite 38.

10.4 Syntaxbäume

Ein Syntaxbaum ist, anders als man eventuell aus der Biologie vermuten kann, keine braun-grüne, Sauerstoff produzierende Pflanze. Unter einem Baum versteht man in der Informatik ein Konstrukt, mit denen sich hierarchische Strukturen verdeutlichen lassen können.

Bevor wir uns jedoch einem (Syntax-) Baum widmen können, muss zuvor geklärt werden, was ein Graph ist, da Bäume auf solchen Graphen aufbauen.

10.4.1 Gerichtete Graphen

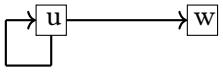
Definition: 10.9 Gerichteter Graph

Ein **gerichteter Graph** ist ein Paar (V, E), wobei

- V eine endliche Menge von **Knoten** (engl. *vertices*) ist, und
- $E \subseteq V \times V$ eine Menge **gerichteter Kanten** (engl. *edges*) zwischen Knoten ist. Ein Element $(v, w) \in E$ bezeichnet eine **Kante von v nach w**.

Beispiel: Graphische Darstellung

Im Folgenden ist ein beispielhafter Graph gegeben:



Dieser Graph besitzt zwei Knoten: u und w, sowie auch zwei Kanten: (u, u) und (u, w). Die erste Kante bedeutet, dass eine Kante von u ausgeht und wieder zu u zurück führt. Die zweite Kante besagt, dass eine Verbindung von u nach w besteht.
Formal geschrieben bedeutet das: $V = \{u, w\}$, $E = \{(u, u), (u, w)\}$

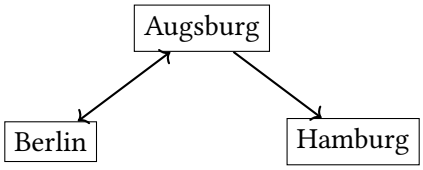
Beispiel:

Graphen können sehr gut anhand realer Dinge beschrieben werden. Eine mögliche, gängige Veranschaulichung von Graphen ist anhand einer Straßenkarte. Wenn wir unsere Welt sehr stark minimieren und zusammenschrumpfen, können wir behaupten, dass drei Städte existieren: Augsburg, Berlin und Hamburg. Es existieren aber auch nur einige Einbahnstraßen (da unsere Graphen gerichtet sind, d.h. die Kanten **eine** Richtung haben):

- Von Berlin nach Augsburg
- Von Augsburg nach Berlin
- Von Augsburg nach Hamburg

Wir können uns das nun folgendermaßen vorstellen: Einbahnstraßen stellen Kanten, Städte stellen Knoten dar.
$V = \{\text{Berlin}, \text{Augsburg}, \text{Hamburg}\}$
$E = \{(\text{Berlin}, \text{Augsburg}), (\text{Augsburg}, \text{Berlin}), (\text{Augsburg}, \text{Hamburg})\}$

Dieser Graph sieht nun folgendermaßen aus:



Wir sehen, dass die Verbindung Augsburg – Berlin in beide Richtungen geht. Das liegt daran, dass wir hier zwei Kanten definiert haben: Eine für Augsburg → Berlin und eine zweite für Berlin → Augsburg.

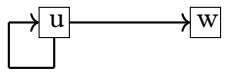
Definition: 10.11 Vorgänger und Nachfolger eines Knotens

Sei (V, E) ein gerichteter Graph:

- Die **Vorgänger** eines Knotens $v \in V$ sind die Elemente der Menge $\{w \mid w \in V, (w, v) \in E\}$
- Die **Nachfolger** eines Knotens $v \in V$ sind die Elemente der Menge $\{w \mid w \in V, (v, w) \in E\}$

Umgangssprachlich bedeutet das: Die Knoten, die auf einen Knoten zeigen, nennt man Vorgänger. Die Knoten, auf die ein Knoten zeigt, nennt man Nachfolger.

Beispiel: Vorgänger, Nachfolger



Vorgänger von u: $\{u\}$
Nachfolger von u: $\{u, w\}$

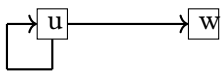
Vorgänger von w: $\{u\}$
Nachfolger von w: $\emptyset$

Definition: 10.13 Azyklische gerichtete Graphen

Sei (V, E) ein gerichteter Graph:

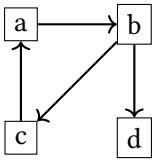
- Ein **Zyklus** ist eine Folge von Knoten $v_1, ..., v_n \in V$ mit $(v_1, v_2), ..., (v_{n-1}, v_n), (v_n, v_1) \in E$ und $n \in \mathbb{N}$.
- Ein gerichteter Graph heißt **azyklisch** falls er keine Zyklen enthält.

Beispiel: Graph mit Zyklus I



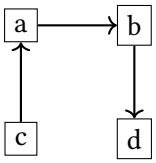
Der Graph enthält den Zyklus u

Beispiel: Graph mit Zyklus II



Der Graph enthält den Zyklus a–b–c–a

Beispiel: Graph ohne Zyklus



Der Graph enthält keinen Zyklus.

10.4.2 Gerichtete Bäume

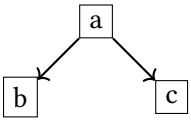
Definition: 10.15 Baum

Ein **gerichteter Baum** ist ein gerichteter Graph (V, E) mit folgenden Eigenschaften:

1. der Graph ist azyklisch
2. es gibt genau einen Knoten ohne Vorgänger, genannt **Wurzel**
3. jeder Knoten hat höchstens einen Vorgänger

Gerichtete Bäume zeichnet man, wie Stammbäume, oft von der Wurzel aus **von oben nach unten**, weil man nicht weiß, wie groß sie werden.

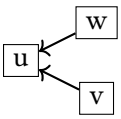
Beispiel: Gerichteter Baum



Die Wurzel ist der Knoten a.

Beispiel: KEINE Bäume

Denn: Eigenschaften (1) und (2) sind nicht erfüllt.



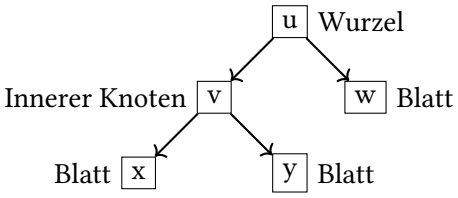
Denn: Eigenschaften (2) und (3) sind nicht erfüllt.

