Grundlagen der Programmierung

Dr. Christian Herzog
Technische Universität München

Wintersemester 2006/2007

Kapitel 8: Information und Repräsentation

Überblick über dieses Kapitel

- Repräsentation, Information, Interpretation
- Informationssystem
- Beispiel: Boolesche Algebra
  - Boolesche Terme, Boolesche Algebra
  - Interpretation Boolescher Terme
  - Gesetze, Semantische Äquivalenz
  - Mengenalgebra
- Signatur
- Abstrakte Algebra, abstrakte Klasse

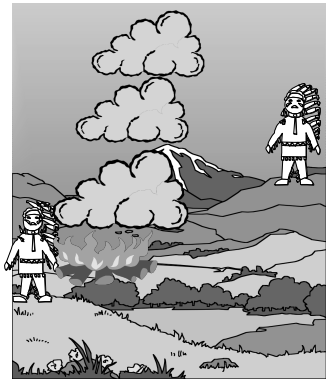
Motivation

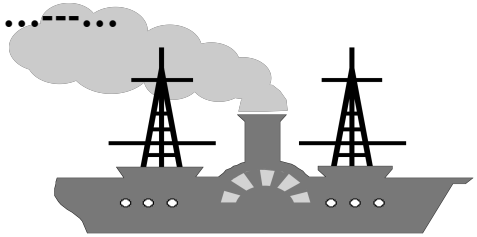
- Bisher haben wir hauptsächlich die Erstellung eines Informatiksystems aus einer Problemstellung im Fokus gehabt.
  - ingenieursmäßige Vorgehensweise eines Informatikers
  - ***prozedurales Wissen***
- Nun betrachten wir stärker das ***deklarative Wissen*** eines Informatikers, also das Wissen um die (theoretischen) Konzepte, die hinter seiner Vorgehensweise stehen.
- Wir suchen nach formalen (mathematischen) Beschreibungen für Begriffe wie
  - Halten von Information
  - Verarbeiten von Information
  - System und Systemschnittstelle

Repräsentation versus Information

- Wir haben bisher Zeichenketten für Strichzahlen betrachtet:
  IIII + II = IIIIII
  <IIII> * <II> = <IIIIIIII>
- Warum haben wir zwei verschiedenen Repräsentationen gewählt?
- Was bezeichnen wir mit III und <III>? Die Zahl 3?
- Es ist schwer, die Zahl 3 zu beschreiben, denn auch 3 ist eine Repräsentation!
- Plato würde sagen: Die natürlichen Zahlen, von denen "3" ein Mitglied ist, sind eine Idee (ähnlich wie "Schönheit" eine Idee ist).
  - Ideen *existieren*, die natürlichen Zahlen sind genauso real wie dieser Tisch hier, aber man kann nicht auf sie zeigen, man kann nur ihren Schatten sehen (Plato's Höhlengleichnis)
  - III, <III> und 3 sind verschiedene ***Schatten*** der ***Idee*** "Zahl 3".
    - Für ***Schatten*** sagen wir heute ***Repräsentation***
    - Für ***Idee*** benutzen wir heute den Begriff der ***Information***.

Information: Die Bedeutung einer Repräsentation





Drei verschiedene Repräsentationen derselben Information:
„Ich bin in Not. Ich brauche sofort Hilfe!“

Nachricht, Repräsentation, Information

- ***Signal***: Die Darstellung einer Mitteilung durch die zeitliche Veränderung einer physikalischen Größe.
  - Akustische Welle, Lichtwelle, elektromagnetische Welle.
- ***Inschrift***: Die dauerhafte Darstellung einer Mitteilung auf einem physikalischem Medium (Schriftträger).
  - Aufzeichnung auf einer CD, auf einem Stück Papier.
- ***Nachricht***: Eine Mitteilung, bei der wir von dem Übertragungsmedium und der Darstellung durch Signale oder Inschriften abstrahieren.
  - Es interessiert uns bei der Mitteilung nicht, ob sie per Fax, E-Mail oder mit der normalen Post gekommen ist.
- ***Repräsentation***: Die äußere Form einer Nachricht
- ***Information***: Der abstrakte Gehalt einer Nachricht. Auch: Die Bedeutung (Semantik) einer Nachricht.

Formen von Nachrichten

- Repräsentation ist die äußere Form der Darstellung einer Nachricht. Mögliche Formen sind
  - ***Ausdrücke*** (z.B. <IIII>, IIII, 3)
  - ***Graphiken*** (z.B. Klassendiagramme, Instanzdiagramme)
  - ***Anweisungen*** (z.B. Student s = new Student(“Viola“, 12345);)
  - ***Aussagen*** (z.B. "Heute ist Sonntag", "Heute ist Montag")
  - ***Piktogramme*** oder ***Ikonen*** (z.B. 

)

- Hinter jeder Repräsentation müssen wir die eigentliche Information finden.

Repräsentation versus Information

- Definitionen:
  - **Information** ist der abstrakte Gehalt (Bedeutung, Semantik) einer Nachricht.
  - **Repräsentation** ist die äußere Form einer Nachricht (Syntax).
- Repräsentationen sind oft mehrdeutig:
  - “Ilse liebt Äpfel.” und “Äpfel liebt Ilse.” (Im Zusammenhang jeweils eindeutig)
  - Aber: “Hans liebt Ilse.” Wer liebt hier wen?
- In der Informatik wünschen wir uns eindeutige Repräsentationen.

