

Konzepte höherer Programmiersprachen Sommersemester 2005

Übungsblatt 4

Aufgabe 1

In Übung 3, Aufgabe 3 wurde eine Funktion `beta_reductions` implementiert, die zu einem Ausdruck e alle möglichen β -Reduktionen liefert. Implementieren Sie jetzt eine Funktion `beta_value`, die nur die *eine* β -Reduktion liefert, die sich nach der call-by-value-von-links-nach-rechts-Strategie ergibt. Diese Strategie sieht so aus:

1. Eine λ -Abstraktion $\lambda id.e$ wird nie reduziert.
2. Bei einer Applikation $e_1 e_2$ wird zuerst e_1 und dann e_2 ausgewertet.
3. Die β -Regel

$$(\lambda id. e_1) e_2 \rightarrow e_1[e_2/id]$$

wird erst dann angewandt, wenn e_2 schon vollständig ausgewertet ist.

Testen Sie Ihre Funktion mit den Ausdrücken aus Übung 3, Aufgabe 2.

Aufgabe 2

Überprüfen Sie die Wohlgetyptheit der folgenden Ausdrücke mit Hilfe der Typregeln für den einfach getypten λ -Kalkül.

- a. `let $x = 1$ in $(\lambda x : \mathbf{int}. + x 1) (+ x 1)$`
- b. `let $x = 1$ in let $f = \lambda y : \mathbf{int}. + x y$ in $f (f x)$`
- c. `let $x = 1$ in let rec $f : \mathbf{int} \rightarrow \mathbf{int} = \lambda x : \mathbf{int}. f (+ x 1)$ in $f x$`

In Aufgabe 3 soll der Typüberprüfungs-Algorithmus für den einfach getypten λ -Kalkül implementiert werden. Dazu sei die folgende abstrakte Syntax für Typen und Ausdrücke vorgegeben. (Diesen Quellcode finden Sie in www.informatik.uni-siegen.de/~sieber/public/2005_SS_KP/0405.ml)

```
type base_type = UNIT | BOOL | INT | FLOAT

type simple_type =
  BASE of base_type
  | ARROW of simple_type * simple_type
  | PRODUCT of simple_type list

type identifier = string

type operator =
  Not | IAdd | ISub | IMul | IDiv
      | FAdd | FSub | FMul | FDiv

type constant =
  Unit
  | Bool of bool
  | Int of int
  | Float of float
  | Op of operator

type expression =
  Const of constant
  | Id of identifier
  | App of expression * expression
  | If of expression * expression * expression
  | Tuple of expression list
  | Lambda of identifier * simple_type * expression
  | Let of identifier * expression * expression
  | Rec of identifier * simple_type * expression
```

Aufgabe 3

Implementieren Sie den Typüberprüfungs-Algorithmus, der sich aus den in der Vorlesung angegebenen Typregeln ergibt. Definieren Sie dazu im einzelnen:

- a. Funktionen `op_type` und `const_type`, die den Operatoren und Konstanten ihre üblichen Typen zuordnen
- b. einen Typ `type_env` zur Darstellung von Typumgebungen,
- c. eine exception `Type_error` zur Signalisierung von Typfehlern,
- d. eine Funktion `lookup`, die zu einer Typumgebung Γ und einem Namen id den Typ $\Gamma(id)$ liefert oder die exception `Type_error` wirft,
- e. eine Funktion `update`, die zu Γ , τ und id die veränderte Typumgebung $\Gamma[\tau/id]$ liefert.
- f. eine Funktion `exp_type`, die zu Γ und e den (eindeutigen) Typ τ mit $\Gamma \triangleright e :: \tau$ liefert oder die exception `Type_error` wirft.