Hochschule München
Fakultät für Informatik und Mathematik
M. Ruckert

Theoretische Informatik II
Übungen

Aufgabe 20. Wir implementieren das memory management für den Scheme Interpreter. Dabei orientieren wir uns an den Möglichkeiten, die dafür etwa 1958 (FORTRAN) zur Verfügung standen. Heute (Java) könnte man dafür ein Objektmodell wählen oder (C) sehr effizient mit Adressen und einzelnen Bits spielen.

Das memory management hat drei Schichten:

- Low level allocation.

  Hier bekommt man mit der Funktion `(malloc n)` einen Zeiger auf n Speicherzellen. Da wir in FORTRAN keine Zeiger haben simulieren wir das ganze durch einen grossen Vektor `memory` und Zeiger sind einfach Indices in diesen Vektor. Die Speicherwerwaltung halten wir einfach indem wir auf garbage collection und Freigabe von Speicher verzichten. Damit kann die Speicherverwaltung durch einen Zeiger (Index) `free` realisiert werden, der den freien oberen Teil des Speichers vom belegten unteren Teil trennt. Für das Debugging (und für eine mögliche Freigabe von Speicher) ist es bequem, wenn vor jedem allokierten Block seine Größe eintragen wird.

- Record access.

  Auf der nächsten Ebene betrachten wir Daten als Records. Das heist, ein Datum besteht aus mehreren aufeinanderfolgenden Speicherzellen, die über eine Basisadresse und einen Offset angesprochen werden. Ein solcher Record mit n Zellen wird mit `(malloc n)` angelegt und mit den Funktionen `(ref p i)` und `(ref! p i value)` mit i zwischen 0 und n-1, gelesen und geschrieben.

- Scheme records.

  Eine besondere Funktion hat das erste Feld eines Records, das Tag. Es kennzeichnet die Art des Records und hat Werte wie `'undefined`, `'true`, `'false`, `'number`, `'symbol`, `'pair`, `'lambda` und andere.

  Einige Objekte wie `true`, `false`, `null`, oder `undefined` bestehen nur aus dem Tag und man kann sie direkt im Speicher anlegen. Um den Zeiger unseres Interpreters auf das Tag `'true` vom Tag selbst und vom normalen `#t` zu unterscheiden bekommt er den Namen `i-true`. Ähnliches gilt für die anderen Konstanten und später werden wir auch die `car` Funktion unseres Interpreters `i-car` nennen.

  Für andere Objekte, die der Scheme Interpreter braucht, wie Zahlen und Symbole, definiere man geeignte Konstruktoren und Getter wie zum Beispiel: `(tag p)`, `(i-symbol name)`, `(symbol->name p)`, `(i-number value)`, `(number->value p)`, `(i-cons h t)`, `(i-car p)`, `(i-cdr p)`, etc.

- Spätestens an dieser Stelle sollten Sie sich zum Debugging Hilfsfunktionen schreiben, die Ihnen einen (lesbaren) Speicherdump erstellen.

  Weiter empfielt sich die Verwendung einer Funktion `error`, die Fehlermeldungen ausgibt und das Programm beendet. (Zum Beenden kann man zum Beispiel die undefinierte Funktion `exit` aufrufen.)