Rechnen mit Repräsentationen

- Wir verwenden Repräsentationen, um zu *rechnen*.
- **Beispiel**: schriftliches Addieren

- Dezimalzahlen:

```
  472
+ 193
  1
-----
  665
```

- Dualzahlen:

```
  111
+  10
   1
------
 1001
```

- Römische Zahlen:

```
  MLXIV
+  LXXI
-------
  ?????
```

- Nicht alle Repräsentationsformen sind gleich gut geeignet!

Interpretation, Informationsbezugssystem

- **Definition Interpretation:** Der Vorgang der Ermittlung der Information aus einer Repräsentation. Geschieht normalerweise
  - **Statisch durch ein Wörterbuch:** Auflistung Repräsentation ↔ Information
  - oder **prozedural durch eine Interpretationsvorschrift**, die für eine Repräsentation die jeweilige Information ermittelt.
- **Definition Informationsbezugssystem:** Die Menge der Gegenstände und Beziehungen zwischen Gegenständen, die nötig sind, um aus einer Repräsentation die Information zu ermitteln.
- Das Informationsbezugssystem wird oft nicht explizit genannt, da es aus dem Zusammenhang („Kontext") klar wird. Beispiele:
  - 79° bedeutet 79° Fahrenheit, wenn ich in den USA bin.
  - 79° bedeutet 79° Celsius, wenn ich in Italien bin.
  - „*Was ist deine Position?*" bedeutet
    - in einem Navigationssystem: „*Was sind deine x-/y-/z-Koordinaten?*"
    - in einem Organisationssystem: „*Was ist dein Rang in der Hierarchie?*"

Interpretation als Abbildung

- Wir modellieren den Begriff Interpretation jetzt als mathematische Abbildung:
  - Sei R eine Menge von Repräsentationen, und A eine Menge von Informationen.
  - Die **Interpretation** ist eine Abbildung **I: R → A** , die der gegebenen Repräsentation r ∈ R eine Information I[r] ∈ A zuordnet.

Hier verwendet man eckige statt runder Klammern!

- **Definition:** Wir bezeichnen (A, R, I) auch als **Informationssystem,** R heißt **Repräsentationssystem**, A nennt man **semantisches Modell**.
- Beispiel:
  - Sei A die Menge der natürlichen Zahlen
  - Dann sind {1, 2, 3, …}, {I, II, III, …} Elemente von Repräsentationssystemen R für A.

Aussagen

- Eine wichtige Form der Repräsentation von Information sind Aussagen.
- Beispiele von Aussagen:
  - „Die Sonne scheint."
  - „Das Oktoberfest ist immer im Januar."
  - „Es ist 79 Grad."
- Aussagen beziehen sich immer auf ein bestimmtes System, d.h. die Komponenten und ihre Beziehungen untereinander.
  - **Definition:** Wir nennen das System im Zusammenhang mit Aussagen das **Bezugssystem**.
- Für ein festes Bezugssystem ist eine Aussage entweder wahr oder falsch.
- Aussagen repräsentieren Wahrheitswerte.
- Wir werden jetzt ein sehr eingeschränktes Repräsentationssystem definieren, in dem wir nur Aussagen einer speziellen Form zulassen, die wir **Boolesche Terme** nennen.

Boolesche Terme

- Gegeben sei eine Menge ID von Bezeichnern (Identifikatoren, Variablen).
  - Ein Bezeichner kann als Platzhalter für eine Aussage aufgefasst werden.
  - Wir werden Bezeichner in der Regel mit *x*, *y* oder *z* bezeichen.
- **Definition:** Die Menge der **Booleschen Terme mit Bezeichnern aus ID** ist dann induktiv wie folgt definiert:
  - **true** und **false** sind Boolesche Terme.
  - Alle Elemente von ID sind Boolesche Terme.
  - Ist *t* ein Boolescher Term, so ist auch $(\neg t)$ ein Boolescher Term.
  - Sind *t1* und *t2* Boolesche Terme, dann sind auch $(t1 \vee t2)$ und $(t1 \wedge t2)$ Boolesche Terme.
  - Sind *t1* und *t2* Boolesche Terme, dann sind auch $(t1 \Rightarrow t2)$ und $(t1 \Leftrightarrow t2)$ Boolesche Terme.