Definition: 10.16 Vater, Kind, Blatt

Sei (V, E) ein gerichteter Baum und $v, w \in V$:

- w ist **Vater** von v, falls $(w, v) \in E$
- w ist **Kind** von v, falls $(v, w) \in E$
- w ist ein **Blatt**, falls w keine Kinder hat
- w ist ein **innerer Knoten**, falls w weder ein Blatt noch die Wurzel ist

Beispiel:



- u ist Vater von v und w
- v und w sind Kinder von u
- v ist Vater von x und y
- x und y sind Kinder von v

10.4.3 Was ist ein Syntaxbaum?

Definition: 10.17 Syntaxbaum

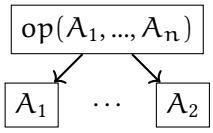
Der **Syntaxbaum eines Terms** A ist der wie folgt definierte gerichtete Baum (V_A, E_A):

- V_A ist die Menge der Teilterme von A, wobei A die Wurzel ist
- Die Teilterme, die Variablen oder Konstanten entsprechen, sind die Blätter des Baumes
- Teilterme der Form $op(A_1, \ldots, A_n)$ haben die Kinder $A_1, ..., A_n$, also $(op(A_1, \ldots, A_n), A_i) \in E_A$ für jedes i.
- Teilterme der Form $f(A)$ haben ein Kind A, also $(f(A), A) \in E_A$.

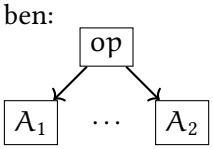
Umgangssprachlich gesagt ist ein Syntaxbaum ein gerichteter Baum, dessen Knoten spezielle Beschriftungen besitzen. Die Wurzel sowie innere Knoten stellen Operationen dar. Die Blätter des Baumes stellen Variablen und Konstanten dar. Durch die Hierarchie des Baumes wird die Abarbeitungsreihenfolge beschrieben.

Oft schreibt man kurz nur op statt $op(A_1, ..., A_n)$ und f statt $f(A)$

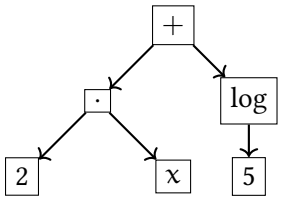
Anstatt das folgende zu schreiben:



Wird der Einfachheit halber folgendes geschrieben:



Beispiel: Syntaxbaum des Terms $((2 \cdot x) + \log(5))$



Der Vorteil eines Syntaxbaumes ist die klare Darstellung von Aufbau und Struktur eines Terms. Auch die Auswertungsreihenfolge in einem Term ist klar dargestellt: Kinder müssen offenbar vor deren Vätern ausgewertet werden.

Solche Syntaxbäume werden z.B. Compiler-intern verwendet, um Ausdrücke in ihre Einzelbestandteile zu zerlegen und in der korrekten Reihenfolge auszuwerten.

10.5 Aussagenlogische Formeln

Die Aussagenlogik ist Teil der Logik und wird unter anderem auch in Diskrete Strukturen und Logik angesprochen. Wir beschäftigen uns nun mit aussagenlogischen Formeln. Das sind Formeln,

die am Ende einen Wahrheitswert zurückgeben. Eine einfaches Beispiel wäre $(p_1 \wedge p_2)$.

Wie so oft müssen wir allerdings noch einige Grundlagen definieren und erklären, bevor wir das eigentliche Thema besprechen können. Hierfür müssen wir zunächst unser Verständnis von Relationen erweitern.

10.5.1 Relation

Im Kapitel 4 hatten wir bereits *binäre Relationen* kennengelernt, die das Verhältnis zweier Elemente zueinander beschreiben. Eine solche binäre Relation wäre beispielsweise $x < y$ auf der Menge $\mathbb{R}$. Eine binäre Relation ist eine 2-stellige Relation, da immer zwei Elemente zueinander in Beziehung gesetzt werden. Es sind aber auch Relationen mit anderen Stelligkeiten möglich:

Definition: 10.19 n-stellige Relation

Seien $n \in \mathbb{N}$ und $T_1, \ldots, T_n$ Mengen

- Die Menge

$$T_1 \times \ldots \times T_n := \{(t_1, \ldots, t_n) \mid t_i \in T_i,\ i \in \mathbb{N},\ 1 \leqslant i \leqslant n\}$$

  heißt **kartesisches Produkt der Mengen** $T_1, \ldots, T_n$. Dessen Elemente heißen **n-Tupel**. t_i ist die **i-te Komponente** von $(t_1, \ldots, t_n)$.
- Eine **n-stellige Relation auf** $T_1 \times \ldots \times T_n$ ist eine Teilmenge $R \subseteq T_1 \times \ldots \times T_n$.

Diese Definition besagt, dass bei einer n-stelligen Relation n-viele Elemente miteinander in Relation stehen.

Beispiel: 1-stellige Relationen

Eine 1-stellige Relation ist eine Menge einfacher Elemente ohne Komponenten.
Beispiele für 1-stellige Relationen:
$\mathbb{N}, \mathbb{Z}, \{0, 1\}, \ldots$

Beispiel: 2-stellige Relationen

Eine 2-stellige Relation ist eine Menge von Elementen mit 2 Komponenten. Solche Relationen heißen auch **binär** und haben wir schon besprochen.

- Beispiele für 2-stellige Relationen auf $\mathbb{R} \times \mathbb{R}$:
  $<, \leqslant, >, \geqslant, =, \neq, \ldots$
- Beispiele für 2-stellige Relationen auf $P(A) \times P(A)$:
  $\subset, \subseteq, \supset, \supseteq, =, \neq, \ldots$

10.5.2 Aussagenlogische Formeln

Definition: 10.22 Aussagenlogische Formel

Wir definieren mit struktureller Induktion:

- Seien $R \subseteq T_1 \times \ldots \times T_n$ eine n-stellige Relation und A_i ein Term vom Typ T_i für $i \in \mathbb{N}$, $1 \leqslant i \leqslant n$. Dann ist $(A_1, \ldots, A_n) \in R$ ist eine **aussagenlogische Formel**.
- Sind p und q aussagenlogische Formeln, so lassen sich folgende weitere **aussagenlogische Formeln** bilden:
  - (Logisches Und) $p \wedge q$
  - (Logisches Oder) $p \vee q$
  - (Logisches Nicht) $\neg p$
  - (Implikation) $p \Rightarrow q$
  - (Äquivalenz) $p \Leftrightarrow q$

Der erste Teil der Definition besagt, dass wir Terme nun mit Relationen zueinander in Beziehung setzen können. Bislang konnten wir nur Terme wie z.B. $x + y$ definieren. Nun können wir zusätzlich Relationen darauf anwenden und z.B. $x < y$ formulieren. Damit handelt es sich um eine aussagenlogische Formel, die wir zu einem Wahrheitswert (Wahr, Falsch) auswerten können.

Im zweiten Teil verbinden wir zwei dieser aussagenlogischen Formeln (p und q) mit logischen Ausdrücken, sodass eine neue aussagenlogische Formel entsteht. Somit sind wir nun in der Lage, auszudrücken, dass z.B. sowohl p als auch q gelten sollen.

