
P O S I T I O N S P A P I E R

in der Fachrichtung
Wirtschaftsinformatik

T H E M A

# Übertragung der Entwurfsvorschrift III auf Lua


| | |
|---|---|
| Eingereicht von: | Michael Bannas, Pascal Görgen, Marc Reineking, Timo Kluge, Matthias Sobek & Jan Werder |
| Zenturie: | I12a |
| Bearbeitungszeitraum: | 20.05.2014 - 09.06.2014 |

# Inhaltsverzeichnis

# 1 Einleitung

Als Semesterarbeit für die Vorlesung Abstraktionskonzepte der Informatik soll die Entwurfsvorschrift III in eine andere Programmiersprache übertragen werden. Die Autoren dieses Positionspapiers haben für diese Aufgabe die Sprache Lua gewählt.

## 1.1 Lua

Lua ist eine imperative und erweiterbare Skriptsprache zum Einbinden in Programme, um diese leichter weiterentwickeln und warten zu können. Lua ist freie Software und steht aktuell unter der MIT-Lizenz. Lua-Programme sind meist plattformunabhängig und werden vor der Ausführung in Bytecode übersetzt. Luas Typisierung ist schwach, implizit und dynamisch.

## 1.2 Entwurfsvorschrift III

Die Entwurfsvorschrift III fügt den bisherigen Entwurfsvorschriften die Rekursion hinzu. Damit ist es fortan möglich beliebig verschachtelte und große Datenstrukturen zu verarbeiten. Dafür ist im minimalen Fall ein Selbstaufruf und eine Terminierungsbedingung nötig.

# 2 Implementierung

Die Implementierung ist für die Semesterarbeit so allgemein gehalten wie möglich. Bei vielen Funktionen ist es möglich Zahlen und Symbole zu verketten und eine beliebige Anzahl an Parametern zu übergeben. Die Funktionsschablonen sind daher hier in diesem Dokument festgehalten und erklärt.

## 2.1 Funktion ’first’

Die Implemtierung der Funktion ’first’ ist in Lua sehr direkt umsetzbar. Der folgende Codeausschnitt beschreibt die Funktionsschablone.

```
function first(tbl)
  if preTable(tbl) then
    if #tbl > 0 then
      ...
    else
      ...
    end
  else
    ...
  end
end
```

Da ’first’ nur eine Eingabe nimmt ist in diesem Fall auch keine Rekursion nötig. Trotzdem muss zu Beginn überprüft werden, ob die Eingabe eine Liste ist. Falls dies nicht der Fall ist wird der entsprechende Fehler ausgelöst. Falls die Eingabe eine Liste ist, so kann das erste Element ausgegeben werden, außer die Liste enthält kein Element, in dem Fall sollte die leere Liste zurückgegeben werden.

## 2.2 Funktion ’rest’

Die Funktion ’rest’ soll die Liste ohne das erste Element als Aufgabe haben. Dies wurde in Lua wie folgt umgesetzt.

```
function rest(tbl)
  ...
  if preTable(t) then
      ...
  else
    ...
  end
end
```

Die Eingabe wird zu Beginn mit der Precondition überprüft und falls sie keine Liste ist, wird ein Fehler zurückgegeben. Falls die Eingabe eine valide Tabelle ist, so wird das erste Element entfernt. Lua übernimmt hierbei die Behandlung der leeren Liste.

## 2.3 Funktion ’cons’

Die Funktion ’cons’ kann in der Version der Autoren beliebige viele Parameter annehmen. Daher ist bei der unten angegeben Schablone zwischen ’{...}’ für die variablen Parameter und ’[...]’ für die reguläre Vereinfachung der Schablone zu differenzieren.

```
function cons(...)
  local args = {...}
  for k,v in ipairs(args) do
    if not preTable(v)then
      [...]
    end
   end
  [...]
end
```

Die Eingabe wird in unserer Implementation in eine lokale Variable geschrieben, was dazu führt, dass alle Parameter zu einer neuen Liste zusammengefügt wer-

den. Da das die genaue Funktionsweise von 'cons' aus Clojure beschreibt, muss nun nur noch jede Eingabeliste auf die Precondition überprüft werden. Hier wird davon ausgegangen, dass jede Eingabe eine Liste sein muss.

## 2.4 Funktion 'append'

Die Funktion 'append' ist ebenfalls in der Lage eine variable Anzahl an Parametern anzunehmen.

```
function append(...)
    local args = {...}
    [...]
    for k1,v1 in ipairs(args) do
      if preTable(v1)then
        for k2,v2 in ipairs(v1) do
          [...]
        end
      else
        [...]
      end
    end
    [...]
end
```

Zuerst werden alle Parameter zu einer Liste zusammengefügt. Danach wird für jede Element überprüft ob es Liste ist. Sollte dies der Fall sein, so wird jedes Element aus der Liste in eine neue Liste überführt.

## 2.5 Funktion 'flatten'

Um die funktionale Programmierung mithilfe unserer Funktionen auszuprobieren, haben wir ebenfalls die Funktion 'flatten' aus der Vorlesung implementiert. Hierbei wird deutlich wie wir die genaue Struktur der Clojure Implementierung übernehmen können.

```
function flatten(args)
    if preTable(args) == false then
```

```
        ...
    end
    if table.empty(args) then
        ...
    elseif not preTable(first(args)) then
        ...
    else
        ...
    end
end
```

Allerdings musste für diese Funktion noch eine Hilfsfunktion implementiert werden, die prüft ob eine Liste leer ist.