Beispiele von Booleschen Termen

- $(x \vee y)$ $(x \wedge y)$ (**true** $\vee$ **false**)
- $(((\neg x) \wedge y) \vee (y \wedge x))$

- Um unnötige Klammern zu vermeiden, führen wir Prioritäten von Operationen ein:
  - Der Negations-Operator ¬ bindet stärker als die binären Operationen $\wedge$, $\vee$, $\Rightarrow$ und $\Leftrightarrow$
  - Der Operator $\wedge$ bindet stärker als $\vee$.
  - Am schwächsten binden die Operatoren $\Rightarrow$ und $\Leftrightarrow$

- $x \vee y$ $x \wedge y$ **true** $\vee$ **false**
- $(\neg x \wedge y) \vee (y \wedge x)$
- $\neg x \wedge y \vee y \wedge x$

Die Boolesche Algebra der Wahrheitswerte

- Boolesche Terme kann man rein syntaktisch sehen.
- Der Hauptzweck von Booleschen Termen ist aber die Repräsentation von Information.
  - Man kann nämlich jedem Booleschen Term einen Wahrheitswert als Information zuordnen.
- Um das zu machen, brauchen wir allerdings zunächst einmal einen präzisen Begriff für die ***Interpretation von Booleschen Termen***.
  - Zur Auffrischung: Die **Interpretation I : R → A** ordnet der Repräsentation r eine Information I[r] aus dem semantischem Modell A zu.
- Unser semantisches Modell A ist die Menge der Wahrheitswerte
  - Als semantisches Modell A für Wahrheitswerte kann man im Prinzip jede Menge mit zwei Elementen verwenden.
  - Wie verwenden in dieser Vorlesung die Menge **B** = {O, L},
  - O und L bezeichnen wir als ***Boolesche Wahrheitswerte***.

Wir können Operationen auf diesen Booleschen Wahrheitswerten definieren

not: B → B	**or: B × B → B**	**and: B × B → B**	**impl: B × B → B**
not(O) = L	**or**(O, O) = O	**and**(O, O) = O	**impl**(O, O) = L
not(L) = O	**or**(O, L) = L	**and**(O, L) = O	**impl**(O, L) = L
	or(L, O) = L	**and**(L, O) = O	**impl**(L, O) = O
	or(L, L) = L	**and**(L, L) = L	**impl**(L, L) = L

equiv: B × B → B

equiv(O, O) = L
equiv(O, L) = O
equiv(L, O) = O
equiv(L, L) = L

- Uns interessiert besonders die Menge der booleschen Wahrheitswerte *zusammen* mit diesen Operationen.
- Eine solche Struktur kennen wir schon: **Klasse** als Zusammenfassung von Attributen und Operationen.
- In der Mathematik heißt diese Struktur **Algebra**, in der Informatik allgemein **Rechenstruktur.**
- Die Menge **B** mit den Operationen **not**, **or**, **and**, **impl** und **equiv** nennen wir **Boolesche Algebra**.

Boolesche Belegung

- Wir sind auf der Suche nach einer Interpretation I, die jedem Booleschen Term einen Wahrheitswert aus **B** zuordnet.
- Boolesche Terme sind Aussagen, die Identifikatoren enthalten.
- Der Wahrheitswert von Booleschen Termen wie z.B. $\neg x \wedge y \vee y \wedge x$ hängt von der *Belegung* der Identifikatoren x und y ab.

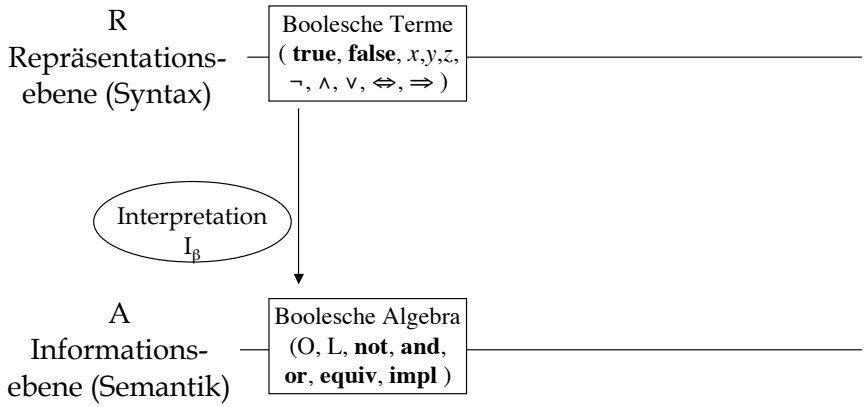
- **Definition:** Eine Abbildung β: ID $\rightarrow$ **B**, die jedem Bezeichner aus einer Menge ID von Identifikatoren einen Wahrheitswert zuordnet, heißt **Boolesche Belegung.**

- **Beispiel einer Belegung β für ID = {*x, y, z*}:**
  - $\beta(x) = L$
  - $\beta(y) = O$
  - $\beta(z) = L$
- **Beispiel einer alternativen Belegung β_1 für ID = {*x, y, z*}:**
  - $\beta_1(x) = O$
  - $\beta_1(y) = O$
  - $\beta_1(z) = L$

Boolesche Belegung ...

- Die Menge aller Belegungen heißt ENV. Also: $\beta, \beta_1 \in ENV$.
- Mit einer gegebenen Belegung β lässt sich jedem Bezeichner aus ID und damit jedem Booleschen Term ein Wahrheitswert zuordnen.
- Wir definieren dafür die Interpretation I_β

Kapitelnavigator: wo stehen wir?



