9. Übung

Abgabe bis 09.01.2017, 10:00 Uhr

Einzelaufgabe 9.1: ADT — 27 EP

Gegeben seien die folgenden Gerüste der ADTs Nat (zur Repräsentation natürlicher Zahlen $\mathbb{N}_0^+$), $EleList$ (vereinfachte elementare Liste aus der Vorlesung) und $NatAlg$ für zusätzliche Funktionen:

---

adt *Nat*
sorts *Nat*
ops

zero:		$\mapsto$ *Nat*	*// entspricht der Zahl „0“*
succ:	*Nat*	$\mapsto$ *Nat*	*// Nachfolger „$(x+1)$“ der Nat-Zahl x*
add:	*Nat* $\times$ *Nat*	$\mapsto$ *Nat*	*// Addition im Nat-Raum*
sub:	*Nat* $\times$ *Nat*	$\mapsto$ *Nat*	*// Subtraktion im Nat-Raum (z.B. „$42_{Nat} - 666_{Nat} = 0_{Nat}$“!)*
mul:	*Nat* $\times$ *Nat*	$\mapsto$ *Nat*	*// Multiplikation im Nat-Raum*
div:	*Nat* $\times$ *Nat*	$\mapsto$ *Nat*	*// ganzzahlige (!) Division im Nat-Raum*

axs

$\forall y \in Nat : sub(zero, y) = zero$ *// wir kennen hier nur $\mathbb{N}_0^+$*
$\ldots$ *// weitere Axiome aus Platzgründen weggelassen*

end *Nat*

---

adt *EleList*
sorts *EleList, T*
ops

create:		$\mapsto$ *EleList*	*// erzeugt eine neue und zunächst leere „Elementare Liste“*
add:	*EleList* $\times$ *T*	$\mapsto$ *EleList*	*// fügt ein neues Element **vorne** in die Liste ein*
head:	*EleList*	$\mapsto$ *T*	*// gibt das **erste** (vorderste) Element der Liste zurück*
tail:	*EleList*	$\mapsto$ *EleList*	*// gibt die Restliste **ohne** das erste Element zurück*

axs

$\ldots$ ***// siehe öffentlicher Testfall!***

end *EleList*

---

adt *NatAlg*
sorts *NatAlg, Nat, EleList*
ops

mod:	*Nat* $\times$ *Nat*	$\mapsto$ *Nat*	*// Modulo-Operator im Nat-Raum*
lt:	*Nat* $\times$ *Nat*	$\mapsto$ *Nat*	*// „$<$“ im Nat-Raum: $zero \mathrel{\widehat{=}} \texttt{false} \wedge succ(zero) \mathrel{\widehat{=}} \texttt{true}$!*
eq:	*Nat* $\times$ *Nat*	$\mapsto$ *Nat*	*// „$=$“ im Nat-Raum: $zero \mathrel{\widehat{=}} \texttt{false} \wedge succ(zero) \mathrel{\widehat{=}} \texttt{true}$!*
gt:	*Nat* $\times$ *Nat*	$\mapsto$ *Nat*	*// „$>$“ im Nat-Raum: $zero \mathrel{\widehat{=}} \texttt{false} \wedge succ(zero) \mathrel{\widehat{=}} \texttt{true}$!*
gcd:	*Nat* $\times$ *Nat*	$\mapsto$ *Nat*	*// größter gemeinsamer Teiler (ggT) im Nat-Raum*
lcm:	*Nat* $\times$ *Nat*	$\mapsto$ *Nat*	*// kleinstes gemeinsames Vielfaches (kgV) im Nat-Raum*
pfz:	*Nat*	$\mapsto$ *EleList*	*// Primfaktorzerlegung im Nat-Raum (aufsteigend sortiert!)*

axs

$\ldots$ *// ToDo ;)*

end *NatAlg*

WICHTIG: In dieser Aufgabe stehen **keine** anderen Datentypen (also **kein** `int`, `String` usw.), Konstanten (auch **nicht** 0, 1 usw.) oder Operationen (also **kein** $<$, $+$, $==$, $\neq$ usw.) zur Verfügung: Sie dürfen ausschließlich die vorgegebenen oder zu ergänzenden **adt**s mit deren **ops** verwenden! Einzige erlaubte *Ausnahme*: In den Bedingungen auf der „rechten Seite" eines Axioms dürfen Sie die Identität mittels $=$ bzw. $\neq$ prüfen.

