Übungsblatt 13 (Lösungshinweise)

3 Aufgaben, 20 Punkte

Objektorientierte Modellierung und Programmierung (inf031) Sommersemester 2020
Carl von Ossietzky Universität Oldenburg, Fakultät II, Department für Informatik

Dr. C. Schönberg

Ausgabe: 2020-07-14 14:00 **Abgabe:** –

Hinweis: Dieses Übungsblatt wird nicht mehr korrigiert. Sie finden aber Lösungen bzw. Lösungshinweise unter den Aufgaben.

Aufgabe 1: *Standard ML* **(1 + 2 + 2 + 2 + 1 Punkte)**

Schreiben Sie folgende Funktionen in Standard ML:

a) `max = fn : int list -> int`

die das Maximum einer Liste von Integer-Werten bestimmt.
Das Maximum der leeren Liste soll 0 sein.

b) `nth = fn : int list * int -> int`

die das n-te Element einer Liste von Integer-Werten zurückgibt.
Bei einer leeren Liste soll 0 zurückgegeben werden.
Sie dürfen *nicht* die Funktion `List.nth` verwenden!

c) `reverse = fn : 'a list -> 'a list`

die eine Liste von beliebigen Werten umdreht.
Das Ergebnis von z.B. `reverse([1, 2, 3]);` soll `[3, 2, 1]` sein.

d) `insert = fn : 'a * 'a list * ('a * 'a -> bool) -> 'a list`

die einen Wert in eine Liste einfügt, und zwar genau an der Stelle, an der die ebenfalls übergebene Funktion zum ersten Mal `true` ergibt. Die übergebene Funktion vergleicht zwei Werte.
Hinweis: Mit `(op <)` können Sie die üblichen Vergleichs-Operatoren als Funktion verwenden.
Beispiel: Der Aufruf von `insert(3, [1,2,4,5], (op <));` soll `[1, 2, 3, 4, 5]` ergeben.

e) `testInverse = fn : (''a -> ''b) * (''b -> ''a) * ''a * ''b -> bool`

die für zwei übergebene Funktionen $f : \alpha \to \beta$ und $g : \beta \to \alpha$ prüft, ob g die Umkehrfunktion von f ist, d.h. ob gilt $f = g^{-1}$ und $g = f^{-1}$.
Da die Prüfung nicht auf dem gesamten (unbekannten!) Definitionsbereich bzw. Wertebereich von f und g erfolgen kann, werden zusätzlich zwei Werte $x : \alpha$ und $y : \beta$ übergeben. *Ausschließlich* für diese beiden Werte soll die Prüfung erfolgen.

Lösungshinweise 1:

```
fun max(nil) = 0
  | max(x::xs) = if x > max(xs) then x else max(xs);

fun nth(nil, n) = 0
  | nth(x::xs, 0) = x
  | nth(x::xs, n) = nth(xs, n-1);

fun reverse(nil) = nil
  | reverse(x::nil) = [x]
  | reverse(x::xs) = reverse(xs) @ [x];

fun insert(x, nil, bef) = [x]
  | insert(x, y::ys, bef) = if bef(x,y)
                            then x::y::ys
                            else y::insert(x, ys, bef);

fun testInverse(f, g, x, y) = f(g(y))=y andalso g(f(x))=x;
testInverse((fn x => x+1), (fn y => y-1), 2, 2);
testInverse((fn x => x+1), (fn y => y*2), 2, 2);
testInverse((fn x => ord(x)), (fn y => chr(y)), #"A", 65);
```

Aufgabe 2: *Prolog* **(6 Punkte)**

Schreiben Sie ein Prolog-Programm mit folgender Faktenbasis:

- alice hat ein Kind (`haschild`): charlie.
- bob hat ein Kind: charlie.
- edith hat ein Kind: alice.
- alice und edith sind Frauen (`woman`), bob und charlie sind Männer (`man`).

Fügen Sie außerdem folgende Regeln hinzu:

- Eltern (`parent`) sind Individuen, die mindestens ein Kind haben.
- Großeltern (`grandparent`) sind Individuen, die mindestens ein Kind haben, welches wiederum mindestens ein Kind hat.
- Väter (`father`) sind Eltern, die Männer sind.
- Mütter (`mother`) sind Eltern, die Frauen sind.
- Großväter (`grandfather`) sind Großeltern, die Männer sind.
- Großmütter (`grandmother`) sind Großeltern, die Frauen sind.

