3 Terme und Algebren
3.1 Terme

In allen formalen Kalkülen benutzt man **Formeln als Ausdrucksmittel**.
Hier betrachten wir **nur ihre Struktur - nicht ihre Bedeutung**. Wir nennen sie **Terme.**

Terme bestehen aus **Operationen**, **Operanden**, **Konstanten und Variablen**:

a + 5 blau ? gelb = grün ♥ > ♦

Terme werden nicht „ausgerechnet“.
Operationen, Konstanten und Variablen werden als **Symbole ohne Bedeutung** betrachtet.

Notation von Termen:

Infix-, **Postfix-**, **Präfix-** und **Baum-**Form

Umformung von Termen:

Grundlage für die Anwendung von Rechenregeln, Gesetzen

Für **Variable** in Termen werden Terme **substituiert:**

in $a + a = 2*a$ substituiere a durch $3*b$ $3*b + 3*b = 2*3*b$

Unifikation: Terme durch Substitution von Variablen gleich machen,
z. B. um die Anwendbarkeit von Rechenregeln zu prüfen

Vorlesung Modellierung WS 2011/12 / Folie 301

Ziele:

Informelle Übersicht

in der Vorlesung:

Erläuterungen dazu

- Terme als Strukturen erklären.
- Terme umformen ohne zu "rechnen".

Sorten und Signaturen

Terme werden zu einer **Signatur** gebildet.
Sie legt die verwendbaren Symbole und die Strukturierung der Terme fest.

Signatur Σ := (S, F), S ist eine Menge von **Sorten**, F ist eine Menge von **Operationen**.

Eine **Sorte s** $\in$ **S** ist ein **Name für eine Menge von Termen**, z. B. ARITH, BOOL;
verschiedene Namen benennen disjunkte Mengen

Eine **Operation f** $\in$ **F** ist ein **Operatorsymbol**, beschrieben durch
Anzahl der Operanden (**Stelligkeit)**,
Sorten der Operanden und **Sorte des Ergebnisses**

0-stellige Operatoren sind Konstante, z. B. true, 1

Beispiele:

einzelne Operatoren:

Name	**Operandensorten**	**Ergebnissorte**
+:	ARITH x ARITH	-> ARITH
<:	ARITH x ARITH	-> BOOL
$\wedge$:	BOOL x BOOL	-> BOOL
true:		-> BOOL
1:		-> ARITH

Signatur Σ_{BOOL} := (S_{BOOL}, F_{BOOL})

S_{BOOL} := { BOOL },
F_{BOOL} :=
{ true: -> BOOL,
false: -> BOOL,
$\wedge$: BOOL x BOOL -> BOOL,
$\neg$: BOOL -> BOOL
}

Vorlesung Modellierung WS 2011/12 / Folie 302

Ziele:

Begriff der Signatur verstehen

in der Vorlesung:

- Erläuterung der Begriffe
- Beispiele für Terme zu Signaturen
- Hinweis: Der Name Signatur wird 2-fach verwendet: wie hier definiert und als "Signatur einer Funktion (siehe Folie Mod-2.11)".

Korrekte Terme

In **korrekten Termen** muss jeweils die Zahl der Operanden mit der **Stelligkeit** der Operation und die **Sorten** der Operandenterme mit den Operandensorten der Operation übereinstimmen.

Induktive Definition der **Menge τ der korrekten Terme der Sorte s zur Signatur Σ = (S, F)**:
Sei die Signatur Σ = (S, F). Dann ist t ein **korrekter Term der Sorte $s \in S$**, wenn gilt

- t = **v** und v ist der **Name einer Variablen** der Sorte s, oder
- t = **f (t_1, t_2, .., t_n)**, also die **Anwendung einer n-stelligen Operation**
  f: s_1 x s_2 x ... x s_n -> s $\in$ F
  wobei jedes **t_i ein korrekter Term der Sorte s_i** ist
  mit $n \geq 0$ (einschließlich Konstante f bei n = 0) und $i \in \{1, ..., n\}$

f (t_1,..., t_n) ist ein **n-stelliger Term**; die t_i sind seine **Unterterme**.

Korrekte Terme, die **keine Variablen** enthalten, heißen **Grundterme**.

Beispiele: korrekte Terme zur Signatur Σ_{BOOL}:
false $\neg$ true true $\wedge$ x $\neg$(a $\wedge$ b) x $\wedge$ $\neg$ y

nicht korrekt: a $\neg$ b $\neg$ ($\wedge$ b)

Vorlesung Modellierung WS 2011/12 / Folie 303

Ziele:

Regeln zur Struktur von Termen

in der Vorlesung:

- Terme zu den Signaturen von Mod-3.2 konsturieren.
- Beispiele für falsche Terme.
- Vergleich mit Typregeln in Programmiersprachen.

Verständnisfragen:

- Welche Terme kann man aus den Operationen 0: -> N0 und succ: N0 -> N0 bilden?
- Geben Sie einige Terme zu den Signaturen von Mod-3.2 an.

Notationen für Terme

Notation eines n-stelligen Terms mit Operation (Operator) f und Untertermen $t_1, t_2, ..., t_n$:

Bezeichnung	Notation	Beispiele
Funktionsform:	Operator **vor der geklammerten Folge** seiner Operanden $f\ (t_1, t_2, ..., t_n)$	 $\wedge\ (<(0, a), \neg\ (<(a, 10)))$
Präfixform:	Operator **vor** seinen Operanden $f\ t_1\ t_2 \ldots t_n$	 $\wedge\ <\ 0\ a\ \neg <\ a\ 10$
Postfixform:	Operator **nach** seinen Operanden $t_1\ t_2 \ldots t_n\ f$	 $0\ a\ <\ a\ 10\ <\ \neg\ \wedge$
Infixform	2-stelliger Operator **zwischen** seinen (beiden) Operanden $t_1\ f\ t_2$	 $0 < a\ \wedge\ \neg\ a < 10$

Die **Reihenfolge der Operanden** ist in allen vier Notationen **gleich**.

