Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. F. Kröger, Dr. M. Hölzl, F. Hacklinger

Probeklausur Informatik I
WiSe 05/06
18. Januar 2006, 10–12 Uhr

Vorname	.....
Nachname	.....
Matrikelnummer	.....
Universität	☐ Ludwig-Maximilians-Universität ☐ Technische Universität
Studienfach	☐ Informatik (Diplom) ☐ Bioinformatik (Bachelor)
Fachsemester	.....

1	2	3	4	5	6	Σ	Note

Aufgabe 1 **Verständnisfragen** (2+1+2 Punkte)

a) Begründen Sie informell, warum folgende SML-Funktion für alle Eingaben $x \in \mathbb{Z}$ terminiert.

```
fun f x = if x = 0 then 0
          else if x < 0 then f(~x + 1)
          else f(x-1)
```

b) Warum ergibt die Eingabe von

```
fun g x = x * let x = 2 in x * x end + x;
```

in einer SML-Sitzung eine Fehlermeldung?

c) Ist die Funktion

```
fun h [] = []
  | h (x::xs) = (h xs) @ [x,x]
```

polymorph? Begründen Sie Ihre Antwort.

Aufgabe 2 **SML** (1+2+2+3+4 Punkte)

a) Geben Sie ein SML-Programm *fromTo* vom Typ **int** $*$ **int** $\rightarrow$ **int list** an, so dass ein Aufruf *fromTo*(m, n) die Liste der ganzen Zahlen von m bis n zurückgibt. (Diese Liste ist leer, falls $m > n$.) Zum Beispiel gibt *fromTo*$(5, 8)$ die Liste $[5, 6, 7, 8]$ zurück.

b) Geben Sie ein SML-Programm *listpos* vom Typ $'a$ **list** $\rightarrow$ (**int** $*$ $'a$) **list** an, das jeder Liste l eine Liste l' zuordnet, deren Element an der Position i das Paar (i, l_i) ist, wobei l_i das i-te Element von l ist. Zum Beispiel soll *listpos*$[3, 5, 9]$ die Liste $[(1, 3), (2, 5), (3, 9)]$ ergeben.

c) Geben Sie ein SML-Programm *prefix* vom Typ **string** $*$ **string list** $\rightarrow$ **string list** an, so dass ein Aufruf *prefix*(p, l) eine Liste zurückgibt, deren i-tes Element die Konkatenation von p mit dem i-ten Element aus l ist. Zum Beispiel soll *prefix*(`"Dr. "`, [`"Maier"`, `"Walter"`, `"Huber"`]) die Liste [`"Dr. Maier"`, `"Dr. Walter"`, `"Dr. Huber"`] zurückgeben.

d) Geben Sie ein SML-Programm *take3* vom Typ $'a$ **list** $\rightarrow$ $'a$ **list** an, so dass ein Aufruf *take3 l* für eine Liste l mit n Elementen die ersten $n/3$ Elemente von l zurückgibt, falls n durch 3 teilbar ist, und in allen anderen Fällen eine Ausnahme auslöst.

e) Geben Sie ein SML-Programm *tfind* vom Typ ($'a \rightarrow$ **bool**) $*$ ($'a$ **bintree**) $\rightarrow$ $'a$ **list** an, so dass ein Aufruf *tfind*(f, b) eine Liste l zurückgibt, für die gilt: l enthält genau die Knoten x von b, für die $f(x)$ den Wert *true* ergibt.