Beispiel: Aussagenlogische Formeln

- A ist Teilmenge von B: $A \subseteq B$
- x ist eine positive ganze Zahl: $x > 0 \wedge x \in \mathbb{Z}$
- x hat maximal 3 Nachkommastellen: $x \cdot 1000 \in \mathbb{Z}$
- x und y haben einen Abstand von höchstens 0.5: $|x - y| \leqslant 0.5$
- x ist nicht durch y teilbar: $\neg(x \bmod y = 0)$

Ob die Formeln erfüllt (wahr) sind oder nicht hängt nur von der Belegung der Variablen A, B, x, y ab.

Exkurs: Implikation und Äquivalenz
Sowohl Implikation als auch die Äquivalenz sind wichtige Grundlagen der Mathematik, Informatik und Logik. In der Logik, wie wir sie momentan betrachten, können dadurch auch Wahrheitswerte evaluiert werden.

Implikation:
Die Implikation $a \Rightarrow b$ besagt: aus a folgt b. In einer Wahrheitstabelle würde es folgendermaßen aussehen:

a	b	$a \Rightarrow b$
Wahr	Wahr	Wahr
Wahr	Falsch	Falsch
Falsch	Wahr	Wahr
Falsch	Falsch	Wahr

Hier gilt zu beachten, dass wenn a falsch ist, die Folgerung **immer** wahr ist. Man sagt für dieses Phänomen auch: *Ex falso quodlibet* – „Aus Falschem folgt Beliebiges“. Ein natürlichsprachiges Beispiel hierfür wäre: „Birnen sind T-Shirts, deswegen können alle den Informatik Bachelor in einem Semester schaffen“.

Äquivalenz:
Die Äquivalenz $a \Leftrightarrow b$ besagt in der Logik: a ist logisch äquivalent zu b. Alternativ kann man die Äquivalenz auch mithilfe der Implikation darstellen: $(a \Leftarrow b) \wedge (a \Rightarrow b)$

In einer Wahrheitstabelle würde es folgendermaßen aussehen:

a	b	$a \Leftrightarrow b$
Wahr	Wahr	Wahr
Wahr	Falsch	Falsch
Falsch	Wahr	Falsch
Falsch	Falsch	Wahr

10.5.3 Erfüllung aussagenlogischer Formeln

Ob eine Formel erfüllt ist oder nicht, hängt von der Belegung der Variablen in den beteiligten Termen ab.

Definition: 10.23 Erfüllung

Seien $R \subseteq T_1 \times \ldots \times T_n$ eine n-stellige Relation und A_i ein Term vom Typ T_i für $i \in \mathbb{N}$, $1 \leqslant i \leqslant n$. Wir definieren mit struktureller Induktion für eine Belegung β der Variablen in den Termen:

- $(A_1, \ldots, A_n) \in R$ ist **erfüllt bzgl.** β, falls $(A'_1, \ldots, A'_n) \in R$
- $p \wedge q$ ist **erfüllt bzgl.** β, falls p **und** q erfüllt sind
- $p \vee q$ ist **erfüllt bzgl.** β, falls p **oder** q erfüllt ist
- $\neg p$ ist **erfüllt bzgl.** β, falls p **nicht** erfüllt ist
- $p \Rightarrow q$ ist **erfüllt bzgl.** β, falls $(\neg p) \vee q$ erfüllt ist (**wenn** p**, dann** q)
- $p \Leftrightarrow q$ ist **erfüllt bzgl.** β, falls $(p \Rightarrow q) \wedge (q \Rightarrow p)$ erfüllt ist (p **genau dann wenn** q)

Beispiel:

Sei nun die aussagenlogische Formel $a \wedge b$ gegeben.
Ist diese Formel erfüllbar? Ja, wenn sowohl a als auch b wahr sind.

Beispiel:

Sei die 2-stellige Relation $p = x < y$ gegeben.

Diese Relation ist erfüllt, wenn der Wert x kleiner ist als y.

Weitere Beispiele aussagenlogischer Formeln
Ob eine Formel erfüllt ist oder nicht, hängt von der Belegung der Variablen in den beteiligten Termen ab.

Beispiel: Implikation

Die Formel $p \Rightarrow q$ ist gleichwertig zu der Formel $\neg p \vee q$. Diese ist in folgenden Fällen erfüllt:

- p und q sind erfüllt
- p und q sind nicht erfüllt
- p ist nicht erfüllt und q ist erfüllt

Allen Fällen gemeinsam ist: **Wenn p erfüllt ist, dann ist auch q erfüllt**.
Wenn p nicht erfüllt ist, dann kann q erfüllt sein oder nicht. Beispiel:

- Teilbarkeit: $(x \bmod 4 = 0) \Rightarrow (x \bmod 2 = 0)$

Beispiel: Äquivalenz

Die Formel $p \Leftrightarrow q$ ist gleichwertig zu der Formel $(p \wedge q) \vee (\neg p \wedge \neg q)$. Beispiel:

- Gleichungen: $(x + 5 = 10) \Leftrightarrow (x = 10 - 5)$

10.5.4 Besondere aussagenlogische Formeln

Eine besondere Art einer aussagenlogischen Formel, die wir kurz ansprechen wollen, ist die *Tautologie*. Eine Formel wird nämlich Tautologie genannt, wenn sie **immer** erfüllbar ist.

Definition: 10.27 Tautologie

Eine Formel p heißt **Tautologie**, wenn sie für **jede Belegung** der Variablen erfüllt ist.
Beispiele:

- $(x < 0) \vee (x \geqslant 0)$
- $(x \bmod 4 = 0) \Rightarrow (x \bmod 2 = 0)$
- $(x + 5 = 10) \Leftrightarrow (x = 10 - 5)$

Im Gegensatz dazu nennt man eine Formel *nicht erfüllbar*, wenn Sie **nie** erfüllbar ist.

Definition: 10.28 Erfüllbarkeit

Eine Formel p heißt **nicht erfüllbar**, wenn sie für **keine Belegung** der Variablen erfüllt ist.
Beispiel:

- $(x < 0) \wedge (x \geqslant 0)$

10.6 Prädikatenlogische Formeln

10.6.1 Motivation

Ob eine aussagenlogische Formel p, die eine Variable x enthält, wie z.B. $x < 5$, erfüllt ist, hängt ab von der Belegung der Variablen.

Mit Hilfe der Prädikatenlogik lassen sich Aussagen der folgenden Art für eine solche aussagenlogische Formel p treffen:

- **Es gibt** eine Belegung von x, so dass p erfüllt ist: $\exists x(p)$
- **Für alle** Belegungen von x ist p erfüllt: $\forall x(p)$

Das bedeutet: Wir wollen mit der Prädikatenlogik eine explizite Aussage treffen, ob eine aussagenlogische Formel erfüllt werden kann und wenn ja, mit welchen Belegungen.