Interpretation I_β von Booleschen Termen

- I_β[**true**] = L I_β[**false**] = 0
- $I_\beta[x] = \beta(x)$
  - d.h. für alle Elemente aus ID ist der Wahrheitswert der Wert, der sich aus der Belegung β ergibt.
- $I_\beta[\neg t] =$ **not** ($I_\beta[t]$)
  - d.h. wir finden den Wahrheitswert von $\neg t$, indem wir den Wahrheitswert vom Term *t* finden, und ihn dann negieren.
- $I_\beta[\ t1 \vee t2] =$ **or** ($I_\beta[t1]$, $I_\beta[t2]$)
  - d.h. wir finden den Wahrheitswert von $t1 \vee t2$, indem wir die Wahrheitswerte der Terme *t1* und *t2* finden, und sie dann mit **or** verknüpfen.
- $I_\beta[\ t1 \wedge t2] =$ **and** ($I_\beta[t1]$, $I_\beta[t2]$)
- $I_\beta[\ t1 \Rightarrow t2] =$ **impl** ($I_\beta[t1]$, $I_\beta[t2]$)
- $I_\beta[t1 \Leftrightarrow t2] =$ **equiv** ($I_\beta[t1]$, $I_\beta[t2]$)

Alles klar?

Beispiel: Interpretation eines Booleschen Terms

- Gegeben sei wieder folgende Belegung β: $\beta(x) = L$, $\beta(y) = O$, $\beta(z) = L$

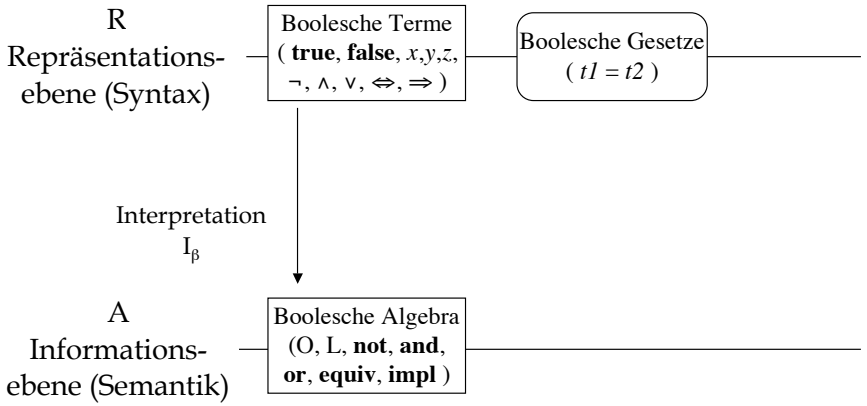
- Damit berechnen wir den Wahrheitswert des Terms $(\neg x \vee y) \wedge (y \vee z)$:

Folie 16

$I_\beta[(\neg x \vee y) \wedge (y \vee z)]$
= **and** ($I_\beta[\neg x \vee y]$, $I_\beta[y \vee z]$)
= **and** (**or** ($I_\beta[\neg x]$, $I_\beta[y]$) , **or** ($I_\beta[y]$, $I_\beta[z]$))
= **and** (**or** (**not** ($I_\beta[x]$) , $I_\beta[y]$) , **or** ($I_\beta[y]$, $I_\beta[z]$))
= **and** (**or** (**not** ($\beta(x)$)) , $\beta(y)$) , **or** ($\beta(y)$, $\beta(z)$))
= **and** (**or** (**not** (L) , O) , **or** (O , L))
= **and** (**or** (O , O) , L)
= **and** (O , L) = O

Kapitelnavigator

Wichtiger Punkt: In Informatiksystemen geschieht Informationsverarbeitung immer auf der Repräsentationsebene!



Beispiele von Booleschen Gesetzen

true = $x \vee \neg x$		Gesetz für **true**
false = ¬**true**		Gesetz für **false**
$\neg\neg x = x$		Involutionsgesetz
$x \wedge y = y \wedge x$	$x \vee y = y \vee x$	Kommutativgesetz
$(x \wedge y) \wedge z = x \wedge (y \wedge z)$	$(x \vee y) \vee z = x \vee (y \vee z)$	Assoziationsgesetz
$x \wedge x = x$	$x \vee x = x$	Idempotenzgesetz
$x \vee (y \wedge \neg y) = x$	$x \wedge (y \vee \neg y) = x$	Neutralitätsgesetz
$x \wedge (x \vee y) = x$	$x \vee (x \wedge y) = x$	Absorptionsgesetz
$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$	Distributivgesetz
$\neg(x \wedge y) = \neg x \vee \neg y$	$\neg(x \vee y) = \neg x \wedge \neg y$	deMorgan's Gesetz

Semantische Äquivalenz

- **Definition**: In einem Informationssystem (A, R, I) heißen zwei Repräsentationen r1, r2 ∈ R **semantisch äquivalent**, wenn sie mittels der Interpretation I auf dasselbe Element des semantischen Modells A abgebildet werden, d.h. wenn gilt: I[r1] = I[r2].
- **Satz**: In allen Booleschen Gesetzen sind die linken und rechten Seiten der Gleichung semantisch äquivalent, d.h. die Gesetze sind verträglich mit der Interpretation.
- **Konsequenz**: Wir dürfen mit den Booleschen Termen im Repräsentationssystem R *rechnen*
  - die Booleschen Gesetze können als *Rechenregeln* aufgefasst werden, mit denen komplizierte Boolesche Terme umgeformt werden können
  - Das Ziel ist, sie auf **true** oder **false** zu vereinfachen (zu reduzieren).
- Die semantische Äquivalenz garantiert uns, dass dabei die Wahrheitswerte der Terme erhalten bleiben.

