Institut für Informatik
der Universität München
Prof. M. Hofmann
A. Abel

WS 2005/2006

1. Dezember 2005

Übungen zur Vorlesung
Rechnergestütztes Beweisen

Blatt 7

Aufgabe 17: [Papier, 5 Punkte] Betrachten Sie die Theorie der Paare und Projektionen. In der Sprache $\mathcal{L}$ gebes einen Grundtyp β. Für je zwei Typen τ_1 und τ_2 gebe es den Produkttyp $\tau_1 * \tau_2$ und 3 Funktionssymbole: Paarbildung $\langle \_, \_ \rangle : [\tau_1, \tau_2 \to \tau_1 {*} \tau_2]$ (Mixfix-Schreibweise), erste Projektion $.1 : [\tau_1 \times \tau_2 \to \tau_1]$ und zweite Projektion $.2 : [\tau_1 \times \tau_2 \to \tau_2]$ (Postfix-Schreibweise). Die Gleichungstheorie wird erzeugt von den Axiomen

$$\langle x, y \rangle.1 = x \qquad \langle x, y \rangle.2 = y \qquad \langle x.1, x.2 \rangle = x\ .$$

Im Folgenden bezeichne T_τ die Terme vom Typ τ und N_τ die *neutralen* Terme vom Typ τ, das sind Folgen von Projektionen angewandt auf eine Variable. Die Theorie hat das folgende Modell: $[\![\beta]\!] = N_\beta$, $[\![\tau_1 * \tau_2]\!] = [\![\tau_1]\!] \times [\![\tau_2]\!]$ (kartesisches Produkt), $[\![\langle \_, \_ \rangle]\!](u, v) = (u, v)$, $[\![.1]\!](u, v) = u$, $[\![.2]\!](u, v) = v$. Nun definieren wir:

$$\begin{array}{rcl} q_\tau & : & [\![\tau]\!] \to T_\tau \\ q_\beta(n) & = & n \\ q_{\tau_1 * \tau_2}(u, v) & = & \langle q_{\tau_1}(u), q_{\tau_2}(v) \rangle \end{array}$$

$$\begin{array}{rcl} u_\tau & : & N_\tau \to [\![\tau]\!] \\ u_\beta(n) & = & n \\ u_{\tau_1 * \tau_2}(n) & = & (u_{\tau_1}(n.1), u_{\tau_2}(n.2)) \end{array}$$

Die Identitätsbelegung ρ_0 sei definiert durch $\rho_0(x) = u_\tau(x)$ für jede Variable $x : \tau$. Weisen Sie nach, dass

$$\begin{array}{rcl} \sigma_\tau & : & T_\tau \to T_\tau \\ \sigma_\tau(t) & = & q_\tau([\![t]\!]_{\rho_0}) \end{array}$$

ein Kanonisierer ist und berechnen Sie $\sigma_{\tau_1 * \tau_2}(\langle x, y \rangle.1)$.

Aufgabe 18: (Merge-Sort) [PVS, 15 Punkte] Zeigen Sie die Korrektheit von Merge-Sort. Von den drei beteiligten Funktionen `split`, `merge` und `sort` ist nur die erste primitiv rekursiv, d.h. der rekursive Aufruf erfolgt auf der unmittelbaren Vorgänger-Liste `cdr(l)`. Deswegen sind nur die Beweise über split durch gewöhnliche Listen-Induktion bequem machbar. Für merge und sort definiert man sich zuerst passende Induktionsschemata merge_induct und sort_induct, die den Rekursionsschemata der jeweiligen Funktionen entsprechen. Dabei ist das erste Schema durch eine doppelte Induktion, und das zweite Schema durch Maß-Induktion (measure-induct+ ...) zu beweisen. Eine Aussage über merge beweist man dann bequem mit merge_induct , und analog für sort .

Sortierung spezifiziert man primitiv rekursiv mit Hilfe der Hilfsrelation lb. Dabei gelte lb(x,l), falls das Element x kleiner gleich jedem Element der Liste l ist. Für sortiertes l ist dieser Test entsprechend einfach zu implementieren.