a) Implementieren Sie den ADT $EleList$ in einer gleichnamigen Klasse.
Ihre `EleList` darf *keinen* expliziten Konstruktor deklarieren. Abgesehen von den zwingend zu implementierenden Methoden (**ops**) und den notwendigen beiden Attributen darf `EleList` ***keine*** weiteren Attribute, Methoden oder innere Klassen haben ($\leadsto$ siehe öffentlicher Test!). Ihre Klasse soll eine „*wirkungsfreie*" Implementierung des ADT $EleList$ mit ausschließlich *unveränderlichen Objekten* darstellen.

b) Ergänzen Sie den ADT $NatAlg$ um die Axiome der Operationen mod, gt, gcd und lcm so, dass diese dem Kommentar entsprechen. Da alle **ops**-Namen für diese Teilaufgabe disjunkt sind, dürfen Sie den zugehörigen ADT-Namen als Präfix weglassen. Beachten Sie unbedingt auch die Einschränkung zu Beginn der Aufgabe!

c) Implementieren Sie den ADT $NatAlg$ in einer gleichnamigen Klasse – diesmal aber *alle* Operationen **ops**. Die vorangehenden Vorgaben zur Klasse `EleList` gelten auch für `NatAlg` entsprechend! (*Tipp:* Der öffentliche Test erzwingt eine „private Hilfsoperation" $pfzH\ \ldots$)

WICHTIG: Erfüllen Sie ***unbedingt alle*** zugehörigen öffentlichen Tests – insbesondere diejenigen, die die inhaltliche Struktur Ihrer Klassen prüfen, andernfalls bekommen Sie dafür *keine* Punkte! Beachten Sie weiterhin, dass Sie ***keinerlei Java-API-Klassen*** verwenden dürfen!
Geben Sie Ihre *vollständige* Lösung über EST ab.

Einzelaufgabe 9.2: Vollständige Induktion — 8 EP

Die Methode `parS(n)` soll Partialsummen einer Collatz-ähnlichen Folge für $n \geq 1$ berechnen:

```java
long parS(long n) {
    if (n == 1) {
        return 4;
    } else if (n % 2 == 0) {
        return parS(n - 1) + n;
    } else {
        return parS(n - 1) + 3 * n + 1;
    }
}
```

Sie dürfen vereinfachend annehmen, dass es keinen Überlauf geben kann.
Anstelle der mathematischen Modulo-Funktion ($x\ mod\ y$) dürfen Sie in Ihrem Beweis vereinfachend die Java-Schreibweise `x % y` benutzen.

Wichtig: *Geben Sie Ihre Beweise möglichst ausführlich an, d.h. einzelne Zwischenschritte und Umformungen müssen stets eindeutig nachvollziehbar sein.*

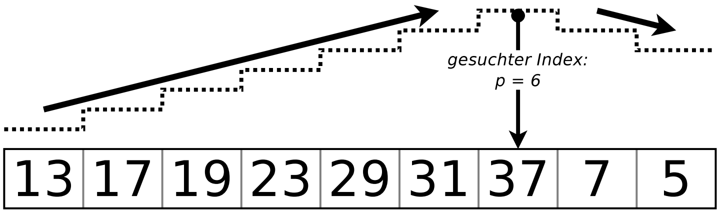
Beweisen Sie *formal* mittels *vollständiger Induktion* die partielle Korrektheit von `parS(n)` bzgl.:

$$\forall n \geq 1 : \texttt{parS(n)} \equiv (n + n\ mod\ 2) \cdot (n + 1)$$

Führen Sie *alle* Induktionsanfänge samt Beweise auf und verdeutlichen Sie den Induktionsschluss.
Geben Sie Ihre Lösung als `parS.pdf` über EST ab.