Nachweis der semantischen Äqivalenz

Um den Satz auf der vorherigen Folie zu beweisen, müssen wir zeigen, dass für jedes Boolesche Gesetz *t1 = t2* - d.h. für alle Gesetze auf Folie 24 - gilt: $I_\beta[t1] = I_\beta[t2]$ und dies ***für jede Belegung β***.

➢ Wir beweisen dies beispielhaft am Absorptionsgesetz $x \wedge (x \vee y) = x$

Wahrheitstabelle

$\beta(x)$	$\beta(y)$	$I_\beta[x \vee y]$	$I_\beta[x \wedge (x \vee y)]$
O	O	O	O
O	L	L	O
L	O	L	L
L	L	L	L

Schreibvereinfachung: Weglassen der Belegung β und Interpretation I_β:

Wahrheitstabelle

x	y	$x \vee y$	$x \wedge (x \vee y)$
O	O	O	O
O	L	L	O
L	O	L	L
L	L	L	L

Vereinfachung von Booleschen Termen

- Beispiel: $y \wedge \neg y$
- Umformung:

$y \wedge \neg y$	$= \neg y \wedge y$	(Kommutativgesetz)
	$= \neg y \wedge \neg \neg y$	(Involutionsgesetz)
	$= \neg (y \vee \neg y)$	(deMorgan)
	$= \neg$ (**true**)	(Gesetz für **true**)
	= **false**	(Gesetz für **false**)

- Wir rechnen hier rein syntaktisch auf der Ebene der Repräsentationen, d.h. die Umformung geschieht nur unter Anwendung von Booleschen Gesetzen.

Implikation und Äquivalenz sind Abkürzungen

- **Frage**: Warum kommen die Operationen ⇒ (Implikation) und ⇔ (Äquivalenz) in den Booleschen Gesetzen nicht vor?
- **Antwort**: ⇒ und ⇔ sind Abkürzungen, die sich mit ∧, ∨ und ¬ definieren lassen:

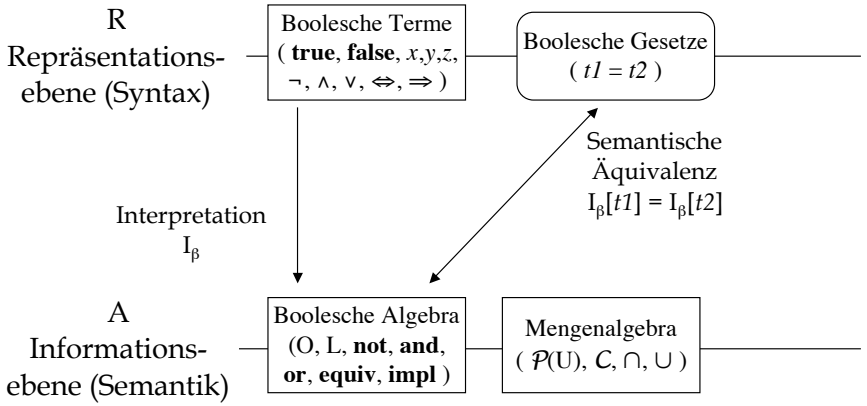
  $t1 \Rightarrow t2 \quad = \quad \neg t1 \vee t2$

  $t1 \Leftrightarrow t2 \quad = \quad (t1 \wedge t2) \vee (\neg t1 \wedge \neg t2)$
- Wir können noch einen Schritt weitergehen. Die Operation ∧ kann mit den Operationen ¬ und ∨ ausgedrückt werden:

  $t1 \wedge t2 \quad = \quad \neg (\neg t1 \vee \neg t2)$
- Umgekehrt kann ∨ mit den Operationen ¬ und ∧ ausgedrückt werden:

  $t1 \vee t2 \quad = \quad \neg (\neg t1 \wedge \neg t2)$

Kapitelnavigator: wo stehen wir?



Mengenalgebra

- Ist U eine beliebige Menge, dann bezeichnen wir mit $\mathcal{P}(U)$ die Menge aller Teilmengen von U.
- $\mathcal{P}(U)$ wird **Potenzmenge von U** genannt
  - und manchmal mit 2^U bezeichnet
- Betrachten wir folgende Operationen auf Teilmengen M, M_1, M_2 von U:
  - $C(M) = U \setminus M$, das Komplement der Teilmenge M bezüglich U
  - $M_1 \cap M_2$, der Durchschnitt der Teilmengen M_1 und M_2
  - $M_1 \cup M_2$, die Vereinigung der Teilmengen M_1 und M_2
- $\mathcal{P}(U)$ zusammen mit den Operationen C, $\cap$ und $\cup$ heißt **Mengenalgebra über U**.

Mengenalgebra: ein Beispiel

Setzen wir $U = \{1,2,3\}$.