Beispiel:

Für alle Werte von x, y gilt: Falls x kleiner ist als y, dann gibt es ein z das größer ist als x und kleiner als y
$\forall x, y((x < y) \Rightarrow \exists z((x < z) \wedge (z < y)))$

- Diese Aussage ist **erfüllt**, falls x, y, z reelle Zahlen sind.
- Diese Aussage ist **nicht erfüllt**, falls x, y, z ganze Zahlen sind.

10.6.2 Exkurs: Quantoren

In der Mathematik bzw. Logik und somit auch in der Informatik unterscheidet man zwei verschiedene Quantoren:
Den Allquantor $\forall$ sowie den Existenzquantor $\exists$.

Mithilfe dieser Quantoren können Belegungen definiert werden.

Beispiel:

$\forall x \in \mathbb{N}\ \exists y \in \mathbb{N} : (x < y)$ besagt: Für alle x aus $\mathbb{N}$ gibt es ein y aus $\mathbb{N}$, das größer ist als das gewählte x.

Oder einfach ausgedrückt: Zu jeder natürlichen Zahl gibt es eine größere natürliche Zahl.

Bindungsstärke
Die Quantoren besitzen auch eine gewisse Bindungsstärke. Sie sind also nur für einen gewissen Teil der Formel gültig. Bei uns gilt: Quantoren binden schwächer als Operatoren.

Das heißt: Sei bspw. folgende Formel gegeben: $\forall x : x \Rightarrow x$
Dann bezieht sich das x aus dem Quantor nur auf das erste x. Das zweite x ist **nicht** vom Allquantor betroffen. Deswegen sollte **immer** geklammert werden, um „auf der sicheren Seite zu sein“.

10.6.3 Definition von prädikatenlogischen Formeln

Definition: 10.30 Prädikatenlogische Formel

Wir definieren mit struktureller Induktion:

- Seien $R \subseteq T_1 \times \ldots \times T_n$ eine n-stellige Relation und A_i ein Term vom Typ T_i für $i \in \mathbb{N}, 1 \leqslant i \leqslant n$. Dann ist $(A_1, \ldots, A_n) \in R$ eine **prädikatenlogische Formel**
- Sind p und q prädikatenlogische Formeln, und ist x eine **freie Variable in** p, so lassen sich folgende weitere **prädikatenlogische Formeln** bilden:
  - $p \wedge q$, $p \vee q$, $\neg p$, $p \Rightarrow q$, $p \Leftrightarrow q$
  - (Allquantor) $\forall x(p)$
    Lies: “Für alle x”
  - (Existenzquantor) $\exists x(p)$
    Lies: “Es gibt ein x”

  In den Formeln $\forall x(p)$ und $\exists x(p)$ ist die Variable x **gebunden**. Nicht gebundene Variablen heißen **frei**.

10.6.4 Erfüllung prädikatenlogischer Formeln

Ob eine prädikatenlogische Formel erfüllt ist oder nicht, hängt von der **Belegung der freien Variablen** in den beteiligten Termen ab.

Definition: 10.31 Erfüllung

Wir definieren mit struktureller Induktion für eine Belegung β der **freien Variablen** in den Termen:

- $(A_1, \ldots, A_n) \in R$ ist **erfüllt bzgl.** β, falls $(A'_1, \ldots, A'_n) \in R$
- $p \wedge q$ ist **erfüllt bzgl.** β, falls p **und** q erfüllt sind
- $p \vee q$ ist **erfüllt bzgl.** β, falls p **oder** q erfüllt ist
- $\neg p$ ist **erfüllt bzgl.** β, falls p **nicht** erfüllt ist
- $p \Rightarrow q$ ist **erfüllt bzgl.** β, falls $(\neg p) \vee q$ erfüllt ist (**wenn** p**, dann** q)
- $p \Leftrightarrow q$ ist **erfüllt bzgl.** β, falls $(p \Rightarrow q) \wedge (q \Rightarrow p)$ erfüllt ist (p **genau dann wenn** q)
- $\forall x(p)$ **erfüllt bzgl.** β, wenn p für **alle** Belegungen von x erfüllt ist.
- $\exists x(p)$ **erfüllt bzgl.** β, wenn p für **eine** Belegung von x erfüllt ist

Diese Definition unterscheidet sich im Grunde von der Definition aus Kapitel 10.5.3 nur in den letzten zwei Zeilen. Es wird die Erfüllbarkeit von Quantoren ergänzt.

Die Rolle der Variablentypen

Die Quantoren beziehen sich immer auf den Typ der Variable. Der Wahrheitswert der Formel hängt von diesem Typ ab.

Beispiel:

$\forall x, y((x < y) \Rightarrow \exists z((x < z) \wedge (z < y)))$ [1]

- ist erfüllt, falls x, y, z den Typ $\mathbb{R}$ haben (Satz aus der Mathematik)
- ist nicht erfüllt, falls x, y, z den Typ $\mathbb{Z}$ haben (Beweis per Gegenbeispiel: für $x = 3$ und $y = 4$ gibt es keine solche ganze Zahl z)

Einschränkung des Typs

Man kann den Typ einer Variable x wie folgt auf einen Typ M einschränken:

- $\forall x((x \in M) \Rightarrow p)$: Hier wird nur für Werte von x aus M gefordert, dass p erfüllt ist.
  Abkürzende Schreibweise: $\forall x \in M(p)$
- $\exists x((x \in M) \wedge p)$: Hier wird gefordert, dass es einen Wert von x aus M gibt, für den p erfüllt ist.
  Abkürzende Schreibweise: $\exists x \in M(p)$

Beispiele für prädikatenlogische Formeln

Beispiel: Sortierung

- Natürlichsprachliche Formulierung:
  Die Folge $a_1, \ldots, a_n$ *ist aufsteigend sortiert*
- Formulierung als prädikatenlogische Formel:

[1] Die Formel wird folgendermaßen ausgesprochen: „Für alle x, y gilt: Ist x kleiner y, folgt daraus, dass ein z existiert, für das gilt: x ist kleiner als z und z ist kleiner als y“.

$\forall i \in \{1, 2, \ldots, n-1\}(a_i < a_{i+1})$ [2]
(i ist vom Typ $\mathbb{N}$, der Typ von $a_1, \ldots, a_n$ ist hier offen gelassen)

Beispiel: Primzahlen

- Natürlichsprachliche Formulierung:
  Die Zahl p *ist eine Primzahl*
- Formulierung als prädikatenlogische Formel:
  $\neg(\exists k((k > 1) \wedge (k < p) \wedge (p \bmod k = 0))$ [3]
  Oder: $\forall k \in \{1, 2, \ldots, p\}((p \bmod k = 0) \Rightarrow (k = 1) \vee (k = p))$
  (k und p sind vom Typ $\mathbb{N}$)

Beispiel: Palindrome

Ein Palindrom ist ein Wort, das Rückwärts gelesen das selbe Wort ergibt wie vorwärts gelesen.

