Hochschule München
Fakultät für Informatik und Mathematik
M. Ruckert

Theoretische Informatik II
Übungen

Aufgabe 24. Implementieren Sie `lambda`-Funktionen.
Um in dem vorhandenen Interpreter auch den folgenden Code

```
(define double (lambda (x) (+ x x)))
(double 3)
```

korrekt interpretieren zu können braucht es zweierlei:

- Neben den eingebauten Funktionen für `define` und `+` braucht man auch eine eingebaute Funktion `lambda`. Letztere wird von dem ersten `define` aufgerufen wenn es den Ausdruck `(lambda (x) (+ x x))` evaluiert.

  Scheme (im Gegensatz zum reinen λ-Kalkül) evaluiert einen `lambda` Ausdruck erst zum Zeitpunkt der Anwendung auf die Argumente, nicht schon bei der Definition. Daher ist die Implementierung der eingebauten Funktion sehr einfach. Die Funktionsdefinition wird einfach im Speicher abgelegt.

  Wir implementieren die folgenden Funktionen:

  - `(new-lambda arglist body env)`
    Diese Funktion (Konstruktor) erzeugt eine neue lambda-Funktion im simulierten Speicher, indem sie dort nach dem Tag `'lambda`, die Argumentliste, den Funktions Body und die Umgebung in der die Funktion definiert wurde (das sind alles Zeiger in's `memory`) ablegt. Dannach brauchen wir noch Getter Funktionen:
  - `(lambda->arglist p)`
  - `(lambda->body p)`
  - `(lambda->env p)`
    die aus einer lambda-Funktion die einzelnen Bestandteile extrahieren.
  - `(i-lambda env values)` Die Funktion, die als eingebaute Funktion an das Symbol `lambda` gebunden wird und die im wesentlichen `new-lambda` aufruft.
  - `i-expr->expr` Diese Funktion muss erweitert werden, so dass auch mit lambda definierte Funktionen wieder ausgegeben werden können.

- Um den zweiten Teil des obigen Codes `(double 3)` ausführen zu können muss die Funktion `i-apply` erweitert werden. Außer den primitiven Funktionen muss sie auch `lambda` definierte Funktionen behandeln können.

  Die Anwendung einer solchen Funktion erfolgt in zwei Schritten. Zuerst wird eine neue Umgebung erzeugt; diese wird zunächst mir den Bindungen aus der **Definitionsumgebung** initialisiert. Die **Definitionsumgebung** wurde zusammen der Funktion gespeichert (closure). Dann wird die neue Umgebung durch Hinzufügen von Bindungen für die Variablen in der Parameterliste erweitert. Dazu werden alle Argumente der Funktion in der **Aufrufumgebung** evaluiert und die Werte an die entsprechenden Variablen in der Parameterliste gebunden. Danach kann dann der Funktions-Body in dieser neuen Umgebung evaluiert werden (mit `i-eval`).

  Es ist nützlich dazu die folgende Hilfsfunktionen zu implementieren:

  - `(add-parameters param-list expr-list call-env env)` iteriert über `param-list` und `expr-list`. Für jede Variable aus `param-list` gibt es einen zugehörigen Expression in `expr-list`. Dieser wird mittels `i-eval` in der Aufrufumgebung `call-env` evaluiert und der erhaltene Wert wird zusammen mit der zugehörigen Variablen mittels `add-variable` der Umgebung `env` hinzugefügt.
  - `(new-lambda-environment param-list expr-list call-env def-env)` Erzeugt eine neue Umgebung, initialisiert diese mit der Definitionsumgebung, und benutzt dann `add-parameters` um die Funktionsargumente zu evaluieren und der Umgebung die Bindungen zwischen den Parametervariablen und diesen Werten hinzuzufügen.

  Mit `new-lambda-environment` lässt sich die Erweiterung von `i-apply` dann einfach implementieren.