Übungsaufgaben zum SML-Kurs, WS 2000/2001

Hans Leiß

Abgabe: Di, 21.11.2000

Wir wollen die Definition des Datentyps der aussagenlogischen Formeln so ändern, daß Konjunktion und Disjunktion nicht ein *Paar* von Formeln verbinden, sondern eine beliebig lange *Liste* von Formeln.

⟨*prop*⟩≡
```
datatype 'a prop =
    Atom of 'a
  | Neg of 'a prop
  | Disj of 'a prop list
  | Conj of 'a prop list
```

Jetzt können wir eine Konjunktion über viele Formeln einfacher durch `Conj [A,B,C,D,E,F]` angeben. (Noch besser wäre es, wenn wir auch die Idempotenz und Kommutativität der Disjunktion berücksichtigen würden, d.h. eine *Menge* als Argument angeben würden. Das hätte aber den Nachteil, daß man zuerst Mengen definieren müßte und dann Probleme bekommt, wenn man Teilformeln einer Disjunktion finden oder in bestimmter Reihenfolge bearbeiten wollte.)

Die Negationsnormalform `nnf phi` einer Formel muß an diese Definition angepasst werden. Das hatten wir in der Übungsstunde getan:

⟨*nnf*⟩≡
```
fun nnf (Atom a) = Atom a
  | nnf (Conj phis) = Conj (map nnf phis)
  | nnf (Disj phis) = Disj (map nnf phis)
  | nnf (Neg (Atom a)) = Neg (Atom a)
  | nnf (Neg (Conj phis)) = Disj (map (nnf o Neg) phis)
  | nnf (Neg (Disj phis)) = Conj (map (nnf o Neg) phis)
  | nnf (Neg (Neg phi)) = nnf phi
```

Aufgabe 1.3 Man passe die Definition der Auswertung `eval` an die geänderte Definition der Formeln an. Dazu gibt es zwei Möglichkeiten:

(i) Wir definieren die Auswertung einer Konjunktion durch Rekursion über die Anzahl der Konjunktionsglieder. Entsprechendes gilt für die Disjunktion. Dann brauchen wir jeweils zwei Fälle:

⟨*eval*⟩≡
```
fun eval h (Conj []) = ...
  | eval h (Conj (phi::phis)) = ...
  | eval h (Disj []) = ...
  | eval h (Disj (phi::phis)) = ...
```

Für die Auswertung der anderen Formeln ändert sich nichts. Man führe dies aus. (4 Punkte)

(ii) Wir definieren die Auswertung einer Konjunktion `(Conj phis)`, indem wir alle Konjunktionsglieder aus der Liste `phis` auswerten und die Ergebnisse, eine Liste `bs` von Boole'schen Werten, mit dem boole'schen „und" miteinander „multiplizieren", d.h.

$$b_1 \wedge b_2 \wedge \cdots \wedge b_n \;=\; b_1 \wedge (b_2 \wedge \cdots \wedge (b_n \wedge \; \texttt{true}) \cdots)$$

bilden. Hierzu bietet sich die Faltung `foldr` (siehe Folien) an. Man muß also zwei Dinge tun und miteinander passend verbinden:

(a) aus der Liste `phis` die Liste `bs` der Auswertungen der `phis` bzgl. `h` bilden,

(b) die „und"-Funktion `(fn (b,b') => (b andalso b'))` mit `foldr` über die Liste der erhaltenen `bs` falten. Man führe das aus. (4 Punkte)

Falls man das schfft, kann man die analoge Auswertung für Disjunktionen `Disj phis` versuchen.

Aufgabe 1.4 Programmiere eine Substitution von Formeln für atomare Formeln.

(i) Das kann man z.B. so machen, daß man eine Zuordnung `s : 'a -> 'a prop` (zur Behandlung der atomaren Formeln) annimmt und dann eine Funktion `subst : ('a -> 'a prop) -> ('a prop) -> ('a prop)` schreibt, also (4 Punkte)

$\langle subst \rangle \equiv$
```
fun subst s (Atom a) = ...
  | subst s (Neg phi) = ...
```

(ii) Oder man gibt statt einer Funktion `s : 'a -> 'a prop` eine Liste `s : ('a * ('a prop)) list` vor und ersetzt nur die atomaren Formeln `Atom a`, für die man ein Paar `(a,phia)` in der Liste findet. Dazu muß man die Liste durchsuchen, was wir besser erst mal in der Übungsstunde machen.