Vorlesung Modellierung WS 2011/12 / Folie 304

Ziele:

Verschiedene Notationen für denselben Term

in der Vorlesung:

An weiteren Beispielen erläutern:

- Struktur der Notationen
- Beispiele für 2-, 1- und 3-stellige Operationen
- Umformungen

Verständnisfragen:

- Kennzeichnen Sie alle Teilterme eines Terms in den 4 Formen!
- Wie finden Sie in der Postfixform die Operanden zu einem Operator?
- Können in einem Term in Infixform die Operanden immer eindeutig zugeordnet werden?

Präzedenzen und Klammern für Infixform

Die **Infixform** benötigt **Klammern** oder **Präzedenzen**, um Operanden an ihren Operator zu binden: Ist in x **+** 3 ***** y die 3 rechter Operand des **+** oder linker Operand des ***** ?

Klammern beeinflussen die Struktur von Termen in der Infixform:
z. B. (x **+** 3) ***** y oder x **+** (3 ***** y)
Redundante Klammern sind zulässig.
Ein Term ist **vollständig geklammert**, wenn er und jeder seiner Unterterme geklammert ist:
z. B. ((x) **+** ((3) ***** (y)))

Für die **Infixform** können den Operatoren unterschiedliche **Bindungsstärken (Präzedenzen)** zugeordnet werden, z. B. bindet ***** seine Operanden vereinbarungsgemäß stärker an sich als **+**, d. h. ***** hat **höhere Präzedenz** als **+**.
Damit sind x **+** 3 ***** y und x **+** (3 ***** y) verschiedene Schreibweisen für denselben Term.

Für **aufeinanderfolgende Operatoren gleicher Präzedenz** muss geregelt werden, ob sie ihre Operanden **links-assoziativ** oder **rechts-assoziativ** binden:
links-assoziativ: x **+** 3 **+** y steht für (x **+** 3) **+** y
rechts-assoziativ: x ****** 3 ****** y steht für x ****** (3 ****** y)

Funktionsform, Präfixform, Postfixform benötigen weder Regeln für Präzedenz oder Assoziativität noch zusätzliche Klammern!

Vorlesung Modellierung WS 2011/12 / Folie 305

Ziele:

Präzedenzen verstehen

in der Vorlesung:

An weiteren Beispielen erläutern:

- notwendige, redundante und vollständige Klammerung von Termen,
- Verwechselung mit Klammern 1-elementiger Folgen vermeiden,
- Präzedenzen und Assoziativität,
- Präzedenzen in Programmiersprachen

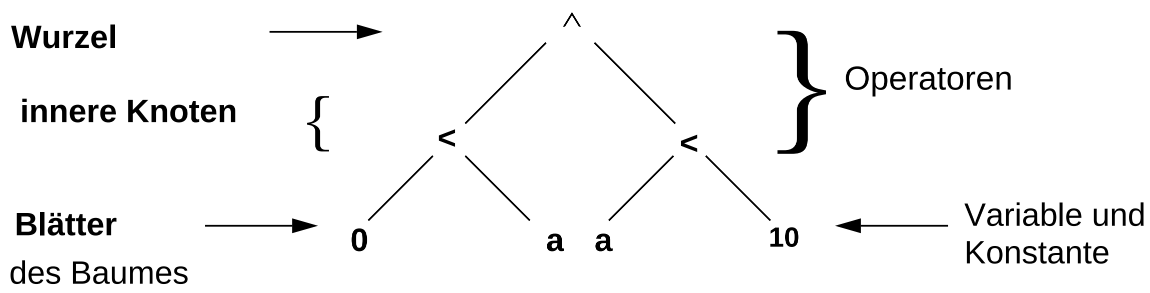
Verständnisfragen:

- Weshalb benötigen Präfix- und Postfixform keine Klammern?
- Welche Präzedenzen haben die Operatoren in Java?

Terme als Bäume

Terme kann man als Bäume darstellen (**Kantorowitsch-Bäume**):

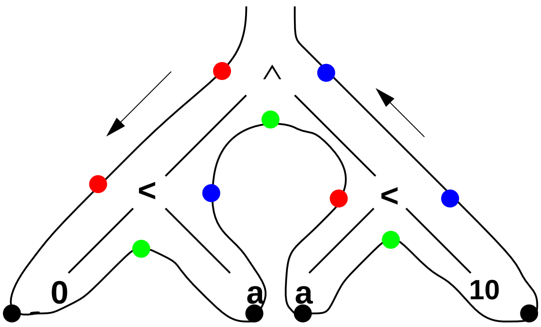
Term: $\mathbf{0 < a \wedge a < 10}$



Aus einem Durchlauf des Baumes in Pfeilrichtung erzeugt man

- **Präfixform**, wenn man beim **ersten Besuch**
- **Postfixform**, wenn man beim **letzten Besuch**
- **Infixform**, wenn man beim **vorletzten Besuch** (bei **2-stelligen** Operatoren)

den Operator aufschreibt.



Vorlesung Modellierung WS 2011/12 / Folie 306

Ziele:

Zusammenhang der Darstellungen verstehen

in der Vorlesung:

- Bäume erläutern
- Baumdurchläufe erläutern
- Rekursive Definition der Notationen

Verständnisfragen:

- Wie hängen Baumdarstellung und vollständig geklammerte Terme zusammen?

Substitution und Unifikation

Eine **Substitution** beschreibt, wie in einem Term vorkommende **Variablen durch Terme ersetzt** werden.