- Natürlichsprachliche Formulierung:
  Das Wort $a_1 \ldots a_n \in A^+$ *ist ein Palindrom*
- Formulierung als prädikatenlogische Formel:
  $\forall i \in \{1, 2, \ldots, n \div 2\}(a_i = a_{n+1-i})$ [4]
  (i ist vom Typ $\mathbb{N}$)

Beispiel: Mit a beginnende Wörter

- Natürlichsprachliche Formulierung:
  Das Wort $w \in A^+$ *beginnt mit dem Buchstaben* $a \in A$
- Formulierung als prädikatenlogische Formel:
  $\exists u \in A^*(w = au)$ [5]

10.7 Problemspezifikation

Grundlegende Forderung an C-Funktionen
Eine grundlegende Forderung an eine C-Funktion ist, dass sie eine vorgegebene Problemstellung löst. In der Praxis ist die Problemstellung aber oft sehr vage. Deshalb präzisiert und formalisiert man die Problemstellung vor der Implementierung der C-Funktion. Eine solche Präzisierung nennt man **Problemspezifikation**. Die Problemspezifikation hilft dann oft auch bei der korrekten Implementierung.

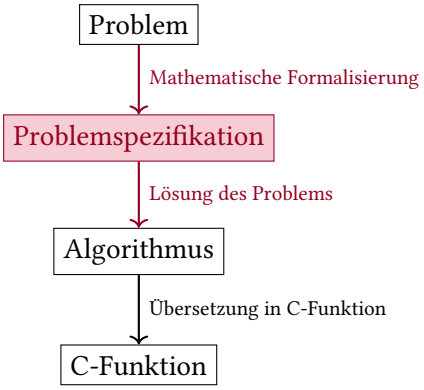
[2] Die Formel wird folgendermaßen ausgesprochen: „Für alle i von 1 bis $n-1$ gilt: a_i ist kleiner als a_{i+1}".
[3] Die Formel wird folgendermaßen ausgesprochen: „Es gilt nicht, dass es ein k gibt, sodass k größer ist als 1 und k kleiner ist als p und dass p modulo k gleich 0 ist".
[4] Die Formel wird folgendermaßen ausgesprochen: „Für alle i von 1 bis $n \div 2$ gilt: a_i ist gleich a_{n+1-i}".
[5] Die Formel wird folgendermaßen ausgesprochen: „Es gibt ein u aus A^*, sodass gilt: w ist gleich a mal u".

Löst eine C-Funktion die durch eine Problemspezifikation gegebene Problemstellung, so nennt man sie **korrekt**.

> Definition: 10.37 Verifikationsverfahren
>
> Verifikationsverfahren dienen dem Nachweis der Korrektheit einer C-Funktion bzgl. einer Problemspezifikation.

> Definition: 10.38 Validierungsverfahren
>
> Validierungsverfahren dienen dem Nachweis der Korrektheit einer Problemspezifikation bzgl. eines realen Problems.

Übersicht



Zur Lösung eines Problems entwirft man vor der Programmierung eine **mathematische Formulierung des Problems** und erstellt eine **programmiersprachen-unabhängige Lösung in Form eines sog. Algorithmus**. Dazu wird die Problemlösung in mehrere Schritte zerlegt. Gleichzeitig wird das Problem präzisiert und konkretisiert sowie Unklarheiten und Widersprüche beseitigt.

Was ist eine Problemspezifikation?

> Definition: 10.39 Problemspezifikation
>
> Eine **Problemspezifikation** ist eine exakte (mathematische) Definition der Problemstellung durch folgende Festlegungen:
>
> - **Eingabedaten**
>   Art und Form der Daten, Wertebereiche, Einschränkungen
> - **Ausgabedaten**
>   Wie bei Eingabedaten
> - **Funktionaler Zusammenhang zwischen Ein- und Ausgabedaten**
>   Welche Eingabedaten sollen welche Ausgabedaten erzeugen?

Es gilt hier zu beachten: Die Problemspezifikation ist programmiersprachenunabhängig. Sie ist formal, z.B. mit Termen bzw. aussagenlogischen oder prädikatenlogischen Formeln aufzuschreiben. Es darf also kein C-Code o.ä. verwendet werden.

Beispiel:

Problemstellung: *Berechne das Produkt zweier ganzer Zahlen*

- **Eingabe**: $a, b \in \mathbb{Z}$
- **Ausgabe**: $x \in \mathbb{Z}$
- **Funktionaler Zusammenhang**: $x = a \cdot b$

Beispiel für eine unpräzise Problemstellung
Die Problemspezifikation hängt in erster Linie von der Problemstellung ab. Ist diese schon ungenau, mehrdeutig, etc., kommt es schon früh zu Missverständnissen und Fehlern.

Beispiel:

Problemstellung: *Suche ein Wort in einer Liste von Wörtern*

Es ist noch Folgendes unklar und muss in der Problemspezifikation präzisiert werden:

- Welche Wörter sind erlaubt (z.B. was ist das Alphabet?,...)
- Ist die Liste sortiert oder nicht sortiert?
- Ist die Liste wiederholungsfrei oder nicht?
- Was ist die Ausgabe, wenn das Wort in der Liste vorkommt (ja/nein, Position des Treffers,...)
- Was ist die Ausgabe, wenn das Wort nicht in der Liste vorkommt?
- Was ist die Ausgabe, wenn das Wort öfter in der Liste vorkommt?

Würde man doch die Problemstellung verwenden, kann sie von unterschiedlichen Leuten unterschiedlich aufgefasst und verstanden werden. So können auch mehrere Varianten der Problemspezifikation entstehen – was dazu führen kann, dass das Programm an anderer Stelle nicht mehr funktioniert oder ähnliches (Stellen Sie sich bspw. vor, Sie und Ihre Kommilitonen nutzen eine gemeinsame Bibliothek, von der jeder eine andere Funktionalität erwartet).

Für die bessere Übersicht wurden hier einige Klammern farblich eingefärbt.

