Übung zur Vorlesung

Automatisierte Logik und Programmierung II

Prof. Chr. Kreitz

Universität Potsdam, Theoretische Informatik — Sommersemester 2009

Blatt 1 — Abgabetermin: 15.05.2009

Aufgabe 1.1 (ML Spielereien)

Programmieren Sie die folgenden elementaren ML Funktionen

1.1–a `member: * -> * list -> bool`
testet, ob ein Element `x` in einer Liste `l` vorkommt.

1.1–b `select: int -> * list -> *`
bestimmt das `n`-te Element einer Liste `l`.

1.1–c `replicate: * -> int -> * list`
erzeugt bei Eingabe von `x` und `n` die `n`-elementige Liste `[x;...x]`

1.1–d `find: (*-> bool) -> * list -> *`
findet bei Eingabe einer Funktion `f` und einer Liste `l` das erste Element `x` von `l`, für das `f(x)=true` ist.

1.1–e `map: (*->**) -> * list -> ** list`
wendet eine Funktion `f` auf alle Elemente einer Liste `l` an.

1.1–f `lookup: (*#**) list -> * -> **`
findet bei Eingabe eines Labels `x` das erste Element `y`, für das ⟨`x,y`⟩ in einer Liste `l` liegt.

Aufgabe 1.2

In Einheit 9 hatten wir die Curry-Howard Isomorphie zwischen den logischen Regeln und den Regeln der Typentheorie besprochen und die Logikoperatoren als definitorische Erweiterung der Sprache der Typentheorie eingeführt. Wir wollen nun auch das Inferenzsystem der Typentheorie um logische Inferenzregeln erweitern.

1.2–a Überlegen Sie zunächst, welche Schritte notwendig sind, um die Inferenzregeln der Typentheorie in Taktiken umzuwandeln, welche die logischen Inferenzregeln nachbilden.

Sie dürfen davon ausgehen, daß Basistaktiken wie `D 0` die logischen Definitionen wie `and`, `or`, `implies` etc. automatisch auffalten und dann das entsprechende typentheoretische Konstrukt (Produkt, Disjunkte Vereinigung, Funktionenraum, ...) zerlegen. Sie müssen aber verhindern, daß eine Regel wie `andR` auch auf eine Implikation anwendbar ist.

Da 15 Inferenzregeln zu programmieren sind, lohnt sich eine Higher-Order Konzeption. Die folgenden Teilaufgaben beschreiben eine mögliche Vorgehensweise

1.2–b Programmieren Sie ein Tactical
`TryOn: ((int -> tactic) -> int -> (term -> bool) -> tactic`,
das bei Eingabe einer (parametrisierten) Taktik tac, einer Klauselummer i (0 steht für die Konklusion), und einer Testfunktion $test$ auf Termen zunächst überprüft, ob der Term in Klausel i den Test erfüllt, im Erfolgsfall die Taktik tac auf Klausel i anwendet und im Mißerfolgsfall mit Fehlermeldung abbricht.

Die übliche Art der Definition einer solchen wäre `let TryOn tac i test p = ...`, wobei `p` für das aktuelle Beweisobjekt steht, das Sie ggf. mit Funktionen wie `conclusion` oder `hypotheses` analysieren müssen. Für die Erzeugung einer Fehlermeldung können Sie die Taktik `Fail` verwenden. Achten Sie darauf, daß `TryOn tac i test` den Typ `tactic` hat.

1.2–c Programmieren Sie beispielhaft die Testfunktion
`is_and_term: term -> bool`,
welche bei Eingabe eines Terms t prüft, ob es sich um eine Konjunktion handelt.

1.2–d Implementieren Sie mit Hilfe von `TryOn`, der Taktik `D`, den Funktionen zum Einfügen von Steuerungsparametern und Testfunktionen wie der obigen die Regeln der Prädikatenlogik als Taktiken. Programmieren Sie insbesondere die Regeln `andL`, `orR1`, exR und `allL`.

Um eventuell entstandene Wohlgeformtheitsziele zu bearbeiten, können Sie nach Ausführung der Basistaktik das Konstrukt `THENW Auto` verwenden. Das Tactical `THENW` arbeitet ähnlich wie `THEN`, wendet die zweite Taktik aber nur auf die verbleibenden Wohlgeformtheitsziele der ersten an. Die Taktik `D` markiert derartige Ziele.

Aufgabe 1.3

In Einheit 12 haben wir die Tacticals `Try`, `Progress` und `Repeat` beschrieben. Geben Sie eine ML-Implementierung dieser Tacticals an.