Eine **einfache Substitution** σ **= [v / t]** ist ein Paar aus einer Variablen v und einem Term t zur Signatur Σ. v und t müssen **dieselbe Sorte** s haben.
Beispiel: σ = [x / 2*b]

Die **Anwendung einer Substitution** σ **auf einen Term** u schreibt man **u** σ, z. B. (x+1) [x / 2*b].

Die **Anwendung einer einfachen Substitution u** σ mit σ **= [v / t]**, ist **definiert** durch

- **u [v / t] = t** , falls u die zu ersetzende Variable v ist,
- **u [v / t] = u** , falls u $\neq$ v und u eine Konstante oder eine andere Variable ist,
- **u [v / t] = f (u_1 [v / t], u_2 [v / t], ..., u_n [v / t])** , falls u = f (u_1, u_2, ..., u_n)

D. h. in u werden **alle Vorkommen der Variablen v gleichzeitig durch den Term t ersetzt**.

Kommt v auch in t vor, so wird es nicht nochmals ersetzt!

Beispiele: (x + 1) [x / 2*b] = (2*b + 1)

(x - x) [x / 3] = (3 - 3)

(x + y) [y / y*y] = (x + y*y)

Vorlesung Modellierung WS 2011/12 / Folie 307

Ziele:

Formale Definition des Einsetzens für Variable

in der Vorlesung:

An Beispielen erlätern:

- konsistentes Ersetzten mehrerer Vorkommen
- gleichzeitiges Ersetzen
- Ersetzen wird nicht iteriert
- Variable können ungebunden bleiben

Hinweis: Wir haben hier **nicht die Notation aus dem Skript vom WS 2000/2001 und nicht die aus dem Buch von Goos verwendet! Dort werden die Paare in umgekehrter Reihenfolge angegeben: [Term/Variable].**

Verständnisfragen:

Geben Sie Beispiele für Substitutionen zu Termen der Signatur zu BOOL an.

Mehrfache Substitution

In einer **mehrfachen Substitution** $\sigma = [\, v_1 / t_1, ..., v_n / t_n]$ müssen alle Variablen v_i paarweise verschieden sein. In jedem v_i / t_i müssen v_i und t_i jeweils derselben Sorte s_i angehören.
σ wird dann auf einen Term u wie folgt angewandt:

- **u σ = t_i**, falls u = v_i für ein i ∈ {1, ..., n},
- **u σ = u** , falls u eine Konstante ist oder eine Variable, die nicht unter v_i für ein i ∈ {1, ..., n} vorkommt,
- **u σ = f (u_1 σ, u_2 σ, ..., u_n σ)** , falls u = f (u_1, u_2, ..., u_n)

D. h. σ ist die gleichzeitige Substitution aller Vorkommen jeder Variablen v_i jeweils durch den Term t_i.

Beispiele: $\sigma = [\, x / 2*b,\ y / 3\,]$

$(x + y)\ \sigma = (2*b + 3)$
$(y + a*y)\ \sigma = (3 + a*3)$
$(x * y)\ [\, x / y,\ y / y*y\,] = (y * (y * y))$

Die **leere Substitution** wird **[ ]** notiert. Für alle Terme t gilt **t [ ] = t**.
Außerdem gilt **[v / v] = [ ]** für jede Variable v.

Vorlesung Modellierung WS 2011/12 / Folie 308

Ziele:

Mehrfache Substitutionen verstehen

in der Vorlesung:

An Beispielen erlätern:

- konsistentes Ersetzten mehrerer Variablen,

Hintereinanderausführung von Substitutionen

Auf einen Term können **mehrere Substitutionen hintereinander** ausgeführt werden,

z. B. $u\ \sigma_1\ \sigma_2\ \sigma_3 = ((u\ \sigma_1)\ \sigma_2)\ \sigma_3$

$(x+y)\ [x/y*x]\ [y/3]\ [x/a] = (y*x+y)\ [y/3]\ [x/a] = (3*x+3)\ [x/a] = (3*a+3)$

Mehrere **Substitutionenen hintereinander** können als **eine Substitution** angesehen werden:

z. B. $u\ \sigma_1\ \sigma_2\ \sigma_3 = u\ (\sigma_1\ \sigma_2\ \sigma_3) = u\ \sigma$

Mehrere **einfache Substitutionen hintereinander** kann man **in eine mehrfache Substitution** mit gleicher Wirkung umrechnen:

Die Hintereinanderausführung $[\ x_1 / t_1, ..., x_n / t_n\ ]\ [\ y / r\ ]$

hat auf jeden Term die gleiche Wirkung wie

falls y unter den x_i vorkommt $[\ x_1 / (t_1[\ y / r\ ]), ..., x_n / (t_n\ [\ y / r\ ])]$
falls y nicht unter den x_i vorkommt $[\ x_1 / (t_1[\ y / r\ ]), ..., x_n / (t_n\ [\ y / r\ ]), y / r\ ]$

Beispiel: $[\ x / y*x\ ]\ [\ y / 3\ ]\ [\ x / a\ ] = [\ x / 3*x, y / 3]\ [\ x / a\ ] = [\ x / 3*a, y / 3\ ]$

Vorlesung Modellierung WS 2011/12 / Folie 308a

Ziele:

Umgang mit Substitutionen verstehen

in der Vorlesung:

An Beispielen erlätern:

- Hintereinanderausführung,
- Umrechnung.

Verständnisfragen:

Begründen Sie die Regel für die Umrechnung.