Beispiel:

Problemspezifikation: Variante 1

- **Eingabe**:
  $w_1, \ldots, w_n \in A^+$ unsortierte Folge und $w \in A^+$ für ein Alphabet A
- **Ausgabe**:
  $k \in \mathbb{N}$
- **Funktionaler Zusammenhang**:
  $[(\forall i \in \{1, 2, \ldots, n\}(w_i \neq w)) \wedge (k = n + 1)]$
  $\vee$

$[(w_k = w) \wedge (\forall i \in \{1, 2, \ldots, k-1\}(w_i \neq w))]$

- Ausgabe der ersten Position k mit $w_k = w$
- i ist vom Typ $\mathbb{N}$

Beispiel:

Problemspezifikation: Variante 2

- **Eingabe**:
  $w_1, \ldots, w_n \in A^+$ aufwärts sortierte Folge (d.h. es gilt $\forall i \in \{1, 2, ..., n-1\}(w_i \leqslant w_{i+1})$) und $w \in A^+$ für ein geordnetes Alphabet A und die lexikografische Ordnung $\leqslant$ auf A^+
- **Ausgabe**:
  $k \in \mathbb{N}$
- **Funktionaler Zusammenhang**:
  $[(\forall i \in \{1, 2, \ldots, n\}(w_i < w)) \wedge (k = n + 1)]$
  $\bigvee$
  $[(w_k \geqslant w) \wedge (\forall i \in \{1, 2, \ldots, k-1\}(w_i < w))]$

  - Ausgabe der ersten Position k mit $w_k \geqslant w$
  - i ist vom Typ $\mathbb{N}$

Ein weiteres Beispiel wäre das Testen, ob ein Wort ein Palindrom ist.

Beispiel:

Problemstellung: *Teste, ob ein Wort ein Palindrom ist*

- **Eingabe**:
  $a_1 \ldots a_n \in A^+$ für ein Alphabet A
- **Ausgabe**:
  $k \in \{0, 1\}$
- **Funktionaler Zusammenhang**:
  $[(\forall i \in \{1, 2, \ldots, n \div 2\}(a_i = a_{n+1-i})) \Rightarrow (k = 1)]$
  $\wedge$
  $[(\exists i \in \{1, 2, \ldots, n \div 2\}(a_i \neq a_{n+1-i})) \Rightarrow (k = 0)]$

  - Erinnerung: Ein Wort ist ein Palindrom, falls es vorwärts und rückwärts gelesen gleich ist
  - i ist vom Typ $\mathbb{N}$

10.8 Beweistechniken

Das Formulieren einer Problemspezifikation ist die eine Sache. Wir müssen aber auch in der Lage sein, unsere These bzw. Idee zu beweisen – denn ohne Beweis könnten wir ja sonst beliebig verrückte Dinge definieren.

10.8.1 Implikation von Aussagen

Seien p und q Formeln für die wir $p \Rightarrow q$ zeigen wollen. Zwei mögliche Beweistechniken sind *Implikationsketten* und die *Kontraposition*.

1. Implikationskette:
Finde eine Kette von Formeln $p = t_1, t_2, \ldots, t_{n-1}, t_n = q$, so dass

$$p = t_1 \Rightarrow t_2 \Rightarrow \ldots \Rightarrow t_n = q$$

wobei $t_i \Rightarrow t_{i+1}$ für jedes i aus einer Definition, einer bereits bewiesenen Aussage oder bekannten Rechenregeln folgt.

Was auch **wichtig zu beachten ist:** Jedes $\Rightarrow$-Zeichen der Rechnung, das aus einer Definition oder Aussage folgt, wird mit darübergestelltem Verweis begründet!

Beispiel: 10.42

Sei A ein Alphabet.
Beweise: Die *„ist Präfix von"*-Relation auf A^* ist transitiv.
Zu zeigen ist für $u, v, w \in A^*$: Ist u Präfix von v und v Präfix von w, dann ist u auch Präfix von w.
Es gilt:

$$\begin{aligned}(u \text{ Präfix von } v) \wedge (v \text{ Präfix von } w) &\overset{(1)}{\Rightarrow} \exists x \in A^*(v = ux) \wedge \exists y \in A^*(w = vy)\\ &\overset{(2)}{\Rightarrow} \exists x, y \in A^*(w = uxy)\\ &\overset{(2)}{\Rightarrow} \exists z \in A^*(w = uz)\\ &\overset{(1)}{\Rightarrow} (u \text{ Präfix von } w)\end{aligned}$$

- (1): Definition von Präfixen
- (2): Lässt sich kürzer schreiben als

2. Kontraposition:
Die Kontraposition nutzt die Gleichheit von logischen Aussagen. Nämlich gilt:

$$(p \Rightarrow q) \Leftrightarrow (\neg q \Rightarrow \neg p)$$

Dass diese Aussage gilt, lässt sich leicht durch eine Wahrheitstafel bestätigen (eine schöne kleine Übung für zwischendurch ;)). Diese Äquivalenz nutzen wir nun aus. Wird die linke Seite der Äquivalenz bewiesen, dann gilt auch die rechte Seite. Wird wiederum die rechte Seite bewiesen, muss natürlich auch die linke Seite gelten.

Somit formulieren wir unser Problem um und wollen nun beweisen:

$$\neg q \Rightarrow \neg p$$

Beispiel: 10.43

Beweise: $(x \bmod 2 \neq 0) \Rightarrow (x \bmod 4 \neq 0)$:

Dieses Problem liegt momentan in der Form von $p \Rightarrow q$ vor. Formulieren wir das Problem um, bekommen wir die folgende Darstellung:
Beweise: $(x \bmod 4 = 0) \Rightarrow (x \bmod 2 = 0)$

Es gilt:

$$\begin{aligned}(x \bmod 4 = 0) &\overset{(1)}{\Rightarrow} \exists y \in \mathbb{N}(x = 4 \cdot y)\\ &\overset{(2)}{\Rightarrow} \exists y \in \mathbb{N}(x = 2 \cdot (2 \cdot y))\\ &\overset{(3)}{\Rightarrow} \exists y \in \mathbb{N}(x = 2 \cdot y)\\ &\overset{(1)}{\Rightarrow} (x \bmod 2 = 0)\end{aligned}$$

- (1): Definition der Operation mod
- (2): Lässt sich auch schreiben als
- (3): Lässt sich auch schreiben als (wähle auf der rechten Seite ein doppelt so großes y wie auf der linken Seite)

10.8.2 Äquivalenz von Aussagen

Bei der Äquivalenz von Aussagen geht man ähnlich vor wie bei der Implikation von Aussagen. Wir können eine *Äquivalenzkette* nutzen, einen *Ringschluss*, die Äquivalenz *in Implikationen zerlegen* oder die *Negation* der Aussage nutzen.

Seien p und q Formeln für die wir $p \Leftrightarrow q$ zeigen wollen.

1. Äquivalenzkette:
Finde eine Kette von Formeln $p = t_1, t_2, \ldots, t_{n-1}, t_n = q$, so dass

- $t_1 \Leftrightarrow t_2 \Leftrightarrow \ldots \Leftrightarrow t_n$, wobei

- $t_i \Leftrightarrow t_{i+1}$ für jedes i aus einer Definition, einer bereits bewiesenen Aussage oder bekannten Rechenregeln folgt

Jedes $\Leftrightarrow$-Zeichen der Rechnung, das aus einer Definition oder Aussage folgt, wird mit darübergestelltem Verweis begründet.

