

Programmierung und Modellierung, SS 2008  
 Übungsblatt 2

Abgabe: bis Di 06.05.2008 (nachts)

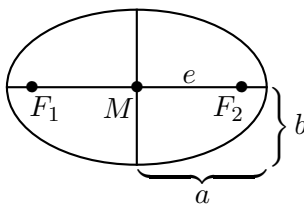
Besprechung Do 08. – Di 13.05.

Aufgabe 2-1 Ellipsenberechnung

SML

Eine Ellipse um einen Mittelpunkt  $M$  ist gegeben durch ihre große Halbachse  $a$  und ihre kleine Halbachse  $b$ . Auf der großen Achse liegen die beiden Brennpunkte  $F_1, F_2$  der Ellipse. Den Abstand  $e$  dieser Brennpunkte vom Mittelpunkt nennt man die Exzentrizität der Ellipse.

Die Exzentrizität hat einen Wert zwischen 0 und  $a$  und ist ein Maß dafür, wie „länglich“ die Ellipse ist. Die Exzentrizität hat den Wert 0 im Fall  $a = b$ , wenn also die Ellipse zu einem Kreis mit Radius  $a$  entartet. Die Exzentrizität hat den Wert  $a$  im Fall  $b = 0$ , wenn also die Ellipse zu einer Strecke der Länge  $2a$  entartet.



Ellipse mit Halbachsen  $a$  und  $b$  mit  $a \geq b \geq 0$

Flächeninhalt	$I$	$=$	$\pi ab$
Exzentrizität	$e$	$=$	$\sqrt{a^2 - b^2}$
Umfang	$U$	$=$	$\pi \left( \frac{3}{2}(a+b) - \sqrt{ab} \right)$

In der obigen Tabelle sind die Formeln zur Berechnung von Flächeninhalt, Exzentrizität und Umfang angegeben. (Die Formel für den Umfang liefert nur einen – allerdings ziemlich genauen – Näherungswert, weil der exakte mathematische Wert durch eine unendliche Reihe gegeben ist, also als Grenzwert einer Summe mit unendlich vielen Summanden.)

Definieren Sie die folgenden SML-Funktionen zur Berechnung der Werte gemäß der obigen Formeln:

```
ellipse_flaecheninhalt(a,b)
ellipse_exzentrizitaet(a,b)
ellipse_umfang(a,b)
```

Sie brauchen nicht zu überprüfen, dass die Bedingung  $a \geq b \geq 0$  tatsächlich eingehalten wird.

**Hinweis:** Die SML-Struktur `Math` enthält trigonometrische und logarithmische Funktionen und andere mathematische Hilfsmittel. Die Variable `Math.pi` hat den Wert  $\pi$ . Die Funktion `Math.sqrt` erwartet ein Argument vom Typ `real` und berechnet dessen Quadratwurzel (der Name `sqrt` ist die Abkürzung der englischen Bezeichnung *square root* für Quadratwurzel).

Aufgabe 2-2 Rekursive Definition

SML

Versicherungen *Gierig* und *Fähig* bieten folgendes Rentenprodukt an. Sie zahlen zu Beginn jedes Jahres `einzahlung` EUR in Ihren Vertrag ein. Am Ende jedes Jahres wird das aufgelaufene Gesamtkapital mit `zinssatz` Prozent verzinst. Die Zinsen werden dem Kapital zugeführt. Nach `laufzeit` Jahren bekommen Sie das Gesamtkapital ausbezahlt. Sie möchten einen Vertrag über 40 Jahre mit einer jährlichen Einzahlung von 1500 EUR abschließen.

Ein Vertreter von *Gierig* bietet das Produkt mit einem Zinssatz von 4.0% an, von *Fähig* bekämen Sie das Produkt mit 5.0%. Nun bezahlt ihnen der Vertreter von *Gierig* eine 2wöchige all-inclusive

Karibik-Reise, falls Sie bei Ihm abschließen. Wieviel Prozent Ihrer Rente würde Ihnen diese Karibik-Reise letztlich kosten?

Zur Beantwortung dieser Frage schreiben Sie sich eine SML-Funktion

```
fun riester (einzahlung, zinssatz : real, laufzeit : int) : real
```

die das angelaufene Kapital nach `laufzeit` Jahren bei einer jährlichen Zahlung von `einzahlung` EUR und einer jährlichen Verzinsung von `zinssatz`% berechnet. Testen Sie sie mit den obigen Daten. (Hinweis: Abzugeben ist nur die Funktion `riester`.)

### Aufgabe 2-3 Rekursive Definition

SML

In dieser Aufgabe sollen natürliche Zahlen als Strings von Ziffern repräsentiert werden, zum Beispiel "238" oder "4711". Strings wie "23a8" oder "" repräsentieren dagegen keine natürlichen Zahlen. Für diese Repräsentation sind folgende Funktionen gegeben:

```
isNat: string -> bool
```

true wenn das Argument eine natürliche Zahl repräsentiert, sonst false.

