Definiertheit rekursiver Funktionen

Gregor Alexandru

5. Juni 2019

Gliederung

Definiertheit

Achtung: der Begriff „Definiertheit“ ist nicht zu verwechseln mit dem Beweisschritt DEFINIERTHEIT im Beweis der Definiertheit einer rekursiven Funktion.

- Was meinen wir mit dem Begriff „Definiertheit“?
- Die Auswertung eines Ausdrucks soll *terminieren* und ein Ergebniss liefern
- Beispiele nicht-definierter Ausdrücke:
  - `head []` – bei der Auswertung wird eine Ausnahme (Exception) geworfen
  - `1 `div` 0` – wie oben
  - Gegeben:

    ```
    f x = f x
    ```

    `f 1` – der Funktionsaufruf terminiert nicht.

Partielle Funktionen

- In den ersten zwei Beispielen der letzten Folie waren nicht-definierte Ausdrücke die Ergebnisse der Anwendung von Funktionen auf bestimmte problematische Werte.
- Wenn Funktionen für bestimmte Werte aus ihrem Ursprung nicht definiert sind, nennen wir sie *partiell*.
- Die Funktion `head` z.B. ist *nicht* definiert für `[]`.
- Die Teilmenge der Werte des angegebenen Ursprungs einer Funktion f, für die diese *definiert* ist, nennen wir ihren *Definitionsbereich* und schreiben sie $dom(f)$.
  - Beispiel für die Funktion `head`: Ursprung ist `[a]`, Definitionsbereich ist $dom(\texttt{head}) = \{x :: [\texttt{a}] | x \neq []\}$

Rekursive Funktionen

- Rekursive Funktionen bieten eine besondere Schwierigkeit bei der Bestimmung ihres Definitionsbereichs, da die Argumentation erstmal zyklisch erscheint.
- Beispiel von vorhin:

  ```
  f x = f x
  ```

  Der Ausdruck (1)=`f a`, a beliebig, ist definiert genau dann, wenn seine Auswertung terminiert, also der Ausdruck zu dem er auswertet, also (2)=`f a`, definiert ist, also seine Auswertung terminiert, also der Ausdruck zu dem er auswertet, also (3)=`f a`, wohdefiniert ist,...

Definiertheit I

- Wir können uns erstmal abhelfen indem wir fordern dass, angenommen dass die rekursiven Aufrufe von f im Rumpf definiert sind, der Aufruf selbst auch definiert ist.
- Konkret heisst das dass mögliche partielle Funktionen[1] die im Rumpf aufgerufen werden nur mit solchen Werten aufgerufen werden, für die sie definiert sind.
- Damit schließen wir Funktionen wie `f x = f x `div` 0` von Vornherein schon aus.

[1] oder Konstrukte, die man als partielle Funktionen interpretieren kann. Erklärung folgt.

Vorgehen Definiertheit

- Wir betrachten die Werte der rekursiven Aufrufe also als black box und schauen ob der Ausdruck dann definiert ist.
- `f x = 1 `div` x - (f (x+1)) + head [f (x-2)]`

Vorgehen Definiertheit

- Wir betrachten die Werte der rekursiven Aufrufe also als black box und schauen ob der Ausdruck dann definiert ist.
- `f x = 1 `div` x - (blablii) + head [blub]`

Vorgehen Definiertheit

- Wir betrachten die Werte der rekursiven Aufrufe also als black box und schauen ob der Ausdruck dann definiert ist.
- `f x = 1 `div` x - (blablii) + head [blub]`
- Ausser ihrer Definiertheit wissen wir über die Werte nur, dass sie den Typ des Ziels der Funktion haben. Für granularere Differenzierung müssten wir den Bildbereich der Funktion ($im(f)$) finden oder Aussagen über die funktionale Abhängigkeit der Funktionswerte von ihren Eingaben treffen, was den Ramen dieser Beobachtung sprengt.