Beispiel: 10.44

Beweise: ($\forall x(p)$ ist erfüllt) $\Leftrightarrow$ ($\neg(\exists x(\neg p))$ ist erfüllt)

Es gilt:
$$\begin{aligned}(\forall x(p) \text{ ist erfüllt}) &\overset{(1)}{\Leftrightarrow} (p \text{ ist für jede Belegung von x erfüllt})\\ &\overset{(2)}{\Leftrightarrow} (\neg p \text{ ist für keine Belegung von x erfüllt})\\ &\overset{(3)}{\Leftrightarrow} (\neg(\exists x(\neg p)) \text{ ist erfüllt})\end{aligned}$$

- (1): Definition von Erfüllbar für den All-Quantor
- (2): natürlichsprachliche Umformulierung
- (3): Definition von Erfüllbar für den Existiert-Quantor

Anmerkung: Wir machen hier Formeln selbst zu mathematischen Objekten in Formeln, indem wir die einstellige 'ist erfüllt'-Relation auf der Menge der Formeln betrachten.

2. Ringschluss:
Finde Ketten von Formeln $p = t_1, t_2, \ldots, t_{n-1}, t_n = q$ und $q = s_1, s_2, \ldots, s_{m-1}, s_m = p$, so dass

- $p = t_1 \Rightarrow t_2 \Rightarrow \ldots \Rightarrow t_n = q$
- $q = s_1 \Rightarrow s_2 \Rightarrow \ldots \Rightarrow s_m = p$

Wir beweisen also einerseits eine Implikationskette von p zu q, andererseits eine Implikationskette von q zu p.

3. Zerlegung in Implikationen:
Wir können eine Äquivalenz in zwei Implikationen zerlegen und somit beide Implikationen einzeln beweisen.
Beweise: $p \Rightarrow q$ und $p \Leftarrow q$.

4. Negation:
Wir können die Äquivalenz auch negieren. Somit kann es sein, dass sich der Beweis vereinfacht.
Beweise: $\neg p \Leftrightarrow \neg q$

10.8.3 Strukturelle Induktion

Die strukturelle Induktion wurde im Kapitel 10.3 bereits ausführlich besprochen. Deswegen wird nur nochmal ein Überblick der Vorgehensweise gegeben.

Sei M eine induktiv definierte Menge:

1. **Induktionsanfang**: Angabe von *Grundelementen* von M
2. **Induktionsschritt**: Angabe von *Bildungsregeln*, mit denen aus vorhandenen Elementen von M neue Elemente von M erstellt werden können

M besteht genau aus den Elementen, die man aus den Grundelementen durch wiederholte Anwendung der Bildungsregeln erstellen kann

Strukturelle Induktion:
Eine Aussage der Form $\forall x \in M(E)$ beweist man wie folgt mit **struktureller Induktion**:

1. **Induktionsanfang**: Zeige $\forall x \in M_G(E)$ für die Teilmenge der Grundelemente M_G von M
2. **Induktionsschritt**: Zeige ($x_1, \ldots, x_n$ erfüllen E) $\Rightarrow$ (jedes x, das aus $x_1, \ldots, x_n$ mit einer der Bildungsregeln erstellt werden kann, erfüllt E)
   Die Aussage ($x_1, \ldots, x_n$ erfüllen E) nennt man **Induktionsvoraussetzung (IV)**

10.8.4 Zahlenvergleich

Seien A und B Terme und p eine aussagenlogische Formel, für die wir $p \Rightarrow (A = B)$ zeigen wollen.

Hierfür wollen wir zwei gängige Vorgehensweisen vorstellen: Die der *Gleichheitskette* und der *Zerlegung in Ungleichungen.*

1. Gleichheitsketten:
Erstelle eine Kette von Termen $A = T_1, T_2, \ldots, T_{n-1}, T_n = B$, so dass

- $T_1 = T_2 = \ldots = T_n$, wobei
- $T_i = T_{i+1}$ für jedes i aus einer Definition, aus einer bereits bewiesenen Aussage, aus p oder einer bekannten Rechenregel folgt

Jedes $=$-Zeichen der Rechnung, das aus einer Definition, einer Aussage oder p folgt, wird mit darübergestelltem Verweis begründet

Das Vorgehen ist, wie eventuell schon bemerkt wurde, sehr ähnlich zur Implikationskette. Ein großer Unterschied ist, dass wir noch die linke Seite der Implikation für unseren Beweis nutzen können. Das heißt, um $A = B$ zu beweisen, dürfen wir beliebige Teilformeln aus p nutzen.

Beispiel: 10.45

Beweise:

$$\underbrace{(x \geqslant 0 \wedge c_{1K,n}(x) = b_{n-1} \ldots b_0)}_{p} \Rightarrow (\underbrace{c_{1K,n}(-x)}_{A} = \underbrace{(1 - b_{n-1}) \ldots (1 - b_0))}_{B}$$

Es gilt:

$$\begin{aligned} c_{1K,n}(-x) &\overset{(1)}{=} c_{2,n}(2^n - 1) - c_{2,n}(x) \\ &\overset{(2)}{=} 1^n - b_{n-1} \dots b_0 \\ &\overset{(3)}{=} (1 - b_{n-1}) \dots (1 - b_0) \end{aligned}$$

- (1): $x \geqslant 0$ und Definition von $c_{1K,n}$
- (2): $c_{1K,n}(x) = b_{n-1} \dots b_0$
- (3): lässt sich ziffernweise schreiben als

2. Zerlegung in Ungleichungen:
Nun nutzen wir aus, dass wir eine Formel umschreiben können (ähnlich wie bei der Kontraposition): $(A = B) \Leftrightarrow ((A \leqslant B) \wedge (A \geqslant B))$. Also zeige $p \Rightarrow (A \leqslant B)$ und $p \Rightarrow (A \geqslant B)$ und benutze ggf. für jede der beiden Aussagen eine Ungleichungskette von Termen
$A = T_1 \leqslant T_2 \leqslant \dots \leqslant T_n = B$ bzw. $A = S_1 \geqslant S_2 \geqslant \dots \geqslant S_n = B$.

Durch diese Zerlegung kann ein Problem vereinfacht werden, so dass wir es mithilfe einer Gleichheitskette lösen können.

Jedes $\leqslant$- und $\geqslant$-Zeichen der Rechnung wird wieder mit darübergestelltem Verweis begründet, falls es nicht aus einer bekannten Rechenregel folgt

Beispiel: 10.46

Beweise: $(u < s \wedge s \leqslant o \wedge u = o - 1) \Rightarrow (s = o)$ für Variablen u, s, o vom Typ $\mathbb{N}$

Dieses Problem können wir wieder umschreiben:

$$(u < s \wedge s \leqslant o \wedge u = o - 1) \Rightarrow (s \leqslant o) \tag{1}$$

und

$$(u < s \wedge s \leqslant o \wedge u = o - 1) \Rightarrow (s \geqslant o) \tag{2}$$

Formel (1) gilt per Voraussetzung: $s \leqslant o$

Formel (2) wird per Gleichheitskette gelöst: $s \overset{u<s}{\geqslant} u + 1 \overset{u=o-1}{=} o$

10.8.5 Vollständige Induktion

Die vollständige Induktion sollte schon aus *Mathematik für Informatiker* bekannt sein. Sie wird für Zahlenreihen in den natürlichen Zahlen verwendet.