`isNat("238")` hat Wert `true`, `isNat("23a8")` hat Wert `false`.

```
isnoNat: string -> bool
```

true wenn das Argument keine natürliche Zahl repräsentiert, sonst false.

`isnoNat("23a8")` hat Wert `true`, `isnoNat("238")` hat Wert `false`.

```
oneDigitNat: string -> bool
```

true wenn das Argument eine einstellige natürliche Zahl repräsentiert, sonst false.

`oneDigitNat("2")` hat Wert `true`, `oneDigitNat("4711")` hat Wert `false`.

```
headNat: string -> string
```

Das Argument muss eine natürliche Zahl repräsentieren.

Liefert die Repräsentation der Zahl, die aus der linkensten Ziffer des Arguments besteht.

`headNat("4711")` hat Wert `"4"`, `headNat("")` verursacht einen Fehler.

```
tailNat: string -> string
```

Das Argument muss eine mehrstellige natürliche Zahl repräsentieren.

Liefert die Repräsentation der Zahl, die aus allen Ziffern des Arguments mit Ausnahme der linkensten Ziffer besteht.

`tailNat("4711")` hat Wert `"711"`, `tailNat("2")` verursacht einen Fehler.

```
appendNat: string * string -> string
```

Liefert die Repräsentation der Zahl, die sich ergibt,

wenn die beiden Argumente aneinandergehängt werden.

`appendNat("238", "4711")` hat Wert `"2384711"`.

In der WWW-Seite der Vorlesung finden Sie unter „Dateien“ unter anderem die Datei `2-3.sml` mit einer Implementierung der obigen Funktionen. Kopieren Sie sich die Datei in Ihr Unterverzeichnis für dieses Übungsblatt. **Die Definitionen in dieser Datei dürfen nicht verändert werden**, und Sie brauchen sie nicht einmal zu lesen. Es reicht, wenn Sie die obige Spezifikation verstehen und die gegebenen Funktionen entsprechend dieser Spezifikation verwenden.

Ergänzen Sie Ihre Kopie der Datei um Definitionen der folgenden beiden Funktionen und geben Sie die modifizierte Datei `2-3.sml` als Lösung ab.

```
shiftNat: string * string -> string
```

Liefert die Repräsentation einer natürlichen Zahl, die aus dem zweiten Argument entsteht,

wenn man so viele Nullen rechts anhängt wie das erste Argument Ziffern enthält.

`shiftNat("4711", "123")` hat Wert `"1230000"`.

```
normalizedNat: string -> string
```

Liefert die Repräsentation einer natürlichen Zahl, die aus dem Argument entsteht,

wenn alle führenden Nullen weggelassen werden.

```
normalizedNat("004711") hat Wert "4711", normalizedNat("0") hat Wert "0",  
normalizedNat("4711") hat Wert "4711", normalizedNat("400") hat Wert "400".
```

Hinweis: Sie dürfen voraussetzen, dass `shiftNat` und `normalizedNat` nur mit gültigen Zahlen aufgerufen werden, d.h. Sie müssen nicht auf `isNat` testen.

#### Aufgabe 2-4 Boole'sche Ausdrücke

In dieser Aufgabe sollen natürliche Zahlen als Strings von Ziffern repräsentiert werden, zum Beispiel "238" oder "4711". Strings wie "23a8" oder "" repräsentieren dagegen keine natürlichen Zahlen. Für diese Repräsentation seien folgende Funktionen gegeben:

```
oneDigitNat: string -> bool  
true wenn das Argument eine einstellige natürliche Zahl repräsentiert, sonst false.  
oneDigitNat("2") hat Wert true, oneDigitNat("4711") hat Wert false.
```

```
headNat: string -> string  
Das Argument muss eine natürliche Zahl repräsentieren.  
Liefert die Repräsentation der Zahl, die aus der linkensten Ziffer des Arguments besteht.  
headNat("4711") hat Wert "4", headNat("") verursacht einen Fehler.
```

```
tailNat: string -> string  
Das Argument muss eine mehrstellige natürliche Zahl repräsentieren.  
Liefert die Repräsentation der Zahl, die aus allen Ziffern des Arguments  
mit Ausnahme der linkensten Ziffer besteht.  
tailNat("4711") hat Wert "711", tailNat("2") verursacht einen Fehler.
```

Darauf aufbauend seien die folgenden beiden Funktionsdefinitionen gegeben. Beide Varianten der Funktion sollen testen, ob ihr Argument eine zweistellige natürliche Zahl repräsentiert.

```
fun twoDigitNat(nString) =  
  not(oneDigitNat(nString)) andalso oneDigitNat(tailNat(nString));  
fun twoDigitNat'(nString) =  
  oneDigitNat(tailNat(nString)) andalso not(oneDigitNat(nString));
```

- Für welche Argumente verhalten sich die beiden Funktionen unterschiedlich?
- Was ist der Grund dafür, dass sie sich überhaupt unterschiedlich verhalten können?