Umfassende Terme

Rechenregeln werden mit **allgemeineren Termen** formuliert, die auf **speziellere Terme** angewandt werden,

z. B. Distributivgesetz:	$a * (b + c)$	=	$a * b + a * c$
angewandt auf	$2 * (3 + 4*x)$	=	$2 * 3 + 2 * 4*x$

Ein **Term s umfasst einen Term t**, wenn es eine Substitution σ gibt, die s in t umformt: $s\ \sigma = t$

s umfasst t, ist eine **Quasiordnung**, d. h. die Relation **umfasst** ist

transitiv: sei $r\ \sigma_1 = s$, $s\ \sigma_2 = t$, dann ist $r\ (\sigma_1\ \sigma_2) = t$

reflexiv: $t\ [\,] = t$, mit der leeren Substitution [ ]

Eine **Halbordnung ist umfasst nicht**, weil

nicht antisymmetrisch: Terme, die sich nur in den Variablennamen unterscheiden, kann man ineinander umformen, z. B.
$2*x\ [\ x / y\ ] = 2*y$ und $2*y\ [\ y / x\ ] = 2*x$

Deshalb gilt zwar der allgemeinere Term $a * (b + c)$ umfasst den spezielleren $2 * (3 + 4*x)$, aber nicht immer ist ein Term s allgemeiner als ein Term t, wenn s umfasst t: $2*x$ und $2*y$

Vorlesung Modellierung WS 2011/12 / Folie 309

Ziele:

Terme als Muster verstehen

in der Vorlesung:

- Substitution als Anwendung einer Rechenregel
- Erläuterung der Relation "umfasst"
- weitere Beispiele

Verständnisfragen:

Warum ist es nicht völlig korrekt, zu sagen, dass t spezieller ist als s, wenn s t umfasst?

Unifikation

Die **Unifikation substituiert zwei Terme, sodass sie gleich werden**.

Zwei Terme s und t sind unifizierbar, wenn es eine Substitution σ **gibt mit s** $\sigma =$ **t** σ.
σ heißt **Unifikator von s und t**.

Beispiel: Terme: s = (x + y) t = (2 + z)
Unifikatoren: σ_1 = [x / 2, y / z] σ_2 = [x / 2, z / y],
σ_3 = [x / 2, y / 1, z / 1] σ_4 = [x / 2, y / 2, z / 2] ...

Ist σ **ein Unifikator** von s und t und τ **eine Substitution**, dann ist auch die Hintereinanderausführung $\sigma\ \tau = \sigma'$ auch ein Unifikator von s und t.

Ein **Unifikator** σ **heißt allgemeinster Unifikator** der Terme s und t, wenn es zu allen anderen Unifikatoren σ' eine Substitution τ gibt mit $\sigma\ \tau = \sigma'$.

Im Beispiel sind σ_1 und σ_2 allgemeinste Unifikatoren, z. B. σ_1 [z / 1] $= \sigma_3$

Es kann **mehrere allgemeinste Unifikatoren** geben. Sie können durch **Umbenennen von Variablen** ineinander überführt werden, z. B.
σ_1 [**z / y**] = [x / 2, y / z] [z / y]= [x / 2, y / y, z / y] = [x / 2, z / y] $= \sigma_2$

Vorlesung Modellierung WS 2011/12 / Folie 310

Ziele:

Allgemeines Prinzip Unifikation verstehen

in der Vorlesung:

An Beispielen zeigen:

- Unifikatoren
- nicht unifizierbare Terme
- allgemeinste Unifikatoren machen keine unnötigen Festlegungen.
- Zur letzten Zeile der Folie: Eine Substitution [y / y] hat keine Wirkung, also [y / y] = []. In mehrfachen Substitutionen kann man Komponenten der Form y / y weglassen und Komponenten vertauschen, ohne die Wirkung der Substitution zu ändern.

Verständnisfragen:

- Wie müssen 2 Terme beschaffen sein, damit es Unifikatoren gibt, die verschieden sind von den allgemeinsten Unifikatoren? Hinweis: Nur wenn ein allgemeinster Unifikator noch Variablen offen lässt, kann es speziellere geben.

Unifikationsverfahren

Unifikation zweier Terme s und t nach Robinson:

Seien s und t Terme in **Funktionsschreibweise**.
Dann ist das **Abweichungspaar A(s, t) = (u, v)** das erste Paar unterschiedlicher, korrespondierender Unterterme u und v, das man beim Lesen von links nach rechts antrifft.

Algorithmus:

1. Setze σ = [ ] (leere Substitution)
2. Solange es ein Abweichungspaar A(s σ, t σ) = (u, v) gibt wiederhole:
   a. ist **u eine Variable x**, die in v nicht vorkommt, dann ersetze σ durch σ [x / v], oder
   b. ist **v eine Variable x**, die in u nicht vorkommt, dann ersetze σ durch σ [x / u],
   c. **sonst** sind die Terme s und t **nicht unifizierbar; Abbruch** des Algorithmus.
3. Bei Erfolg gilt s σ = t σ und σ **ist allgemeinster Unifikator**.

Beachte, dass bei jeder Iteration die bisherige Substitution auf die vollständigen Terme s, t angewandt wird.

Vorlesung Modellierung WS 2011/12 / Folie 311

Ziele:

Terme systematisch unifizieren

in der Vorlesung:

- Verfahren an Beispielen zeigen.
- Begründen, weshalb die Substitution immer wieder auf s und t angewandt wird.
- Begründen, weshalb in 2a und 2b geprüft wird, ob die Variable x in dem Unterterm vorkommt.

Verständnisfragen:

- Zeigen sie an einem Beispiel, dass es nötig ist, in 2a und 2b zu prüfen ob die Variable x in dem Unterterm vorkommt.

Beispiel für Unifikationsverfahren

Unifikation zweier Terme s und t nach Robinson:

s = + (***** (2, x), 3)
t = + (z, x) σ = [ ]

Schritt ↓ Abweichungspaar

1 s σ = + (***** (2, x), 3) Fall 2b: σ = [ ] [z / ***** (2, x)]
t σ = + (z, x)

↓ Abweichungspaar

2 s σ = + (***** (2, x), 3) Fall 2b: σ = [ ] [z / ***** (2, x)] [x / 3]
t σ = + (***** (2, x), x)

3 s σ = + (***** (2, 3), 3) allgemeinster Unifikator: σ = [z / ***** (2, x)] [x / 3] =
t σ = + (***** (2, 3), 3) [z / ***** (2, 3), x / 3]

Vorlesung Modellierung WS 2011/12 / Folie 312

Ziele:

Verfahren von Robinson anwenden

in der Vorlesung:

Verfahren an Beispielen zeigen.