Dann besteht $\mathcal{P}(U)$ aus den Mengen
$\varnothing$, {1}, {2}, {3}, {1,2}, {1,3}, {2,3} und {1,2,3}

Dann gilt unter anderem

$C(\{2\}) = U \setminus \{2\} = \{1,3\}$

$\{1,3\} \cap \{2,3\} = \{3\}$

$\{1,3\} \cup \{2,3\} = \{1,2,3\}$

$\{1,3\} \cap \varnothing = \varnothing$

$\{1,3\} \cup \varnothing = \{1,3\}$

$\{1,3\} \cap U = \{1,3\}$

$\{1,3\} \cup U = U$

$C(\varnothing) = U$ und $C(U) = \varnothing$

Eine alternative Interpretation von Booleschen Termen

- Bisher haben wir die Repräsentation Boolescher Terme als Wahrheitswerte interpretiert, d.h. ein Term war entweder 0 oder L.
- Jetzt wollen wir mal einen ***Booleschen Term*** nicht als Wahrheitswert sondern als ***Teilmenge von U*** interpretieren
  - Die Belegung β ist dann eine Abbildung β: ID $\rightarrow$ P(U), die jedem Bezeichner aus ID eine Teilmenge von U zuordnet.
- Wir definieren jetzt eine Interpretation $\hat{I}_\beta$ folgendermassen :

$\hat{I}_\beta[\textbf{true}] = U$ $\quad$ $\hat{I}_\beta[\textbf{false}] = \varnothing$

$\hat{I}_\beta[x] = \beta(x)$

$\hat{I}_\beta[\neg t] = C(\hat{I}_\beta[t])$

$\hat{I}_\beta[t1 \vee t2] = \hat{I}_\beta[t1] \cup \hat{I}_\beta[t2]$

$\hat{I}_\beta[t1 \wedge t2] = \hat{I}_\beta[t1] \cap \hat{I}_\beta[t2]$

$\hat{I}_\beta[t1 \Rightarrow t2] = C(\hat{I}_\beta[t1]) \cup \hat{I}_\beta[t2]$

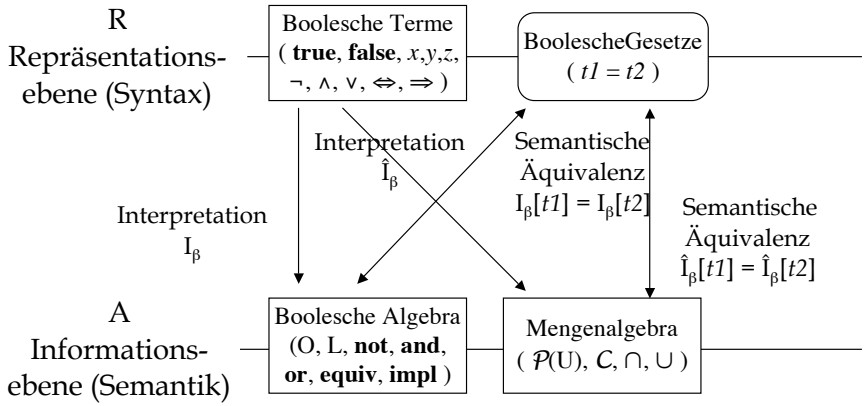
$\hat{I}_\beta[t1 \Leftrightarrow t2] = (\hat{I}_\beta[t1] \cap \hat{I}_\beta[t2]) \cup (C(\hat{I}_\beta[t1]) \cap C(\hat{I}_\beta[t2]))$

- Ist diese Interpretation $\hat{I}_\beta$ sinnvoll? Das sei erst einmal dahingestellt, wir können sie jedenfalls definieren. Betrachten wir es als Spiel.

Mengenalgebra und Boolesche Gesetze

- Auch wenn die Interpretation von Booleschen Termen wie ein Spiel mit Definitionen wirkt, gilt ein bemerkenswerter Satz:
- **Satz**: Wenn wir Boolesche Terme als Elemente einer Mengenalgebra interpretieren, dann gelten die Booleschen Gesetze, d.h. die linken und rechten Seiten jeder Gleichung sind semantisch äquivalent.
- Zum Beweis müssten wir wieder zeigen, dass für jedes Boolesche Gesetz $t1 = t2$ und jede Belegung β gilt: $\hat{I}_\beta[t1] = \hat{I}_\beta[t2]$
  - Dies wollen wir hier jedoch nicht durchführen.
- Wir haben also zwei völlig unterschiedliche semantische Modelle **B** und $\mathcal{P}(U)$ für Boolesche Terme, die beide mit den Booleschen Gesetzen verträglich sind:
  - **B** besteht nur aus den zwei Elementen O und L.
  - $\mathcal{P}(U)$ hat unendlich viele Elemente, wenn U unendlich ist.

Kapitelnavigator



Resümee

- Wir haben
  - formal festgelegt, wie Boolesche Terme aufgebaut werden,
  - Gesetze angegeben, welche Terme semantisch äquivalent sind,
- Wir haben außerdem zwei Algebren (Rechenstrukturen) angegeben, in denen Boolesche Terme unter Berücksichtigung der Booleschen Gesetze interpretiert werden können.
- Nun wollen wir im Rest dieses Kapitels vertiefen, dass
  - wir durch Angabe von zulässigen Termen und von Booleschen Gesetzen *abstrakt beschreiben*,
  - was in gegebenen Rechenstrukturen *konkret vorliegt*.
- Zunächst werden wir ein allgemeineres Konzept vorstellen, wie der Aufbau von Termen festgelegt wird
- Dazu führen wir den Begriff der *Signatur* ein.

