
Informatik II

Blatt 5

Abgabe: 15.6.2000

1. In der Vorlesung kam folgender Code für nebenläufige Prozesse vor:

```
(define *coroutines* '())

(define (add-coroutine! thunk)
  (set! *coroutines* (append *coroutines* (list thunk))))

(define (start)
  (let ((next (car *coroutines*)))
    (set! *coroutines* (cdr *coroutines*))
    (next)))

(define (pause)
  (call-with-current-continuation
   (lambda (k)
     (add-coroutine! (lambda () (k #f)))
     (start))))
```

Jeder Prozeß wird durch eine Prozedur ohne Parameter (also einen Thunk) repräsentiert, die, wenn aufgerufen, endlos läuft. Ein Prozeß wird beim System angemeldet durch Aufruf von `add-coroutine!` mit dem zugehörigen Thunk als Parameter. Die Prozesse werden durch Aufruf von `start` gestartet. Jeder Prozeß muß ab und zu `pause` aufrufen, damit auch andere Prozesse Rechenzeit abbekommen.

- (a) [4 Punkte] Entwickle ein kleines Beispiel für die Benutzung nebenläufiger Prozesse. Vorschlag: Starte einen Prozeß, der Primzahlen ausrechnet. Starte für jede neue Primzahl einen Prozeß, welcher die Potenzen dieser Primzahlen ausdrückt.
- (b) [7 Punkte] Entsteht ein Problem, wenn ein Prozeß terminiert, also sein Thunk zurückkehrt, ohne `pause` aufzurufen? Falls ja, gib ein Beispiel an, welches demonstriert, daß es ein Problem gibt. Schreibe eine Prozedur `quit` (ohne Parameter), welche von einem Prozeß aufgerufen werden kann, und diesen so beendet, daß die anderen Prozesse normal weiterlaufen können.
- (c) [4 Punkte] Ändere das Verfahren, einen neuen Prozeß zu erzeugen, dahingehend, daß ein Prozeß auch einfach (also ohne vorherigen Aufruf von `quit`) terminieren kann, und die anderen Prozesse dann regulär weiterlaufen.

2. Ändere den CPS-Interpreter mit Registern und Continuation-Frames so, daß der Stack aus Frames sichtbar wird.

- (a) [2 Punkte] Implementiere zunächst direkt mit Hilfe der Typ-Abstraktionen aus der Vorlesung einen ADT für Stacks mit Konstruktoren `make-stack` und `push`, Prädikaten `stack?`, `empty-stack?` und Selektoren `top` und `pop`. `Pop` und `top` sollen Fehlermeldungen auslösen, falls der Stack leer ist.

```
> (define s1 (make-stack))
> (define s2 (push s1 23))
> (top s2)
23
> (stack? s2)
#t
> (stack? '(1 2))
#f
> (empty-stack? s2)
#f
> (empty-stack? s1)
#t
> (define s3 (push s2 42))
> (top s3)
42
> (top (pop s3))
23
```

- (b) [1 Punkt] Erläutere, warum die Continuation-Frames einen Stack bilden. Zeichne ein geeignetes Bild, um diesen Umstand zu verdeutlichen.
- (c) [5 Punkte] Ersetze im Interpreter das `*k*`-Register durch zwei Register `*cont-frame*` und `*stack*`. Dabei enthält `*stack*` den Stapel von Continuation-Frames, aus denen die momentane Continuation besteht. `*Cont-frame*` nimmt jeweils das oberste Frame des Stacks auf. Schreibe Prozeduren `push-stack!` und `pop-stack!`, welche den Stack in diesem Stack-Register verwalten.

```
> (push-stack! 23)
> (push-stack! 42)
> (pop-stack!)
42
> (pop-stack!)
23
```

Entferne das `previous-frame`-Feld aus den Continuation-Frame. Wo im alten Interpreter das `*k*`-Register gesetzt wird, benutze `push-stack!` und `pop-stack!`. Ignoriere zunächst `let/cc`.

- (d) [1 Punkte] Zeichne ein Bild, was die neue Anordnung der Continuation-Frames im Stack zeigt und den Unterschied zur vorigen Repräsentation verdeutlicht.
- (e) [4 Punkte] Implementiere nun auch noch `let/cc`. Schreibe dazu Prozeduren `get-stack` und `set-stack!`, welche das `*stack*`-Register auslesen bzw. setzen. Ersetze dazu `*k*` im alten Interpreter durch entsprechende Aufrufe dieser Prozeduren.

```
> (push-stack! 23)
> (push-stack! 42)
> (define s (get-stack))
> (pop-stack!)
42
> (set-stack! s)
> (pop-stack!)
42
```

- (f) [3 Punkte] Implementiere eine imperative Version des Stack-ADTs, welcher einen Vektor für die Repräsentation eines Stacks benutzt. Der Konstruktor `make-stack` bekommt dazu einen zusätzlichen Parameter, welche die Maximalgröße des Stacks bestimmt. Dazu wird `push` durch eine Prozedur `push!` ersetzt, sowie `pop` durch `pop!`:

```
> (define s (make-stack 1000))
> (push! s 23)
> (push! s 42)
> (pop! s)
42
> (pop! s)
23
```

Ändere den Interpreter dahingehend, daß er diese Repräsentation für Stacks benutzt. Laß zunächst `let/cc` weg. Dies sollte lediglich Änderungen in den Abstraktionen über dem `*stack*`-Register erfordern.

- (g) (Bonus, 6 Punkte) Warum funktioniert der alte Code für `let/cc` nicht mehr mit den imperativen Stacks? Gib ein Beispiel an, welche den Defekt aufzeigt. Was muß getan werden, um `let/cc` zum Laufen zu bringen?