Gruppenaufgabe 9.3: Unimodale Suche **15 GP**

Eine Liste $\mathcal{L} = \{e_0, e_1, e_2, \ldots, e_n\}$ heißt *unimodal* bzw. *eingipflig* wenn es einen Index $0 \leq p \leq n$ so gibt, dass die Werte vom Anfang der Liste bis zum Index p bezüglich einer *Ordnungsrelation* „$\prec$“ streng monoton ansteigen ($e_0 \prec e_1 \prec e_2 \prec \ldots \prec e_p$) sowie die restlichen Werte ab Index p bis zum Ende der Liste streng monoton abfallen ($e_n \prec e_{n-1} \prec e_{n-2} \prec \ldots \prec e_p$) – zum Beispiel:



Java-Objekte haben oft eine sog. *natürliche Ordnung*, die durch das Interface Comparable ausgedrückt wird (z.B. die *lexikographische* Ordnung bei Strings). Objekte haben aber i.d.R. mehrere Eigenschaften, so dass dafür auch mehrere zusätzliche Ordnungsrelationen definiert werden können müssen – das wird in Java „von außen“ durch Comparatoren umgesetzt.
Implementieren Sie genau ***zwei*** verschiedene ***rekursive*** Methoden, um das Maximum einer (garantiert) *unimodalen* Liste $\mathcal{L}$ mit einer Laufzeit in $\mathcal{O}(log(n))$ (gemessen an der Anzahl der Zugriffe auf die n Listenelemente) zu bestimmen. Dabei sind weder der Datentyp der Listenelemente noch die Länge der Liste bekannt - stattdessen wird die Ordnung „$\prec$“ wie oben „Java-üblich“ ausgedrückt.
WICHTIG: Erfüllen Sie ***unbedingt alle*** zugehörigen öffentlichen Tests – insbesondere diejenigen, die die ***Rekursion*** und die „***Laufzeit***“ prüfen, andernfalls bekommen Sie dafür ***keine*** Punkte!

Gruppenaufgabe 9.4: wp-Kalkül **10 GP**

WICHTIG: *Geben Sie Ihre Beweise möglichst ausführlich an, d.h. einzelne Zwischenschritte (Umformungen gemäß wp-Axiome) müssen nachvollziehbar sein.* In dieser Aufgabe dürfen Sie grundsätzlich vereinfachend annehmen, dass es keinen Überlauf in den Code-Variablen geben kann.

```
long fib(long n) {
    long i = 0, a = 0, b = 1;
    while (i++ != n)
        b = a + (a = b);
    return a;
}
```

$$\forall n \in \mathbb{N}_0^+ : Fib_n := \begin{cases} 0 & \text{falls} \quad n \leq 0 \\ 1 & \text{falls} \quad n = 1 \\ Fib_{n-1} + Fib_{n-2} & \text{falls} \quad n \geq 2 \end{cases}$$

a) Schreiben Sie `fib` so in eine Methode `fibTrafo` um, dass beide funktional äquivalent sind, aber `fibTrafo` ausschließlich einfache Anweisungen pro Zeile hat (also z.B. nur „`i++;`“). Testen Sie Ihre `fibTrafo` unbedingt, ehe Sie hier weitermachen!

b) Geben Sie eine geeignete Schleifeninvariante I und Terminierungsfunktion $T(n)$ an, um die *totale Korrektheit* bzgl. $\forall n \in \mathbb{N}_0^+ :$ `fib(n)` $\equiv Fib_n$ zu beweisen.

c) Sei $init$ der Code-Block vor der Schleife und $P :\equiv n \geq 0$ die Vorbedingung. Beweisen Sie *formal* mittels wp-Kalkül: $P \Rightarrow wp(init, I)$

d) Sei SR der Schleifenrumpf und b die Schleifenbedingung Ihrer `fibTrafo`. Beweisen Sie *formal* mittels wp-Kalkül: $I \wedge b \Rightarrow wp(SR, I)$

e) Sei $Q :\equiv$ `a` $= Fib_n$ die Nachbedingung. Beweisen Sie *formal* mittels wp-Kalkül: $I \wedge \neg b \Rightarrow Q$.

35 EP + 25 GP = 60 Punkte