Signatur

- **Definition Signatur**: Eine Signatur besteht aus
  - einer Menge S von **Sorten**;
    - Sorten sind Bezeichner für Mengen, auf denen die Operationen von Rechenstrukturen definiert sind, z.B. bool für **B** oder $\mathcal{P}$(U).
    - Wichtig: **B** oder $\mathcal{P}$(U) sind keine Bezeichner sondern Mengen, bool ist der Bezeichner!
    - Diese Mengen bezeichnet man auch als **Trägermengen**.
  - einer Menge F von **Funktionssymbolen**;
    - Funktionssymbole sind Bezeichner für Operationen bzw. Funktionen von Rechenstrukturen.
  - der Angabe einer **Funktionalität** für jedes Funktionssymbol.
    - Die Funktionalität gibt an, welche Sorten eine Operation auf welche Sorte abbildet.
    - Dabei wird auch vermerkt, wenn ein Funktionssymbol f nicht in Funktionsschreibweise f(a,b) sondern beispielsweise in Infix-Schreibweise a f b verwendet wird.

Beispiel: Eine Signatur für die boolesche Algebra

- Menge der Sorten: S = {bool}
- Menge der Funktionssymbole: F = {**true**, **false**, ¬, ∧, ∨}

  true und **false** sind nullstellige Funktionssymbole (Konstanten)
- Funktionalitäten:

  true: → bool

  false: → bool

  ¬: bool → bool // Schreibweise ohne Funktionsklammern

  ∧: bool × bool → bool // Infix-Operator

  ∨: bool × bool → bool // Infix-Operator

- **Bemerkung**: die Signatur legt fest, wie Terme aufgebaut werden.

Beispiel: Signatur für nat

- Menge der Sorten: S = {nat}
- Menge der Funktionssymbole: F = {zero, succ, add}
- Funktionalitäten:

  zero: → nat

  succ: nat → nat

  add: nat × nat → nat

- Terme über nat sind dann
  - zero, succ(zero), succ(succ(zero)), succ(succ(succ(zero)))
  - add(zero, zero), add(succ(succ(zero)), add(zero, zero))
- ... und, wenn man Bezeichner x, y zulässt, auch
  - succ(succ(succ(*x*))), add(*x*, add(succ(succ(*y*)), add(*x*, *y*)))

Abstrakte Algebra

- **Definition**: Eine Signatur Σ zusammen mit Gesetzen der Form *t1 = t2*, wobei *t1* und *t2* Terme über Σ (mit Identifikatoren) sind, nennen wir **abstrakte Algebra** oder **abstrakte Rechenstruktur**.
- Eine abstrakte Algebra beschreibt
  - die *Schnittstelle* einer Rechenstruktur (mittels der Signatur)
  - die *Spezifikation* der Operationen (mittels der Gesetze)

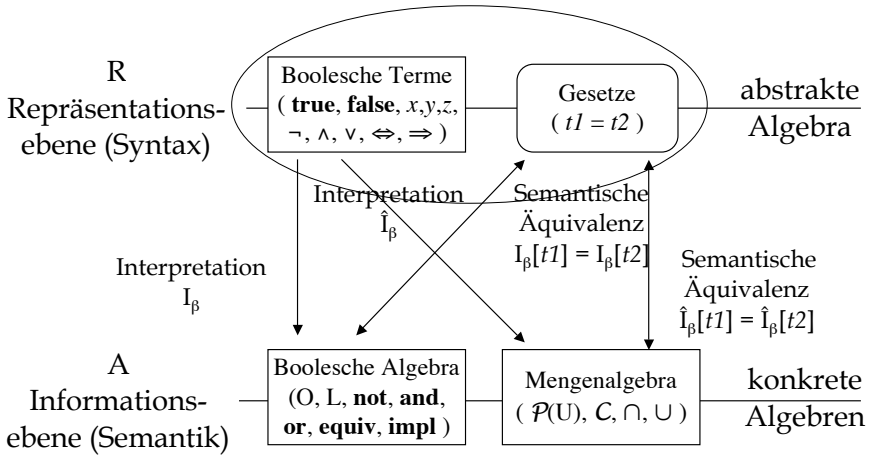
  ohne dass die Rechenstruktur konkret angegeben wird (d.h. es gibt keine *Implementation*).
- Aus der Mathematik sind uns abstrakte Strukturen vertraut. So beschreibt z.B. der Begriff Körper eine abstrakte Struktur (Trägermengen, Operationen, Gesetze), und die Körper der rationalen bzw. reellen Zahlen sind Konkretisierungen davon.

Beispiele von abstrakten Algebren

- Die Signatur für bool zusammen mit den Booleschen Gesetzen bildet die abstrakte Algebra **bool** der Wahrheitswerte.
- Die Signatur für nat und die Gesetze
  - add(*x*, zero) = *x*
  - add(*x*, succ(*y*)) = succ (add (*x*, *y*))

  ergibt die abstrakte Algebra **nat** der natürlichen Zahlen.