3.2 Algebren

Eine **Algebra** ist eine **formale Struktur,** definiert durch
eine **Trägermenge**, **Operationen** darauf und **Gesetze** zu den Operationen.

In der Modellierung der Informatik spezifiziert man mit Algebren
Eigenschaften veränderlicher Datenstrukturen und dynamische Systeme,
z. B. Datenstruktur *Keller* oder die Bedienung eines Getränkeautomaten.

Wir unterscheiden 2 Ebenen: **abstrakte Algebra** und **konkrete Algebra**:

Eine **abstrakte Algebra** spezifiziert Eigenschaften **abstrakter Operationen**,
definiert nur duch eine **Signatur** - Realisierung durch Funktionen bleibt absichtlich offen

Trägermenge: korrekte Terme zu der Signatur

Gesetze erlauben, Vorkommen von Termen durch andere Terme zu ersetzen
z. B. $\neg$ false –> true pop (push (k, t)) –> k

Eine **konkrete Algebra** zu einer abstrakten Algebra

definiert **konkrete Funktionen** zu den Operationen der Signatur,
so dass die Gesetze in **Gleichungen zwischen den Funktionstermen** übergehen.

Sie beschreibt so eine **Implementierung** der spezifizierten Datenstruktur,
bzw. des Systems

Vorlesung Modellierung WS 2011/12 / Folie 313

Ziele:

Vorschau zur Modellierung mit Algebren

in der Vorlesung:

Erläuterungen dazu

Abstrakte Algebra

Eine **abstrakte Algebra A = (τ, Σ, Q)** ist definiert durch die
Menge korrekter Terme τ zur **Signatur** Σ und eine **Menge von Axiomen (Gesetzen) Q**.

Axiome haben die Form t_1 -> t_2, wobei t_1, t_2, **korrekte Terme gleicher Sorte** sind, die **Variablen** enthalten können. Die Algebra definiert, wie man Terme **mit den Axiomen in andere Terme umformen** kann.

Mit Axiomen umformen heißt: Unter Anwenden eines Axioms t_1 -> t_2 kann man einen Term s_1 in einen Term s_2 umformen. Wir schreiben s_1 -> s_2, wenn gilt:

- s_1 und s_2 stimmen in ihren „äußeren“ Strukturen überein und unterscheiden sich nur durch die Unterterme r_1 und r_2 an entsprechenden Positionen in s_1 und s_2, und
- es gibt eine Substitution σ , sodass gilt $t_1\,\sigma = r_1$ und $t_2\,\sigma = r_2$

Terme	s_1 =	.... r_1	->	.... r_2	=	s_2
		=		=		
		$t_1\,\sigma$		$t_2\,\sigma$		
Axiom		t_1	->	t_2		

s ist in t umformbar, wenn es eine endliche Folge von Termen s = s_0, s_1, ..., s_n = t mit s_{i-1} -> s_i gibt; wir schreiben dann s -> t.

„->“ ist **transitiv**. Wenn es auch **irreflexiv** ist (so sollten die Axiome gewählt werden), ist es eine **strenge Halbordnung**.

Vorlesung Modellierung WS 2011/12 / Folie 314

Ziele:

Axiome definieren Terme als gleichbedeutend

in der Vorlesung:

- Anwendungen von Axiomen auf Termpaare zeigen (Substitution)
- Beispiel: Kommutativgesetz anwenden

Verständnisfragen:

Erklären Sie Anwendungen des Kommutativgesetzes präzise in der definierten Terminologie.

Beispiel: abstrakte Algebra Bool

Signatur Σ = ({BOOL}, F)

Operationen F:

true:		-> BOOL
false:		-> BOOL
$\wedge$:	BOOL x BOOL	-> BOOL
$\vee$:	BOOL x BOOL	-> BOOL
$\neg$:	BOOL	-> BOOL

Axiome Q: für alle x,y der Sorte BOOL gilt

Q_1:	$\neg$ true	–> false
Q_2:	$\neg$ false	–> true
Q_3:	true $\wedge$ x	–> x
Q_4:	false $\wedge$ x	–> false
Q_5:	x $\vee$ y	–> $\neg$ ($\neg$ x $\wedge$ $\neg$ y)

Die Axiome sind geeignet, alle korrekten Terme ohne Variablen in in einen der beiden Terme **true** oder **false** umzuformen.

true und **false** heißen **Normalformen** (siehe Folie 3.20).

Vorlesung Modellierung WS 2011/12 / Folie 315

Ziele:

Beispiel für eine abstrakte Algebra

in der Vorlesung:

- Algebra Bool erläutern
- Gesetze anwenden
- Weitere Gesetze formulieren

Konkrete Algebra

Zu einer abstrakten Algebra A_a = (τ, (S, F), Q), kann man

konkrete Algebren wie $\mathbf{A_k}$ **= ($\mathbf{W_k}$, $\mathbf{F_k}$, Q)**

angeben, wobei

W_k eine **Menge von Wertebereichen** ist, je einer **für jede Sorte** aus S,

F_k eine **Menge von Funktionen** ist, je eine **für jede Operation** aus F.

Die Definitions- und Bildbereiche der Funktionen müssen konsistent den Sorten der Operationen zugeordnet werden.

Den **Axiomen Q** müssen **Gleichungen zwischen den Funktionstermen** in den Wertebereichen entsprechen.

