TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Sprachen und Beschreibungsstrukturen
Einführung in die Informatik 2
Prof. Dr. Helmut Seidl, T. M. Gawlitza, S. Pott, M. Schwarz

WS 2008/09
3. Merkblatt
19.11.08

Zur Lösung der folgenden Aufgabenblätter benötigen Sie das unter

http://wwwseidl.in.tum.de/lehre/vorlesungen/WS08/info2/mo.zip

herunterladbare ZIP-Archiv, das das Modul `Mo` enthält, das einen Datentypen für Terme und Funktionen zur Ein- und Ausgabe von Termen bereitstellt.

1 Verwendung des Moduls `Mo`

Durch Entpacken der Datei `mo.zip` entsteht ein Ordner `mo`, der folgende Dateien enthält: `Makefile`, `calc.ml`, `mo.cmi`, `mo.cma` und `test.term`. Wechseln Sie in diesen Ordner. Folgender Befehl ruft OCaml mit dem Modul `Mo` auf:

```
ocaml mo.cma
```

Öffnen Sie anschließend das Modul `Mo` durch den folgenden Befehl:

```
# open Mo ;;
```

Danach stehen die folgenden Funktionen zur Verfügung:

```
val term_from_stdin : unit -> term
val term_from_file : string -> term
val string_from_term : term -> string
```

Der Aufruf `term_from_stdin ()` liest einen *Term* von der Standard-Eingabe. Der Aufruf `term_from_file filename` liest den in der Datei `filename` abgelegten *Term* ein. Der Aufruf `string_from_term t` erzeugt eine Stringdarstellung des Terms `t`. Der Typ `term` ist wie folgt definiert:

```
type term = Node of string * term list
```

Also ein Paar aus einem Label vom Typ `string` und einer Liste weiterer Terme vom Typ `term`. Beispielsweise wird der Term

$f(a, b)$

durch den Wert

```
Node ("f ", [Node ( "a ", []); Node ( "b ", [])])
```

vom Typ `term` repräsentiert.

Geben Sie

```
# let t = term_from_stdin () ;;
```

ein. Jetzt wartet das System auf die Eingabe eines Terms, die sie mit `;;` abschließen müssen. Auf die Eingabe

```
f(a,b) ;;
```

antwortet OCaml mit

```
val t : term = Node ("f", [Node ("a", []); Node ("b", [])])
```

Die Variable `t` bezeichnet jetzt den eingelesenen Term. Diesen können Sie weiterverarbeiten. Beispielsweise können Sie die Funktion `mirror : term -> term` wie folgt definieren:

```
# let rec mirror (Node(x,ts)) =
       Node(x,List.rev (List.map mirror ts)) ;;
```

Der Aufruf `# mirror t;;` liefert schließlich:

```
# - : Mo.term = Node ("f", [Node ("b", []); Node ("a", [])])
```

Der Term ist also gespiegelt worden.

2 Das Taschenrechner-Beispiel

Die Datei `calc.ml` enthält die Implementierung eines „Taschenrechners“. Kompilieren können Sie die Datei durch den Aufruf

```
ocamlc -o calc mo.cma calc.ml
```

bzw. unter Windows:

```
ocamlc -o calc.exe mo.cma calc.ml
```

Es entsteht eine ausführbare Datei namens `calc`. Rufen Sie das erzeugte Programm mit `./calc` (bzw. `calc` unter Windows) auf. Geben Sie z.B. `+(5,*(2,3));;` ein. Dieser Term repräsentiert den arithmetischen Ausdruck $5 + 2 \cdot 3$. Entsprechend wird das Ergebnis 11 ausgegeben.