Geben Sie *zusätzlich* an, welche Antworten die folgenden Anfragen zurückliefern:

- `?- woman(alice).`
- `?- man(alice).`
- `?- haschild(X, charlie).`
- `?- parent(X).`
- `?- grandmother(X).`
- `?- parent(charlie).`

Lösungshinweise 2:

```
haschild(alice, charlie).
haschild(bob, charlie).
haschild(edith, alice).
woman(alice).
woman(edith).
man(bob).
man(charlie).
parent(X) :- haschild(X, _).
grandparent(X) :- haschild(X, Y), haschild(Y, _).
father(X) :- parent(X), man(X).
mother(X) :- parent(X), woman(X).
grandfather(X) :- grandparent(X), man(X).
grandmother(X) :- grandparent(X), woman(X).

?- woman(alice). -> yes.
?- man(alice). -> no.
?- haschild(X, charlie). -> alice, bob.
?- parent(X). -> alice, bob, edith.
?- grandmother(X). -> edith.
?- parent(charlie). -> no.
```

Aufgabe 3: *Drools* **(6 Punkte)**

Gegeben seien folgende drei Java-Klassen:

```
1  public class Person {
2
3    private String name;
4
5    public Person(String name) {
6      super();
7      this.name = name;
8    }
9
10   public String getName() {
11     return name;
12   }
13
14   public void setName(String name) {
15     this.name = name;
16   }
17
18 }
19 public class Message {
20
21   private String sender;
22   private String receiver;
23   private String text;
24
25   public Message(String sender, String receiver, String text) {
26     super();
27     this.sender = sender;
28     this.receiver = receiver;
29     this.text = text;
30   }
31
32   // getters and setters...
33
34 }
35 public class MessageRunner {
36
37   public static void main(String[] args) {
38     KieServices kieServices = KieServices.Factory.get();
39     KieContainer kContainer = kieServices.getKieClasspathContainer();
40     KieSession kSession = kContainer.newKieSession("ksession-rules");
41
42     kSession.insert(new Person("Frodo"));
43     kSession.insert(new Person("Samwise"));
44     kSession.insert(new Person("Peregrin"));
45     kSession.insert(new Person("Meriadoc"));
46     kSession.insert(new Message("Frodo", "Samwise", "Hello Samwise."));
47     kSession.insert(new Message("Peregrin", "Samwise", "Hello Samwise."));
48     kSession.insert(new Message("Peregrin", "Meriadoc", "Hello Meriadoc."));
49     kSession.insert(new Message("Frodo", "Meriadoc", "Hello Meriadoc."));
50     kSession.insert(new Message("Groot", "Rocket", "I am Groot!"));
51     kSession.insert(new Message("Gandalf the Grey", "Gandalf the White",
52                   "Am I talking to myself?"));
53     kSession.fireAllRules();
54     kSession.dispose();
55   }
56 }
```

Definieren Sie eine Drools-Regel, die eine Nachricht annimmt, sofern sie zwischen zwei dem System bekannten Personen geschickt wird (Namensgleichheit). Die Regel soll dann Sender und Empfänger sowie den Text der Nachricht ausgeben, und eine neue Antwortnachricht erzeugen.

Das Ergebnis der Regelauswertung soll eine unendliche Folge von Ausgaben folgender Art sein:

```
Frodo -> Meriadoc: Hello Meriadoc.
Frodo -> Samwise: Hello Samwise.
Meriadoc -> Peregrin: Hello Peregrin.
Samwise -> Peregrin: Hello Peregrin.
Meriadoc -> Frodo: Hello Frodo.
Samwise -> Frodo: Hello Frodo.
Peregrin -> Meriadoc: Hello Meriadoc.
Peregrin -> Samwise: Hello Samwise.
Frodo -> Meriadoc: Hello Meriadoc.
Frodo -> Samwise: Hello Samwise.
...
```

Lösungshinweise 3:

```
1  rule ”Reply”
2      when
3        $sender : Person()
4        $receiver : Person()
5          $message : Message(sender == $sender.name,
6                             receiver == $receiver.name)
7      then
8          System.out.println($sender.getName() + ” -> ” +
9                             $receiver.getName() + ”: ” +
10                            $message.getText());
11         insert(new Message($receiver.getName(),
12                            $sender.getName(),
13                            ”Hello ” + $sender.getName() + ”.”));
14 end
```