Es können in der konkreten Algebra noch weitere Gleichungen gelten.

Eine konkrete Algebra heißt auch **Modell der abstrakten Algebra**.

Vorlesung Modellierung WS 2011/12 / Folie 316

Ziele:

Zusammenhang zwischen konkreter und abstrakter Algebra verstehen

in der Vorlesung:

Am Beispiel Mod-225a erläutern

Beispiel für eine konkrete Algebra

Beispiel: eine konkrete Algebra FSet zur abstrakten Algebra Bool:

	konkrete Algebra FSet	**abstrakte Algebra Bool**
W_k:	$\{\emptyset, \{1\}\}$	Sorte BOOL
F_k:	$\{1\}$	true
	$\emptyset$	false
	Mengendurchschnitt $\cap$	$\wedge$
	Mengenvereinigung $\cup$	$\vee$
	Mengenkomplement bezüglich $\{1\}$	$\neg$

Axiome Q:
Man kann zeigen, dass die Axiome Gleichungen zwischen den Termen in W_k entsprechen:
z. B. $\emptyset \cap x = \emptyset$ entspricht false $\wedge$ x -> false

Die boolesche Algebra mit den üblichen logischen Funktionen ist natürlich auch eine konkrete Algebra zur abstrakten Algebra Bool.

Vorlesung Modellierung WS 2011/12 / Folie 317

Ziele:

Beispiel für Zusammenhang zwischen konkreter und abstrakter Algebra

in der Vorlesung:

Beispiel mit Folie Mod-3.16 erläutern

- Funktionstafeln der konkreten Funktionen angeben
- Gültigkeit der Gleichungen zu den Axiomen zeigen
- Ebenso für die konkrete boolesche Algebra

Übungsaufgaben:

Tauschen Sie in FSet die Funktionen zu T und F. Gelten die Gleichungen zu den Axiomen noch?

Verständnisfragen:

- Zeigen Sie, dass die Gleichungen zu allen Axiome Q von Bool in FSet gelten.

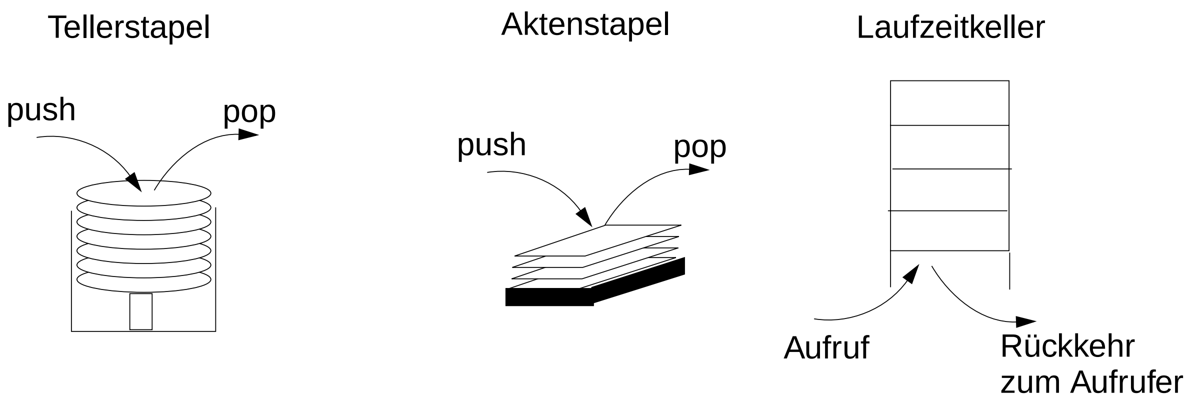
Beispiel 2.2: Datenstruktur Keller

Die Eigenschaften einer **Datenstruktur Keller** beschreiben wir zunächst informell.
Folgende **Operationen** kann man mit einem Keller ausführen:

create Stack:	liefert einen leeren Keller
push:	fügt ein Element in den Keller ein
pop:	entfernt das zuletzt eingefügte Element
top:	liefert das zuletzt eingefügte und nicht wieder entfernte Element
empty:	gibt an, ob der Keller leer ist.

Die Eigenschaften der Datenstruktur Keller sollen präzise durch eine abstrakte Algebra spezifiziert werden.

Beispiele



Vorlesung Modellierung WS 2011/12 / Folie 318

Ziele:

Das Kellerprinzip informell verstehen

in der Vorlesung:

- Keller-Prinzip: Last-in-first-out (LIFO)
- Beispiele erläutern
- Anwendung eines Kellers: Infix in Postfix umwandeln

Verständnisfragen:

- Erklären Sie das Kellerprinzip an der Abarbeitung von Funktions- bzw. Methodenaufrufen in Programmen.

Beispiel: Abstrakte Algebra spezifiziert Keller

Abstrakte Algebra **Keller**:

Signatur Σ = (S, F),
Sorten S = {Keller, Element, BOOL},
Operationen F:

createStack:		-> Keller
push:	Keller x Element	-> Keller
pop:	Keller	-> Keller
top:	Keller	-> Element
empty:	Keller	-> BOOL

Axiome Q: für beliebige Terme t der Sorte Element und k der Sorte Keller gilt:

K1:	empty (createStack)	-> true
K2:	empty (push (k, t))	-> false
K3:	pop (push (k, t))	-> k
K4:	top (push (k, t))	-> t

Keller ist die Sorte, deren Terme Kellerinhalte modellieren.
Element und BOOL sind **Hilfssorten** der Algebra.

Implementierungen der abstrakten Algebra Keller können durch **konkrete Algebren** dazu beschrieben werden.

Vorlesung Modellierung WS 2011/12 / Folie 319

Ziele:

Abstrakte Algebra als Spezifikation verstehen

in der Vorlesung:

- Terme umformen
- K3 mit LIFO begründen
- Anschauliche Darstellungen und Implementierungen von Kellerinhalten entsprechen konkreten Algebren

Verständnisfragen:

- Geben sie verschiedene Terme an, die zu top (push (createStack, 1)) und zu push (push (createStack, 1), 2) gleichbedeutend sind.