Aufgabe 3 **Datatype-Deklaration** (3+3 Punkte)

Gegeben sei die folgende *datatype*-Deklaration:

```
datatype 'a TwoThree = nothing
                     | two of 'a * 'a TwoThree * 'a TwoThree
                     | three of 'a * 'a TwoThree * 'a TwoThree * 'a TwoThree
```

Bei Elementen des Datentyps *TwoThree*, die in der Form $two(d, t1, t2)$ dargestellt werden können, nennt man d den Inhalt und $t1, t2$ die direkten Substrukturen. Bei Elementen, die in der Form $three(d, t1, t2, t3)$ dargestellt werden können, nennt man d den Inhalt und $t1, t2, t3$ die direkten Substrukturen. Elemente, die in der Form *nothing* dargestellt werden können, haben weder Inhalt noch direkte Substrukturen. Die Substrukturen eines Elements t des Datentyps *TwoThree* sind t selbst und alle Substrukturen der direkten Substrukturen von t.

a) Geben Sie eine SML-Funktion *flatten* vom Typ $'a\ TwoThree \rightarrow 'a$ **list** an, so dass ein Aufruf $flatten(t)$ eine Liste mit den Inhalten aller Substrukturen von t zurückgibt.

b) Geben Sie eine SML-Funktion *count23* vom Typ $'a\ TwoThree \rightarrow$ **int** $*$ **int** an, so dass ein Aufruf $count23(t)$ ein Paar $(n2, n3)$ zurückgibt, für das gilt:

- $n2$ ist die Anzahl der Substrukturen von t, die in der Form $two(d, t1, t2)$ dargestellt werden können.
- $n3$ ist die Anzahl der Substrukturen von t, die in der Form $three(d, t1, t2, t3)$ dargestellt werden können.

Aufgabe 4 **Bäume** (5 Punkte)

Ein Baum, bei dem jeder Knoten beliebig viele Unterbäume haben kann, lässt sich durch folgende *datatype*-Deklaration beschreiben:

```
datatype 'a ntree = n_empty | n_build of 'a * ('a ntree) list
```

Die Knoten eines derartigen Baums sind induktiv definiert: *n_empty* hat keine Knoten, die Knoten von $n\_build(k, ts)$ sind k und alle Knoten von Elementen aus ts.

Geben Sie eine SML-Funktion *contains* vom Typ $''a * ''a\ ntree \rightarrow \textbf{bool}$ an, so dass ein Aufruf $contains(x, t)$ genau dann *true* zurückgibt, wenn x in t als Knoten vorkommt.

Aufgabe 5 **Terminierung** (2+4 Punkte)

Betrachten Sie die folgende SML-Funktion:

```
fun f (x, y) = if (x + y) < 0 then ~1 else 1 + f(x + 2, y - 3)
```

a) Werten Sie $f(31, -28)$ aus. (Notieren Sie die Auswertung in „Kurzschreibweise“.)

b) Beweisen Sie, dass die Berechnung von $f(x, y)$ für alle Werte $(x, y) \in \mathbb{Z} \times \mathbb{Z}$ terminiert.

Aufgabe 6 **BNF-Grammatiken** (4+2 Punkte)

Es sei $\Sigma = \{a, \ldots, z, A, \ldots, Z, \_\}$. Eine Zeichenreihe $w \in \Sigma^*$ ist ein zulässiger Funktionsname in der Programmiersprache C, falls w nicht leer ist und eine der folgenden Bedingungen gilt:

- w beginnt nicht mit einem Unterstrich ($\_$).
- w beginnt mit einem Unterstrich ($\_$), gefolgt von einem Kleinbuchstaben (a–z), gefolgt von einer beliebigen (möglicherweise leeren) Zeichenreihe aus Σ^*.

Zum Beispiel sind *ABC*, *printf*, $\_\,x$, $\_\,aName$ zulässige Funktionsnamen in C-Programmen, die Zeichenreihen $\_\_\,x$, $\_\,X$ und $\_\,Name$ dagegen nicht.

Wir setzen $\mathcal{L} = \{w \in \Sigma^* \mid w \text{ ist zulässiger Funktionsname in C}\}$.

a) Geben Sie eine BNF-Grammatik G mit Startzeichen S an, so dass $\mathcal{L}(S) = \mathcal{L}$.

b) Leiten Sie *printf* und $\_\,x$ aus dem Startsymbol S Ihrer Grammatik ab.