Graphische Kapitelübersicht: wo stehen wir?



Konkrete Algebra

- Gegeben sei eine abstrakte Algebra A mit Sorten S und Funktionssymbolen F in ihrer Signatur Σ.
- Eine Algebra K, die
  - zu jeder Sorte s aus S eine Trägermenge enthält und
  - zu jedem Funktionssymbol f aus F eine Operation bzw. Funktion enthält
  - die alle mit den Funktionalitäten aus Σ und den Gesetzen von A verträglich sind,

  heißt eine **Konkretisierung** oder **konkrete Algebra** zu A.
- **Beispiele**:
  - Boolesche Algebra und Mengenalgebra sind Konkretisierungen der abstrakten Algebra **bool**.
  - Die natürlichen Zahlen mit der Konstanten 0, dem Inkrement +1 und der Addition + sind eine Konkretisierung der abstrakten Algebra **nat**.

Eine abstrakte Algebra für den Keller

- Ein Keller (Stapel, Stack) speichert Daten in der Form, dass immer das zuletzt gespeicherte Datum als erstes wieder ausgelesen wird.
- Kellerartige Strukturen haben enorme Bedeutung in der Informatik
  - beim Speichern der Zwischenergebnisse von Ausdrücken
  - bei der Speicherverwaltung rekursiver Methoden
  - bei der Realiserung von LIFO-Strategien (*last in first out*)
- Eine abstrakte Algebra für Keller erlaubt es uns, die wesentlichen Eigenschaften eines Kellers präzise festzulegen (zu spezifizieren), ohne der Implementation vorwegzugreifen
  - Wir bezeichnen die abstrakte Algebra für den Keller mit **stack**
- **Zur Erinnerung:** Eine abstrakte Algebra (abstrakte Rechenstruktur) besteht aus einer ***Signatur Σ*** und ***einer Anzahl von Gesetzen***

Definition: abstrakte Rechenstruktur stack (Signatur Σ)

- **Menge der Sorten:** S = {*T,* stack *T,* bool}
  - *T* bezeichnet die Menge, aus der die im Keller zu speichernden Elemente kommen. (*T* ist wie ein Parameter. Die nähere Struktur von *T* wird erst bei der Implementation benötigt).
  - stack *T* bezeichnet die Menge aller Keller über *T*
  - bool bezeichnet die Rechenstruktur der Wahrheitswerte. Sie wird „importiert", d.h. ihre Funktionalitäten und Gesetze werden nicht noch einmal aufgeführt.
- **Menge der Funktionssymbole:** F = {create, isEmpty, push, top, pop}
- **Funktionalitäten:**

  create : → stack *T*
  isEmpty : stack *T* → bool
  push : stack *T* × *T* → stack *T*
  top : stack *T* → *T*
  pop : stack *T* → stack *T*

Fortsetzung: die abstrakte Rechenstruktur stack (Gesetze)

Gesetze für stack:

S1: isEmpty (create) = **true**
S2: isEmpty (push (*s*, *x*)) = **false**
S3: top (push (*s*, *x*)) = *x*
S4: pop (push(*s*, *x*)) = *s*

- Beispiele von syntaktisch korrekten Termen:

  create
  top (push (pop (push (push (*s*, *x*) , *y*)), *z*))
  top (create)
  - semantisch sinnlos, denn ein leerer Keller hat kein Element!

- Beispiel für eine Termvereinfachung gemäß der Gesetze:

  push (pop (push (push (s, *x*) , *y*)), *z*) ergibt
  push (push (*s*, *x*), *z*) nach Anwendung von Gesetz S4

pop(push(push(s, *x*), *y*)) = push (s, *x*)

Abstrakte Rechenstrukturen in Java

- Eine Klasse in Java kann als konkrete Rechenstruktur aufgefasst werden:
  - Trägermengen: Typen der Attribute sowie der Ergebnisse und Parameter von Methoden
  - Operationen: Methoden
- Eine direkte Entsprechung einer abstrakten Rechenstruktur gibt es in Java nicht.
- Es gibt jedoch in Java **abstrakte Klassen**, die den Signaturen abstrakter Rechenstrukturen entsprechen.
  - Methoden abstrakter Klassen haben keinen Rumpf
  - Abstrakte Klassen treten als Oberklassen auf, konkretisiert werden sie in Unterklassen, in denen die Rümpfe implementiert sind.
  - Verschiedene Konkretisierungen sind damit möglich.
  - Abstrakte Klassen können nicht instantiiert werden.
  - Anwendungen können bereits mit der Schnittstelle abstrakter Klassen arbeiten.
- Näheres im Kapitel über den objektorientierten Programmierstil.

Zusammenfassung

- Repräsentation, Information, Interpretation
- Boolesche Terme
- Interpretation von Booleschen Termen
- Eine besondere Gleichheit: Semantische Äquivalenz
- Boolesche Gesetze
- Vereinfachung von Booleschen Termen
- Mengenalgebra als alternatives semantisches Modell
- Abstrakte Algebra, abstrakte Rechenstruktur
- Abstrakte Klassen in Java