11. Übungsblatt „Programmiersprachen“ WS 97/98

Leif Kornstaedt, Prof. Gert Smolka

Ausgabe: 29. Januar 1998 **Abgabe:** 9. Februar 1998

XI.1: Listen in Oz (4 + 4 Punkte)

a. Schreiben Sie eine Prozedur zum Aneinanderhängen zweier Listen in Oz. Verwenden Sie hierbei funktionale Notation:

```
fun {MyAppend Xs Ys} ... end
```

b. Formulieren Sie obige Prozedur mit relationaler Notation:

```
proc {MyAppend Xs Ys Zs} ... end
```

Ihre Lösung soll endrekursiv sein.

XI.2: Backpatching über logische Variablen (8 Punkte)

Gegeben sei eine fiktive Assemblersprache mit folgenden Instruktionen:

Instruktion	Bedeutung
`lbl`(x)	Markiert die folgende Instruktion mit dem symbolischen Label x.
`branch`(x)	Springt zu der Instruktion mit dem symbolischen Label x.
`noop`	Sonstige Instruktionen.

Die symbolischen Labels x seien hierbei durch Oz-Atome gegeben. Der erzeugte Maschinencode für die Instruktionen `branch`(x) und `noop` sei jeweils ein Wort lang; `lbl`(x) ist eine Pseudo-Instruktion, für die kein Code erzeugt wird. Schreiben Sie eine Prozedur `ComputeLabels`, die eine Liste von Assemblerinstruktionen wie oben erhält und eine Liste zurückliefert, in der alle Auftreten symbolischer Labels durch die Offsets ab Codeanfang der zugehörigen `lbl`-Instruktionen ersetzt worden sind. Zudem sollen die Pseudo-Instruktionen in der Ergebnisliste nicht mehr erscheinen. Sie dürfen davon ausgehen, daß kein Label mehrfach definiert wird und daß keine Referenzen auf undefinierte Labels in dem eingegebenen Code auftreten.

Beispiel:

```
{ComputeLabels [lbl(y) noop branch(x) lbl(x) noop branch(y)]}
= [noop branch(2) noop branch(0)]
```

XI.3: Nebenläufigkeit 1: k-Semaphor (9 Punkte)

Schreiben Sie eine Prozedur zum Erzeugen eines k-Semaphors wie folgt:

```
proc {NewSemaphore K S} ... end
```

Das Eingabeargument `K` muß eine positive ganze Zahl sein, die angibt, wie oft das Semaphor gleichzeitig betreten werden darf, und das Ausgabeargument `S` enthält nach Ausführung von `NewSemaphore` eine einstellige Prozedur (vgl. die in der Vorlesung vorgestellte Modellierung von Locks). Bei einer Applikation

```
{S proc {$} ... end}
```

wird das Semaphor betreten, sobald dies möglich ist, und die übergebene nullstellige Prozedur ausgeführt, woraufhin das Semaphor wieder verlassen wird. Achten Sie darauf, daß das Semaphor den Zutritt fair gewährt.

XI.4: Nebenläufigkeit 2: Synchroner Kanal (8 Punkte)

Implementieren Sie einen synchronen Kanal. Ein synchroner Kanal hat Operationen `Put` und `Get`, die wie folgt definiert sind: Die Operation `{Put X}` blockiert solange, bis an dem Kanal eine unbefriedigte `{Get Y}`-Anfrage anliegt und die Operation `{Get Y}` blockiert solange, bis an dem Kanal eine unbefriedigte `{Put X}`-Operation anliegt. Haben sich zwei Anfragen gefunden, so wird `X = Y` gesetzt und die Threads laufen weiter.

Implementieren Sie synchrone Kanäle durch eine Prozedur

```
proc {NewSynchronousChannel Put Get} ... end
```

Nach Applikation dieser Prozedur stehen in den beiden Ausgabeargumenten einstellige Prozeduren, die die genannten Operationen realisieren. Achten Sie darauf, daß bei jedem Aufruf von `NewSynchronousChannel` ein neuer synchroner Kanal eröffnet wird.