Führen Sie die Beweise untenstehender Aussagen in PVS und denken Sie auch an die TCCs.

```
mergesort : THEORY
  BEGIN

    t: TYPE
    x,y,z : VAR t
    l,l1,l2,xs,ys,zs : VAR list[t]

    % cases on lists
    list_case : LEMMA
        FORALL (p:[list[t] -> boolean]): FORALL (l):
          (l=null IMPLIES p(null)) AND
          (FORALL(x,xs):l=cons(x,xs) IMPLIES p(cons(x,xs))) IMPLIES
                p(l)

    % less than

    lt : [t,t->bool]
    lt_ax : AXIOM
      FORALL (x,y:t): NOT (lt(x,y) AND lt(y,x))

    % splitting

    split(l:list[t],flag:boolean) : RECURSIVE [list[t],list[t]] =
       IF null?(l)
         THEN (null,null)
         ELSE LET (u,v) = split(cdr(l),NOT flag) IN
           IF flag THEN (cons(car(l),u),v)
             ELSE (u,cons(car(l),v))
           ENDIF
      ENDIF
```

```
MEASURE length(l)

split1 : LEMMA
  FORALL(l:list[t], flag:boolean):
    LET (u,v) = split(l,flag) IN
      length(u) <= length(l) AND
      length(v) <= length(l) AND
      (NOT null?(l) AND NOT null?(cdr(l)) IMPLIES
        length(u) < length(l) AND
        length(v) < length(l))

% merging

merge_induct : LEMMA
  FORALL (p : [list[t],list[t] -> boolean]):
    (FORALL (l2): p(null,l2)) AND
    (FORALL (l1): p(l1,null)) AND
    (FORALL (x,y,xs,ys): p(xs,cons(y,ys)) AND p(cons(x,xs),ys)
      IMPLIES p(cons(x,xs),cons(y,ys)))
    IMPLIES (FORALL (l1,l2): p(l1,l2))
% induction on l1 and l2

merge(u:list[t], v:list[t]) : RECURSIVE list[t] =
  IF null?(u) THEN v
  ELSIF null?(v) THEN u
  ELSIF lt(car(u),car(v))
    THEN cons(car(u),merge(cdr(u),v))
    ELSE cons(car(v),merge(u,cdr(v)))
  ENDIF
MEASURE length(u)+length(v)

% merge sort

sort_induct : LEMMA
  FORALL (p : [list[t] -> boolean]):
    p(null) AND
    (FORALL (x): p (cons(x,null))) AND
    (FORALL (x,y,l): p(split(cons(x,cons(y,l)),TRUE)`1) AND
                     p(split(cons(x,cons(y,l)),TRUE)`2) IMPLIES
                     p(cons(x,cons(y,l))))
    IMPLIES FORALL (l): p(l)
% measure-induct

sort(l:list[t]) : RECURSIVE list[t] =
```

```
  IF null?(l)
    THEN null
  ELSIF null?(cdr(l))
    THEN l
    ELSE LET (u,v) = split(l,TRUE) IN
      merge(sort(u),sort(v))
  ENDIF
MEASURE length(l)

% occurrences and permutation

occ(x:t,l:list[t]) : RECURSIVE nat =
  CASES l
    OF null : 0 ,
    cons(y,l) : IF x=y THEN occ(x,l)+1 ELSE occ(x,l) ENDIF
  ENDCASES
MEASURE length(l)

perm(l1,l2 : list[t]) : bool = FORALL(x:t):occ(x,l1) = occ(x,l2)

split2 : LEMMA
   FORALL(l:list[t],flag:boolean):
     LET (u,v) = split(l,flag) IN
       FORALL(x:t):occ(x,l) = occ(x,u) + occ(x,v)

merge1 : LEMMA
   FORALL(u,v:list[t]):
       FORALL(x:t):occ(x,merge(u,v)) = occ(x,u)+occ(x,v)
% use merge_induct

% lower bound of a sorted list

lb (x:t, l:list[t]) : boolean =
  null?(l) OR NOT lt(car(l),x)

% "sorted" in terms of "lower bound"

sorted(l:list[t]) : RECURSIVE boolean =
  IF null?(l)
     THEN TRUE
     ELSE sorted(cdr(l)) AND lb(car(l),cdr(l))
  ENDIF
MEASURE length(l)
```

```
  % merge preserves lower bounds

  lb_merge : LEMMA
      FORALL (x): FORALL (l1,l2):
      lb(x,l1) AND lb(x,l2) IMPLIES lb(x,merge(l1,l2))
  % use merge_induct

  % merge preserves sortation

  merge2 : LEMMA
     FORALL(u,v:list[t]):
         sorted(u) AND sorted(v) IMPLIES sorted(merge(u,v))
  % use merge_induct and lb_merge

  sort_correct : THEOREM
       FORALL(l:list[t]):
          sorted(sort(l)) AND perm(l,sort(l))
  % use sort_induct, merge1 and merge2

END mergesort
```

Abgabe: bis **Dienstag, 06.12., 18.00 Uhr** im Sekretariat (**Z1.05**) oder bei Andreas Abel im Büro (**D1.09**). Elektronische Lösungsteile per Mail an `abel@informatik.uni-muenchen.de`.