Klassifikation von Operationen

Die Operationen einer Algebra werden in 3 disjunkte Mengen eingeteilt:

Konstruktoren: Ergebnissorte ist die definierte Sorte

Hilfskonstruktoren: Ergebnissorte ist die definierte Sorte und
sie können durch Axiome aus Termen entfernt werden

Projektionen: andere Ergebnissorte

z. B. in der Keller-Algebra: definierte Sorte ist Keller

createStack:		-> Keller	**Konstruktor**
push:	Keller x Element	-> Keller	**Konstruktor**
pop:	Keller	-> Keller	**Hilfskonstruktor** (K3 entfernt ihn)
top:	Keller	-> Element	**Projektion**
empty:	Keller	-> BOOL	**Projektion**

Vorlesung Modellierung WS 2011/12 / Folie 320

Ziele:

Einteilung der Operationen

in der Vorlesung:

- Klassifikation erläutern
- Konstruktoren zur Algebra Bool angeben.

Verständnisfragen:

- Klassifizieren Sie die Operationen der Algebra Bool.

Normalform

Terme ohne Variable der definierten Sorte sind in **Normalform**, wenn sie nur **Konstruktoren** enthalten **kein Axiom anwendbar** ist.

Normalform-Terme der Algebra Bool sind: true false

Normalform-Terme der Keller-Algebra haben die Form:
push (.... push (createStack, n_1) , ...), n_m), mit $m \geq 0$

Die **Terme in Normalform** sind die minimalen Elemente bzgl. der strengen Halbordnung ->.

Terme s, t, die in **dieselbe Normalform** umformbar sind, heißen **gleichbedeutend**, $s \equiv t$.

Undefinierte Terme:

Terme der definierten Sorte, die man **nicht in eine Normalform** umformen kann, werden als **undefiniert** angesehen. Sie modellieren eine **Fehlersituation**, z. B. pop (createStack)

Für manche **Projektionen** gibt es nicht zu jedem Term in Normalform ein anwendbares Axiom; dies modelliert auch **Fehlersituationen**, z. B. top (createStack)

Vorlesung Modellierung WS 2011/12 / Folie 320a

Ziele:

Reduktion von Termen auf ihre Normalform

in der Vorlesung:

- Reduzierbarkeit auf Normalform durch strukturelle Induktion zeigen
- Undefinierte Terme erläutern

Verständnisfragen:

- Gibt es undefinierte Terme in der Algebra Bool?

Anwendungen algebraischer Spezifikationen: Eigenschaften aus den Axiomen erkennen

Beispiel: Keller

1. K3: pop (push (k, t)) -> k

Keller-Prinzip: zuletzt eingefügtes Element wird als erstes wieder entfernt
(last-in-first-out, LIFO)

2. top: Keller -> Element
   K4: top (push (k, t)) -> t

top ist die einzige Operation, die Keller-Elemente liefert:
Nur auf das zuletzt eingefügte, nicht wieder entfernte Element kann **zugegriffen** werden.

3. push (.... push (createStack, n_1) , ...), n_m), mit $m \geq 0$
   K3: pop (push (k, t)) -> k

Zählt man in einem Term von innen nach außen die push-Operationen positiv und die pop-Operationen negativ, und ist der Wert immer nicht-negativ, so ergibt sich die **Anzahl der Elemente im Keller**, andernfalls ist der Term undefiniert.
Begründung: Rückführung auf Normalform, eine push-Operation für jedes Element im Keller.

Vorlesung Modellierung WS 2011/12 / Folie 321

Ziele:

Axiome spezifizieren Eigenschaften

in der Vorlesung:

- Die drei Beispiele erläutern
- Über Keller nachdenken, ohne sie zu implementieren

Spezifikation um Operationen erweitern

Erweitere die Keller-Spezifikation um eine **Operation size**.
Sie soll die **Anzahl der Elemente im Keller** liefern.

1. Operation **size in die Signatur einfügen:**
   size: Keller -> NAT
2. Ergebnis-Sorte **NAT zu den Sorten zufügen**:
   S = {Keller, Element, BOOL, NAT}
3. **Axiome zufügen**, so dass size für jeden Keller-Wert definiert ist:
   K7: size (createStack) -> null
   K8: size (push (k, t)) -> succ (size (k))
4. Weil in der **Normalform** nur createStack und push vorkommen, braucht size nur für solche Terme definiert zu werden.

Dabei wird vorausgesetzt, dass folgende Algebra bekannt ist:
Sorten: S = {NAT}
Operationen: null: -> NAT, succ: NAT -> NAT
(succ (n) modelliert den Nachfolger von n, also n + 1.)

Vorlesung Modellierung WS 2011/12 / Folie 322

Ziele:

Operationen und Axiome entwerfen

in der Vorlesung:

Schritte zur Erweiterung der Spezifikation zeigen

Verständnisfragen:

Erweitern Sie die Spezifikation um eine Operation, die 2 Elemente einfügt.

