Limbo

Vertiefungsseminar 2012/2013
Betreuer: Thomas Sternagel

Markus Müller

23. Januar 2013

Inhalt

Einleitung

Was ist Limbo?

- C-ähnliche Programmiersprache
- in den Bell Labs 1995 entwickelt
- Operating System: Inferno
- Entwicklung von verteilten Systemen

Verwendung

- architekturunabhängig
- nur unter Inferno verwendbar
- Compiler produziert Objektcode
  - von virtuellen Maschine „Dis“ interpretiert
  - just-in-time in Maschinencode kompiliert

Aufbau

- modularer Aufbau
- *.m und *.b Dateien
- nur eine Modulinstanz im Speicher

Syntax

- Nachfahre von C
- Deklaration und Initalisierung ähnelt Ada
- Operatoren werden wie in funktionalen Sprachen angewandt

Datentypen

Limbo bietet neben bekannten Typen wie `byte` und `int` weitere, meist ähnlich zu bereits bekannten Typen:

- `big`, 64-bit unsigned, vergleichbar mit `long` in C
- `real`, 64-bit floating point, vergleichbar mit `double`
- `list`, Typ, der es erlaubt Listen zu deklarieren, Listen ähnlich wie in Ocaml oder Haskell (hd::tl)
- `array`, Struktur zur dynamischen Verwaltung von Elementen desselben Typs (in C Typen mit `[]` versehen)
- `string`, Typ zur Verwaltung von Zeichenketten, in C nur durch `char`-Array möglich

Datentypen cont'd

- `tuple`, geordnete Sammlung von mindestens 2 Elementen, auch verschiedenen Typs
- `channel`, für Inter-Prozess Kommunikation, vergleichbar mit Pipes in C
- `adt` (abstract data type), enthält Datenobjekte und Funktionen welche damit arbeiten
- `module`, Limbos Version von Bibliotheken inkl. Implementierung

Deklaration und Zuweisung

- ```
  p: Point;
  x, y: int;
  (x, y) = p;
  ```
- ähnelt Ada auf Grund der Rechtsassoziativität
- automatische Aufteilung der Objekte aus `p` auf `x` und `y`
- auch `x := 1;` möglich
  ```

Keywords

Limbo besitzt wie jede andere Sprache eine Reihe von reservierten Wörtern:

- alle Datentypen
- `return, continue, break, exit` etc.
- `hd` und `tl`: Operatoren, gibt erstes Element einer Liste bzw. die restliche Liste zurück
- `nil`: Konstante, vergleichbar mit `NULL` in C
- `len`: Operator, gibt Länge einer Liste oder Arrays zurück
- `self`: vergleichbar mit der `this`-Referenz in Java, wird in `adt`-Objekten verwendet
- `cyclic`: zur Realisierung von zyklischen Datenstrukturen
- `fn`: zur Deklaration von Funktionen

99 Bottles of Beer

```
1  implement BeerBottles;
2
3  include "sys.m";
4    sys: Sys;
5  include "draw.m";
6    draw: Draw;
7
8  BeerBottles: module{
9    init: fn(ctxt: ref Draw->Context, argv: list of string);
10 };
11 init(ctxt: ref Draw->Context, argv: list of string){
12   sys = load Sys Sys->PATH;
13   for (int b = 99; b > 0; b--) {
14       sys->print("%d bottle(s) of beer on the wall,\n", b);
15       sys->print("%d bottle(s) of beer.\n", b);
16       sys->print("Take one down, pass it around,\n");
17       sys->print("%d bottle(s) of beer on the wall.\n\n", b-1);
18   }
19 }
```

Features

- Modular Programming
- Concurrency
- Type Checking
- Garbage Collection
- Abstract Data Types (ADT)

Nebenläufigkeit

- Threads mit `spawn` erzeugen: `spawn run(arg);`
- Channel zur Synchronisation als auch Kommunikation
  - kann Typ besitzen, `chan of int;`
  - unterstützt auch mehrere Typen, `chan of (int, string);`

Beispiel

```
1  init(nil: ref Draw->Context, argv: list of string){
2   sys = load Sys Sys->PATH;
3
4   sync := chan of int;
5   n := len argv;
6   spawn timer(sync, n);
7
8   sys->print("Command Line Parameters\n");
9   for (i := 0; i < n; i++) {
10    <-sync;
11    sys->print("%d: %s\n", i, hd argv);
12    argv = tl argv;
13  }
14 }
15
16 timer(sync: chan of int, n: int){
17  for (i := 0; i < n; i++) {
18    sys->sleep(1000);
19    sync <-= 1;
20  }
21 }
```

Anwendungsbereiche

- wenig bis keine bekannten Projekte
- gitfs
  - Tool für Git scm
  - Fileserver Interface

Zusammenfassung

- Programmiersprache für Inferno OS
- viele Ähnlichkeiten zu C, Ada und anderen Sprachen
- Entwicklung von verteilten Systemen
- einfache Möglichkeiten zur Nebenläufigkeit
- heutzutage kaum Anwendungsgebiete