10. Übungsblatt zu Informatik 1 WS 98/99

Andreas Rossberg, Prof. Gert Smolka

Abgabe: 22. Januar 1999 vor der Vorlesung

Hinweis: Alle vorgegebenen Programme finden Sie unter `http://www.ps.uni-sb.de/courses/info1-ws98/uebung/sml/`. Dort finden Sie auch ein kleines Beispielprogramm `filesize.sml`, das zeigt, wie Dateien gelesen werden können und die Kommandozeilenargumente bei einem externen Programm ermittelt werden. Im Unterverzeichnis `mosml/` finden Sie zudem die dokumentierten Signaturen aller von Moscow ML zur Verfügung gestellten Standardstrukturen. Lesen Sie dazu die entsprechenden Kapitel in Paulson (Kapitel 8.7–8.9) bzw. Ullman (Kapitel 4) lesen.

Bitte alle Programme per Email abgeben.

Aufgabe 10.1: Große natürliche Zahlen (6 Punkte)

Unter `http://www.ps.uni-sb.de/courses/info1-ws98/uebung/sml/` finden Sie die Signatur und eine unvollständige Implementierung einer Struktur `Bignat`, die natürliche Zahlen beliebiger Größe verwirklicht.

(a) Vollziehen Sie die Funktionsweise der Implementierung nach und vervollständigen Sie sie um die Prozeduren `sub` (Subtraktion mit Übertrag) und `exp` (Exponentiation). Achten Sie bei `sub` auf die Elimination von führenden Nullen!

(b) Schreiben Sie eine Prozedur zur Berechnung der Fakultät unter Verwendung von `Bignat`. Schreiben Sie weiterhin eine Prozedur `digits: int -> int`, so dass `digits` n die Anzahl der Dezimalstellen von $n!$ berechnet. Wieviele Dezimalstellen hat die Fakultät von 300?

Aufgabe 10.2: Große ganze Zahlen (8 Punkte)

Schreiben Sie eine Struktur `Bigint`, die den abstrakten Datentyp `bigint` für beliebig große ganze Zahlen realisiert und folgende Signatur erfüllt:

```
eqtype bigint

val fromInt :       int    -> bigint
val toInt :         bigint -> int
val toString :      bigint -> string

val < :             bigint * bigint -> bool
val <= :            bigint * bigint -> bool
val isEven :        bigint          -> bool
val isNegative :    bigint          -> bool

val ~ :             bigint          -> bigint
val + :             bigint * bigint -> bigint
val - :             bigint * bigint -> bigint
val * :             bigint * bigint -> bigint
val div :           bigint * bigint -> bigint
val mod :           bigint * bigint -> bigint
val abs :           bigint          -> bigint
val exp :           bigint * bigint -> bigint
```

(Diese Signatur finden Sie unter `http://www.ps.uni-sb.de/courses/info1-ws98/uebung/sml/`)

Implementieren Sie den Typ `bigint` als

```
type bigint = bool * Bignat.bignat
```

wobei das boolesche Flag mit `true` anzeigt, dass die Zahl negativ ist. Die Prozedur `toString` soll (entsprechend der SML-Syntax) `~` als negatives Vorzeichen verwenden. Hinweis: Achten Sie darauf, dass `bigint` ein Typ mit Gleichheit sein soll, also muss insbesondere die Zahl 0 in eindeutiger Weise dargestellt werden.

Aufgabe 10.3: Reverses Echo (5 Punkte)

Schreiben Sie ein direkt ausführbares Programm mit einer Kommandooberfläche, das zeilenweise Eingaben vom Benutzer einliest, und diese in zeichenweise umgekehrter Reihenfolge wieder ausgibt. Dazu soll jeweils ein einfacher Prompt ausgegeben werden. Das Programm wird beendet, indem der Benutzer mit Ctrl-D ein Dateiende-Zeichen eingibt.

Aufgabe 10.4: Zeilennummern (7 Punkte)

Schreiben Sie ein direkt ausführbares Programm `num`, welches als Argumente zwei Dateinamen erwartet. Es soll die mit dem ersten Namen bezeichnete Datei einlesen und eine neue Datei unter dem an zweiter Stelle angegebenen Namen erzeugen. Die Zeichen der Quelldatei werden in die Zieldatei kopiert, wobei aber jeder Zeile ihre Zeilennummer vorangestellt wird. Zum Beispiel soll aus der Datei `foo.txt`:

```
bla bla bla
gnulp gnulp
palpenpumpf
```

durch den Aufruf

```
% num foo.txt bar.txt
```

eine Datei `bar.txt`:

```
1: bla bla bla
2: gnulp gnulp
3: palpenpumpf
```

erzeugt werden.

Aufgabe 10.5∗: Dateigrößen (0+3 Punkte)

Verzichtet man auf das Schließen der Dateien, so lässt sich das oben erwähnte Beispielprogram `filesize` in einem einzigen Ausdruck formulieren. Dieser Ausdruck enthält kein `let`, sondern verwendet nur die Prozedurkomposition `o` und Prozeduren aus Standardstrukturen (insbesondere `List`). Finden Sie diesen Ausdruck.