Realisierung der Spezifikation durch eine konkrete Algebra

Beispiel: eine Realisierung von Kellern durch **Funktionen auf Folgen** von natürlichen Zahlen:

Zuordnung der Sorten:

konkret	**abstrakt**
Bool	BOOL
$\mathbb{N}_0$	Element
N-Folge = $\mathbb{N}^*$	Keller

Signatur und **Zuordnung von Funktionen**

konkret			**abstrakt**
newFolge:		-> N-Folge	createStack
append:	N-Folge x $\mathbb{N}_0$	-> N-Folge	push
remove:	N-Folge	-> N-Folge	pop
last:	N-Folge	-> $\mathbb{N}$	top
noElem:	N-Folge	-> Bool	empty

Definition der Funktionen

newFolge( )	-> ()
append $((a_1, ..., a_n), x)$	-> $(a_1, ..., a_n, x)$
remove $((a_1, ..., a_{n-1}, a_n))$	-> $(a_1, ..., a_{n-1})$
last $((a_1, ..., a_n))$	-> a_n
noElem (f)	-> f = ()

Gültigkeit der Axiome zeigen

Vorlesung Modellierung WS 2011/12 / Folie 324

Ziele:

Beispiel für eine Realisierung

in der Vorlesung:

Funktionen erläutern

- Zuordnung erläutern
- Gültigkeit der Axiome zeigen

Übungsaufgaben:

Mit der Klasse Vector aus java.lang kann man Keller implementieren. Schlagen sie in der Dokumentation nach und begründen Sie, dass sich die Methoden addElement, removeElemenAt, lastElement, und size dafür eignen.

Keller in Algorithmen einsetzen

Aufgabe: Terme aus **Infixform in Postfixform** umwandeln

gegeben: Term t in Infixform, mit 2-stelligen Operatoren unterschiedlicher Präzedenz; (zunächst) ohne Klammern

gesucht: Term t in Postfixform

Eigenschaften der Aufgabe und der Lösung:

1. **Reihenfolge der Variablen und Konstanten bleibt unverändert**
2. **Variablen und Konstanten werden vor ihrem Operator ausgegeben**, also sofort
3. In der Infixform aufeinander folgende **Operatoren echt steigender Präzedenz** stehen in der Postfixform **in umgekehrter Reihenfolge**; also kellern.
4. **Operatorkeller enthält Operatoren echt steigender Präzedenz**.
   Es gilt die **Kellerinvariante KI**:
   Sei push (... push (CreateStack, opr_1), opr_2), ...) dann gilt
   Präzedenz (opr_i) < Präzedenz (opr_{i+1})

Vorlesung Modellierung WS 2011/12 / Folie 325

Ziele:

Kelleranwendung verstehen

in der Vorlesung:

- Erläuterung der Eigenschaften
- Eigenschaften der Aufgabe verstehen, bevor sie gelöst wird
- Kellerinvariante: Eine Aussage, die für die Benutzung des Kellers immer gelten muss.

Verständnisfragen:

Wie werden bei diesen Regeln aufeinander folgende Operatoren gleicher Präzedenz behandelt?

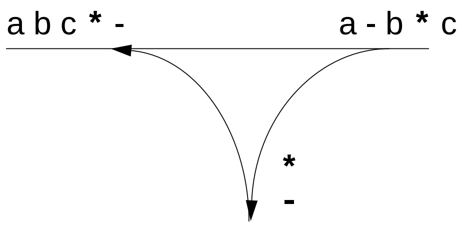
Algorithmus: Infix- in Postfixform wandeln

Die Eingabe enthält einen Term in Infixform;
die Ausgabe soll den Term in Postfixform enthalten

```
Variable: keller ∈ Keller; symbol ∈ Operator ∪ ElementarOperand

keller = createStack();
solange Eingabe nicht leer wiederhole                {KI}
        lies symbol
        falls symbol ∈ ElementarOperand
              gib symbol aus
        falls symbol ∈ Operator                      {KI}
              solange  not empty (keller) ∧
                       Präzedenz (top (keller)) ≥ Präzedenz (symbol)
              wiederhole                             {KI}
                       gib top (keller) aus;
                       keller = pop (keller);
              keller = push(keller, symbol);         {KI}

solange not empty (keller) wiederhole
        gib top(keller) aus;
        keller = pop(keller);
```



An den Stellen {KI} gilt die Kellerinvariante.

Vorlesung Modellierung WS 2011/12 / Folie 326

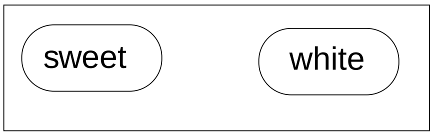
Ziele:

Benutzung der Kelleroperationen verstehen

in der Vorlesung:

- Algorithmus erläutern
- Graphik erläutern
- Gültigkeit der Kellerinvariante zeigen

Abstrakte Algebra für Teilaspekt des Getränkeautomaten



Knöpfe des Getränkeautomaten
zur Auswahl von Zutaten

Die Sorte **Choice** modelliert die Auswahl;
Add ist eine Hilfssorte

Signatur Σ = (S, F);

Sorten S := {Add, Choice}

Operationen F:

sweet:	-> Add
white:	-> Add
noChoice:	-> Choice
press: Add x Choice	-> Choice

Bedeutung der Axiome:

Q_1: Knopf nocheinmal drücken macht Auswahl rückgängig.

Q_2: Es ist egal, in welcher Reihenfolge die Knöpfe gedrückt werden.

Axiome Q: für alle a der Sorte Add und
für alle c der Sorte Choice gilt:

Q_1: press (a, press (a, c)) -> c

Q_2: press (sweet, press (white, c)) ->
press (white, press (sweet, c))

Beispiel-Terme: press (white, noChoice)
press (sweet, press (white, press (sweet, noChoice)))

Vorlesung Modellierung WS 2011/12 / Folie 326b

Ziele:

Abfolge von Bedienoperationen modellieren

in der Vorlesung:

Erläuterungen zu

- den beiden Sorten,
- den Axiomen: sie identifizieren Terme gleicher Bedeutung;
- der Bedeutung der Axiome

Übungsaufgaben:

Untersuchen und erläutern Sie

- Alternativen zu dieser Algebra;
- Algebren zur Modellierung von Knöpfen, die nur alternative betätigt werden können.

Verständnisfragen:

Begründen Sie die Bedeutung der Axiome anhand von Termen.