Sei n eine Variable vom Typ $\mathbb{N}$, $k \in \mathbb{N}$ und $A(n)$ eine prädikatenlogische Formel mit freier Variable n (für jedes $n \in \mathbb{N}$)

Vollständige Induktion:
Eine Aussage der Form $\forall n \in [k, \infty[\ (A(n))$ beweist man wie folgt mit **vollständiger Induktion**:

1. **Induktionsanfang**: Zeige $A(k)$
2. **Induktionsschritt**: Zeige $A(n) \Rightarrow A(n+1)$
   Die Aussage $A(n)$ nennt man **Induktionsvoraussetzung (IV)**

> Beispiel: 10.47
>
> Beweise $\forall n \in [1, \infty[\ (\sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2})$
>
> 1. **Induktionsanfang**: Zeige $A(1)$: $\sum_{i=1}^{1} i = 1 = \frac{1 \cdot (1+1)}{2}$
> 2. **Induktionsschritt**: Zeige $A(n) \Rightarrow A(n+1)$: $\sum_{i=1}^{n+1} i = (n+1) + \sum_{i=1}^{n} i \overset{A(n)}{=} (n+1) + \frac{n \cdot (n+1)}{2} = \frac{(n+1) \cdot (n+2)}{2}$

Im Induktionsschritt ist es immer das Ziel, die Aussage so umzuformen, dass die Induktionsvoraussetzung sich einsetzen lässt. Im Beispiel haben wir die Summe $\sum_{i=1}^{n+1} i$, welche in der Induktionsvoraussetzung von $i = 1$ bis n läuft. Man macht sich dann zu Nutze, dass man sie auch als $\sum_{i=1}^{n} i + (n+1)$ schreiben kann[6]. Somit hat man genau die Aussage aus der Induktionsvoraussetzung $\sum_{i=1}^{n} i$ da stehen und kann diese durch den Bruch $\frac{n \cdot (n+1)}{2}$) aus der Induktionsvoraussetzung ersetzen.

10.8.6 Mengenvergleich

Wir wollen nun zwei verschiedene Dinge für Mengen zeigen: Die *Teilmengenbeziehung* und *Mengengleichheit.*

1. Teilmengenbeziehung
Seien X und Y Mengen, für die wir $X \subseteq Y$ zeigen wollen.

Elementprinzip:
Zeige $\forall x((x \in X) \Rightarrow (x \in Y))$ oder $\forall x((x \notin Y) \Rightarrow (x \notin X))$

> Beispiel: 10.48
>
> **Beweise**: (x prim $\wedge x > 2$) $\Rightarrow$ (x ungerade)

[6] Falls das für Sie nicht klar ist, setzen Sie einfach mal $n = 3$ ein und schreiben Sie die Summe als $1 + 2 + ...$

Dafür nutzen wir die zweite Art des Elementprinzips und negieren die Formel. Somit müssen wir zeigen:

$$\neg(x \text{ ungerade}) \Rightarrow \neg(x > 2) \vee \neg(x \text{ prim})$$

Es gilt:

$$\begin{aligned}\neg(x \text{ ungerade}) &\overset{(1)}{\Rightarrow} x = 2 \vee \exists n \geqslant 2 (x = 2 \cdot n) \\ &\overset{(2)}{\Rightarrow} \neg(x > 2) \vee \neg(x \text{ prim})\end{aligned}$$

- (1): Definition von geraden Zahlen.
  Man müsste den Fall $x = 2$ nicht extra herausziehen, aber es hilft uns im nächsten Schritt.
- (2): Definition von Primzahlen.
  Den vorderen Teil kann man einfach abschwächen: Wenn $x = 2$, dann gilt offensichtlich $\neg(x > 2)$.
  Für den hinteren Teil reicht es aus, zu wissen, dass eine Primzahl, die größer als 2 ist, nicht durch 2 teilbar sein kann. Dass wir eine Zahl $x > 2$ also als $x = 2 \cdot n$ (für $n \geqslant 2$) schreiben können, sagt uns bereits, dass die Zahl x keine Primzahl sein kann.

Anmerkung: Der Vergleich von Mengen wurde hier auf die Implikation von Formeln zurückgeführt, da die Elementbeziehung einer einstelligen Relation entspricht

2. Mengengleichheit
Seien X und Y Mengen, für die wir $X = Y$ zeigen wollen.

Elementprinzip:
Zeige $\forall x((x \in X) \Leftrightarrow (x \in Y))$

Beispiel: 10.49

Beweise das Distributivgesetz für Mengen X, Y, Z:

$$(X \cup Y) \cap Z = (Y \cap Z) \cup (X \cap Z)$$

Es gilt:

$$\begin{aligned}(x \in (X \cup Y) \cap Z) &\overset{(1)}{\Leftrightarrow} (x \in X \cup Y \wedge x \in Z) \\ &\overset{(2)}{\Leftrightarrow} ((x \in X \vee x \in Y) \wedge x \in Z) \\ &\overset{(W)}{\Leftrightarrow} (x \in X \wedge x \in Z) \vee (x \in Y \wedge x \in Z) \\ &\overset{(1),(2)}{\Leftrightarrow} (x \in (Y \cap Z) \cup (X \cap Z))\end{aligned}$$

- (1): Definition Mengenschnitt
- (2): Definition Mengenvereinigung
- (W): Wahrheitstafel

Anmerkung: Mengenoperationen kann man also auf aussagenlogische Formeln zurückführen

10.8.7 Beweis durch Gegenbeispiel

Der Beweis durch Gegenbeispiel unterscheidet sich von den anderen, bereits kennengelernten Beweistechniken. Wir beweisen damit nicht im eigentlichen Sinne, dass etwas gilt, sondern das etwas **nicht** gilt.

Hier gilt es zu beachten, dass der Beweis durch Gegenbeispiel nur gültig ist, wenn wir beweisen wollen, dass etwas nicht gilt. Möchten wir beweisen, dass etwas gilt, reicht es nicht, nur ein Beispiel dafür angeben.

Um etwas zu widerlegen, müssen wir nur ein Gegenbeispiel finden.

Beispiel:

Widerlege: Alle natürlichen Zahlen sind durch 2 teilbar.

Gegenbeispiel: 5 ist eine natürliche Zahl. Es gilt jedoch: $5 \bmod 2 = 1$. Somit ist die 5 nicht durch 2 teilbar, weshalb nicht alle natürlichen Zahlen durch 2 teilbar sind.