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

   Die **Anzahl der Elemente im Keller** ist
   die Anzahl der push-Operationen abzüglich der Anzahl der pop-Operationen im Term.
   Begründung: Rückführung auf Normalform, eine push-Operation für jedes Element im Keller.

© 2001 bei Prof. Dr. Uwe Kastens

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 -> N

2. Ergebnis-Sorte **N zu den Sorten zufügen**:
   S = {Keller, Element, BOOL, N}

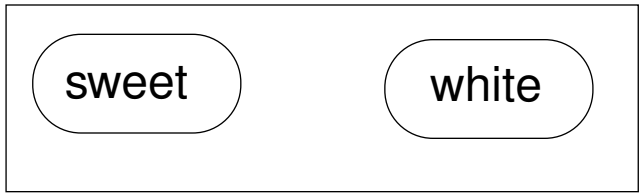
3. **Axiome zufügen**, so dass size für jeden Keller-Wert definiert ist:
   K7: size (createStack) ≡ 0
   K8: size (push (k, t)) ≡ size (k) + 1

Weil in der **Normalform** nur createStack und push vorkommen, braucht size nur für solche Terme definiert zu werden.

Es wird vorausgesetzt, dass für die Sorte N die Konstanten 0 und 1 sowie die Operation + definiert sind.

© 2000 bei Prof. Dr. Uwe Kastens

Abstrakte Algebra für Teilaspekt des Getränkeautomaten



Knöpfe des Getränkeautomaten
zur Auswahl von Zutaten

Signatur $\Sigma = (S, F)$;

Sorten S := {Add, Choice}

Operationen F:

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

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

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, b der Sorte Add und
für alle c der Sorte Choice gilt:

Q_1: press (a, press (a, c)) $\equiv$ c

Q_2: press (a, press (b, c)) $\equiv$ press (b, press (a, c)

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

© 2001 bei Prof. Dr. Uwe Kastens

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 = $\mathbb{N}_0^*$	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	-> $\mathbb{N}_0$	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

© 2001 bei Prof. Dr. Uwe Kastens

Keller in Algorithmen einsetzen

Aufgabe: Terme aus **Infix-Form in Postfix-Form** umwandeln
gegeben: Term t in Infix-Form, mit 2-stelligen Operatoren unterschiedlicher Präzedenz, zunächst ohne Klammern
gesucht: Term t in Postfix-Form

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 Infix-Form aufeinander folgende **Operatoren echt steigender Präzedenz** stehen in der Postfix-Form **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 ($opr_{\mathbf{i}}$) < Präzedenz ($opr_{\mathbf{i+1}}$)

© 2001 bei Prof. Dr. Uwe Kastens

Algorithmus: Infix- in Postfix-Form wandeln

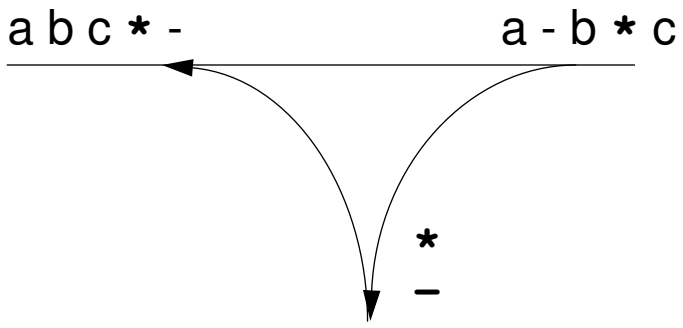
Die Eingabe enthält einen Term in Infix-Form;
die Ausgabe soll den Term in Postfix-Form enthalten

```
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.



© 2000 bei Prof. Dr. Uwe Kastens