Learning to Program Recursive Functions

Thomas Hieber

Lehrstuhl für Angewandte Informatik - Kognitive Systeme

11. September 2008

Inhaltsverzeichnis

Rekursion

"To iterate is human, to recurse, divine"
(L. Peter Deutsch)
Rekursion kommt von lat. *recurrere* - *'zurückkehren'*, *'zurücklaufen'*

Rekursion

- Iteration (for - Schleife)
- Rekursion
- Endrekursion

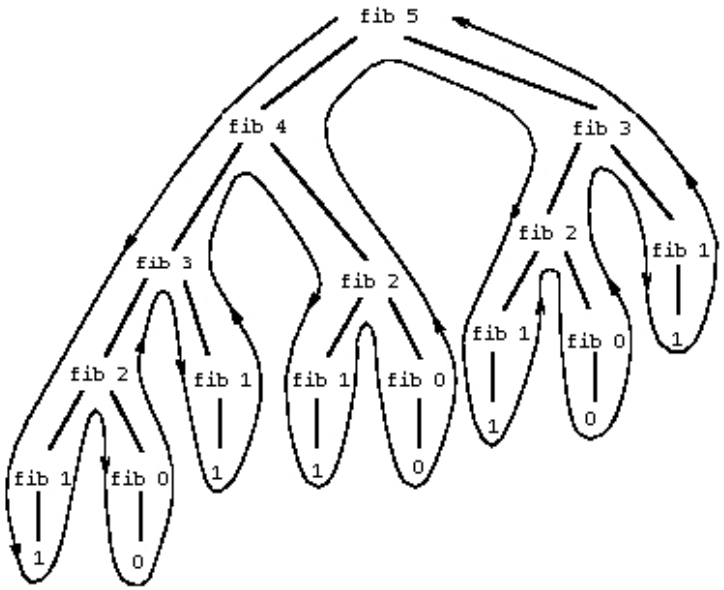

Abbildung: Fibonacci Zahlen
(`http://mitpress.mit.edu/sicp/chapter1/node13.html`)

Pea's Buggy Path

In der Arbeit von Pea über den 'Buggy Path' hin zur Programmierkenntnis werden grundsätzlich zwei Arten von Fehlern unterschieden.

- Sprachabhängige Fehler
- Sprachunabhängige

Sprachunabhängige Fehler

- Parallelism
- Intentionality
- Egocentrism
- Goal/Plan Merging

Sprachabhängige Fehler

- knowledge Unavailability
- Knowledge Unaccessability

Kognitive Problematik

"It seems that the human mind cannot suspend one process, perform a recursive calculation, and then return to the orignial suspended process"
(Pirolli)

Studien

Zahlreiche Studien haben sich mit dem Thema beschäftigt - alle stoßen auf Probleme.

- Pirolli - Fokus auf Lernen durch Beispiele
- Levi - Fokus auf Studenten und deren Terminologie
- Kruse - Detailierte Anweisungen für die Lehre
- Da Rosa - Fokus auf die Probleme an sich

Maschinenlernen

Beispiele: GRAPES, IGOR
Beide Systeme benötigen Beispiele bzw. eine Formale Spezifikation um ein Regelwerk zu erstellen, aus dem nach und nach ein Rekursives Programm abgeleitet wird.
Vorteil: Maschinen haben große Speicherkapazität Nachteil: Erzeugung der Beispiele bzw Spezifikation durch Menschen.

Beschreibung

Unter der Annahme dass die Studenten im Bamberg ähnliche Schwierigkeiten haben wie die Studenten in den erwähnten Studien wurde an der Universität Bamberg eine Studie duchgeführt, die auf die von Da Rosa formulierte Problematik des Transfers von Problemlösung hin zur Formalisierung weiter untersucht.

Sample

Teilnehmer: 30 Studierende der Informatik (Haupt- oder Nebenfach), hauptsächlich im ersten oder zweiten Semester
Expertengruppe: 5 fortgeschrittene Studenten/Absolventen aus dem Fachbereich Informatik
Voraussetzung: Fast alle Teilnehmer haben die Vorlesung Informatik I bei Prof. Wirtz gehört und dabei eine Einführung in Scheme erhalten.

