



Übungsblatt 11

Aufgabe

1. Syntax der Funktions-Typvereinbarung in Gofer (1,5 Punkte)

Beschreiben Sie die in Gofer benutzte Syntax zur Typvereinbarung von Funktionen unter Verwendung der Nichtterminale $\langle \text{Funktion} \langle \text{Typ} \rangle \rangle$, $\langle \text{Bezeichner} \rangle$ und $\langle \text{Typ} \rangle$

- graphisch als Syntaxdiagramm (Sie benötigen für jedes Nichtterminal jeweils ein Syntaxdiagramm.)
- textuell in BNF-Notation

Betrachten Sie als mögliche Typen lediglich Int, String und Bool. Für die Typvereinbarung genügt die Berücksichtigung von einfachen Datentypen (Bsp: $f :: \text{Int} \rightarrow \text{Int}$) und Listen (Bsp: $f :: [\text{Int}] \rightarrow \text{Int} \rightarrow [\text{Int}]$). Tupel und andere Strukturen sollen nicht berücksichtigt werden.

Mögliche Funktionsnamen sollen nur f, g, h sein.

Hinweis: Auch eine Liste von Listen ist ein Typ, der in Ihre Betrachtungen miteinbezogen werden soll.

Aufgabe

2. Erste Goferprogramme (2 Punkte)

Machen Sie sich mit der Gofer-Programmierungsumgebung vertraut, und erstellen Sie zu den folgenden Aufgabenstellungen Gofer-Programme. Testen Sie die Programme mit mindestens drei Testeingaben am Rechner.

- Berechnung der größeren von zwei Zahlen,
- Berechnung der Fakultätsfunktion,
- Berechnung des größten gemeinsamen Teilers zweier Zahlen,
- Berechnung der kleinsten Zahl aus einer Liste mit 4 Elementen.

Hinweis: In Gofer werden einige Funktionen wie z.B. die Funktionen max zur Berechnung des Maximums zweier Zahlen oder product zur Berechnung des Produktes aller Elemente einer Integer-Liste l bereitgestellt. Erstellen Sie Ihre Gofer-Programme ohne Verwendung dieser Funktionen.

Aufgabe

3. Gofer: Ganzzahlige Quadratwurzel (2 Punkte)

Erstellen Sie ein Gofer-Programm, das von einer ganzen Zahl die Quadratwurzel als ganzzahlige Näherung bestimmt. (vgl. Aufgabe Java: Quadratwurzel).

Bei ganzen Zahlen, für die keine exakte ganzzahlige Lösung existiert, gilt für die Annäherung: $i^2 \leq n < (i+1)^2$ ($i \in \mathbb{N}$ ist die angenäherte Quadratwurzel, n die ganze Zahl)

Beispiel: Quadratwurzel von 4 ist 2; Quadratwurzel von 5 ist 2;

Hinweis: Verwenden Sie zur Lösung der Aufgabe nicht die Gofer-Funktion `sqrt`

Aufgabe

4. Gofer: Rechenstruktur Keller (2 Punkte)

Schreiben Sie ein Gofer-Programm, das die Algebra Keller realisiert, die in der Vorlesung in Abschnitt 2.7 eingeführt wurde.

Das Programm soll also korrekte Kellerausdrücke beliebiger Form auswerten können. Für den Fall, dass ein Keller der Wert des Kellerausdrucks ist, so die Rückgabe zum Beispiel so aussehen:

Push (Push (Create) 3) 4

Um mit Kellern in Gofer operieren zu können, soll zunächst ein neuer (rekursiver) Datentyp definiert werden. Wählen Sie als Elemente dieses Typs den leeren Keller (Create) und einen beliebigen nicht leeren Keller der Form `push(k,t)` (k Keller, t Kellerelement, Umsetzung dieses Ausdrucks in Gofer-Syntax!)

Signatur und der Axiome der Algebra können dann unter Beachtung der Gofer-Syntax direkt umgesetzt werden. Umsetzung der Signatur in Gofer bedeutet hier Angabe der Funktionen mit samt ihrer Funktionalität.

Fügen Sie am Ende Ihres Programms folgende Kellerausdrücke zum Testen des Programms hinzu, die Sie durch Eingabe von `f1`, ..., `f5` ihrem Programm übergeben können.

`f1 = top(Push (Create) 3)`

`f2 = top(Push (Push (Create) 3) 4)`

`f3 = empty(pop(Push (Push (Create) 3) 4))`

`f4 = Push (pop(Push (Push (Create) 3) 4)) 2`

`f5 = pop(Push (pop(Push (Push (Create) 3) 4)) 2)`

Aufgabe

5. Gofer: Mergesort (2,5 Punkte)

Teilaufgabe 1:

Gegeben seien zwei sortierte Listen mit ganzen Zahlen. Schreiben Sie eine Gofer-Funktion `mmerge`, die diese beiden Listen zu einer sortierten Liste zusammenfügt.

Teilaufgabe 2:

Schreiben Sie eine Gofer-Funktion `mergesort`, die eine Liste mit ganzen Zahlen mit Hilfe des Mergesort-Verfahrens (s.u.) sortiert.

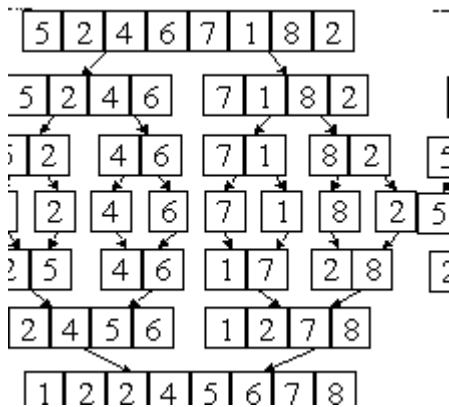
Hinweis: Zum Aufteilen der Liste können Sie die Gofer-Funktionen `drop n l` (gibt `l` ab der `n`-ten Stelle zurück) bzw. `take n l` (gibt `n` bis zur `(n-1)`-ten Stelle zurück) verwenden. `length l` gibt die Länge einer Liste an.

Beispiel: `drop 3 [1,2,3,4,5] = [4,5]` und `take 3 [1,2,3,4,5] = [1,2,3]`

Prinzip Mergesort:

Zerteile die gegebene Liste solange, bis der Basisfall, Liste mit einem Element, eintritt.

Dann verschmelze je zwei Listen (mittels `mmerge`) zu einer doppelt so langen, sortierten Liste, bis die gesamte Liste sortiert ist.



Pseudocode Mergesort:

`mergesort (l):`

falls Länge von `l = 1`

return `l`

sonst

zerteile `l` in gleichlange listen `l1` und `l2`

`mergesort(l1), mergesort(l2)`

verschmelze sortierte listen `l1` und `l2` mit `mmerge`