Aufgaben

Zwei Arten von Aufgaben waren zu lösen:

- Erzeugen eines rekursiven Scheme Programmes
- Erzeugen einer bestimmten Zahl von Eingabe/Ausgabe Beispielen

Folgende Probleme zur Manipulation von Listen wurden gestellt:

- length
- reverse
- oddlength
- flatten

Zusätzlich stand für die Programmieraufgaben ein beschränkter Befehlssatz zur Verfügung.

Vorgehensweise

Der Testbogen enthielt zwei Exemplarische Aufgaben mit Lösung zur Verdeutlichung. Die Aufgaben mussten der Reihe nach in einer vorgegebenen Zeit bearbeitet werden. Als Vorversuch wurde in der Expertengruppe die Zeit erfasst die für die einzelnen Aufgaben benötigt wurde.
Bewertungskriterien (v.A Programmierung):

- Rekursionsabbruch korrekt
- Rekursiver Aufruf vorhanden
- Korrekte Programmsemantik

Auswertung

Deutliches Ergebnis der Studie:
Die Versuchspersonen hatten große Schwierigkeiten beim erstellen der Programme.
Das Erstellen der Besipiele hingegen funktionierte relativ gut.

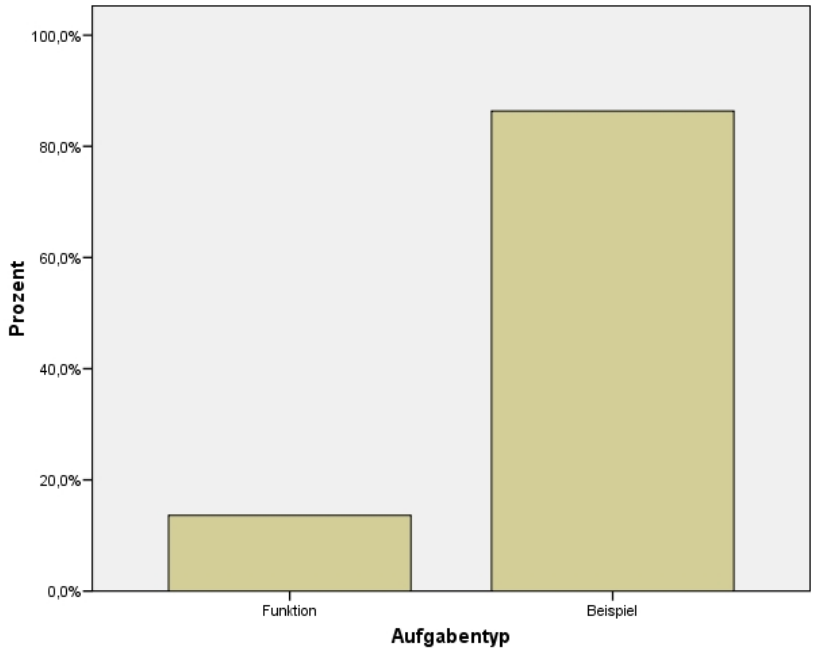

Abbildung: Korrkte Lösungen Gesamt (ohne Flatten)

Auffälligkeiten

- Verwendung von Hilfsfunktionen (Endrekursion)
- Zeichnungen verwendet als Visualisierungshilfe

Beispiel

Listing 1: "Beinahe korrektes Programmbeispiel"

```
1
2 (define (length liste) (help liste 0))
3
4 (define (help liste zaehler)
5 (if (null? (cdr liste)) zaehler
6     (help (cdr liste (+ zaehler 1)))
7 )
8 )
```

Flatten - Spezialfall

Das Problem ‘Flatten‘ war offenbar nicht nur schwer zu formalisieren, sondern auch schwer zu konzeptualisieren.
Als Beleg hierfür dient die Tatsache dass es keine gültige Programmlöslung, aber auch nur 2 korrekte Beispiellösungen gibt.

Zusammenfassung

Folgende Thesen wurden durch die Studie erhärtet:

- Die Studierenden haben Probleme bei der Lösung rekursiver Probleme
- Die Probleme bestehen vor Allem bei der Umsetzung in eine Formale Sprache
- Komplexere Probleme bereiten große Schwierigkeiten
- Das Erzeugen der Beispiele funktioniert bei weniger komplexen Problemen gut