Anmerkung: partielle Konstrukte

- `case`- und `let`- Expressions kann man auch als gleichzeitige Definition und Applikation anonymer Funktionen (also Lambda-Ausdrücken) auffassen. Wie genau das funktioniert sehen wir auf den nächsten Folien. Insbesondere können diese Funktionen auch partiell sein. z.B.
- ```
  let (x:xs) = ys
  in x
  ```
  $\simeq$ `head ys` $\simeq$
  ```
  case ys of
    x:xs -> x
  ```
- Diese Ausdrücke sind äquivalent – um die Definiertheit zu zeigen müssen wir also in jedem Fall zeigen, dass `ys` nie die leere Liste sein kann, wenn der jeweilige Ausdruck ausgewertet wird.
  ```

`let` als Def & App eines λ-Ausdrucks

- ```
  let x = bar
      y = foo      ≃ (\x -> \y -> buz x y) bar foo
  in buz x y
  ```
- Die Expansion geht nicht immer, da let rekursiv ist, d.h. etwas wie z.B. `let x = 3:x in x` ≃ `repeat 3` können wir auf die Weise nicht umschreiben. Die meisten `let`-Ausdrücke jedoch die uns begegnen werden sind nicht rekursiv.
  ```

case als Def & App eines λ-Ausdrucks

- ```
  case reverse xs of
    [] -> foo
    y:ys -> bar
  ```
  $\simeq$
  ```
  (\ys ->
    case ys of
      [] -> foo
      y:ys -> bar
  ) (reverse xs)
  ```
- Erinnerung: „äußere“ Pattern-Unterscheidungen kann man mit einem case-Ausdruck immer „nach innen ziehen“. z.B.:
- ```
  foo (x:xs) = bar
  foo [] = buz
  ```
  $\simeq$
  ```
  foo = (\ys -> case ys of
                  x:xs -> bar
                  [] -> buz)
  ```

Partielle Terminierung

- Wir hatten gesehen, dass Funktionen partiell definiert sein können, also nur für bestimmte Werte des Ursprungs definiert sind. Insb. kann sich das so äußern, dass Ihre Auswertung nur für bestimmte Werte terminiert.
- Beispiel: die partielle Fakultätsfunktion:

```
fak 0 = 1
fax x = x * fak (x-1)
```

  terminiert nur für $x \in \mathbb{N} \subseteq \mathbb{Z}$
- Wir wollen also ggf. nur die partielle Definiertheit (und damit insb. Terminierung) einer Funktion zeigen, also dass `f x` definiert ist für Werte $x \in A'$, wobei A' Teilmenge des Ursprungs ist.

- Wollen wir zeigen, dass `f x` definiert ist, müssen wir erst seine Definiertheit zeigen.
- Dafür mussten wir zeigen dass der Rumpf von `f x` definiert ist, unter der Vorraussetzung, dass alle rekursiven Aufrufe im Rumpf definiert sind.
- Unser Ziel ist jetzt aber nur, für Werte $x \in A'$ die Definiertheit von `f x` zu zeigen!
- D.h. wir dürfen die Definiertheit der rekursiven Aufrufe nur dann annehmen, wenn diese wieder mit Parametern $y \in A'$ getätigt werden!
- Diese Beobachtung führt uns zur Formulierung der nächsten Eigenschaft, und danach der Umformulierung der Definiertheit.

Abgeschlossenheit, Definiertheit

Abgeschlossenheit

- Sei $f : A \to B$, $A' \subseteq A$. Wollen wir zeigen, dass für alle $x \in A'$ `f x` definiert ist, darf `f x` im Rumpf keine rekursiven Aufrufe mit Parametern $y \notin A'$ tätigen.

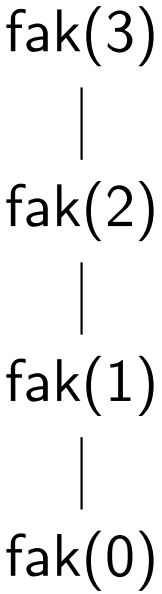
Definiertheit

- Sei $f : A \to B$, $A' \subseteq A$. Wollen wir zeigen, dass für alle $x \in A'$ `f x` definiert ist, unter der Annahme dass alle rekursiven Aufrufe die getätigt werden es sind, muss der Rumpf an sich definiert sein.

Terminierung

- Wir haben nun alle Vorkehrungen getroffen um sicher zu sein dass die Weiterverarbeitung der Ergebnisse rekursiver Aufrufe definiert sein wird. Was wir noch nicht wissen ist aber ob wir diese Ergebnisse je erhalten werden, d.h. ob die rekursiven Aufrufe irgendwann etwas zurückliefern.
- Das Problem liegt an dem Aufrufverhalten rekursiver Funktionen. Sie sollten Basisfälle haben, wir wissen aber nicht ob die unbedingt erreicht werden.
- Die längste Kette rekursiver Aufrufe bis zu einem Basisfall nennen wir die *Rekursionstiefe* eines Aufrufs
- Es folgen Beispiele mit den Aufrufbäumen der baumrekursiven *Fibonnaci*-Funktion und der linearrekursiven Fakultätsfunktion.

Beispiele Rekursionstiefe



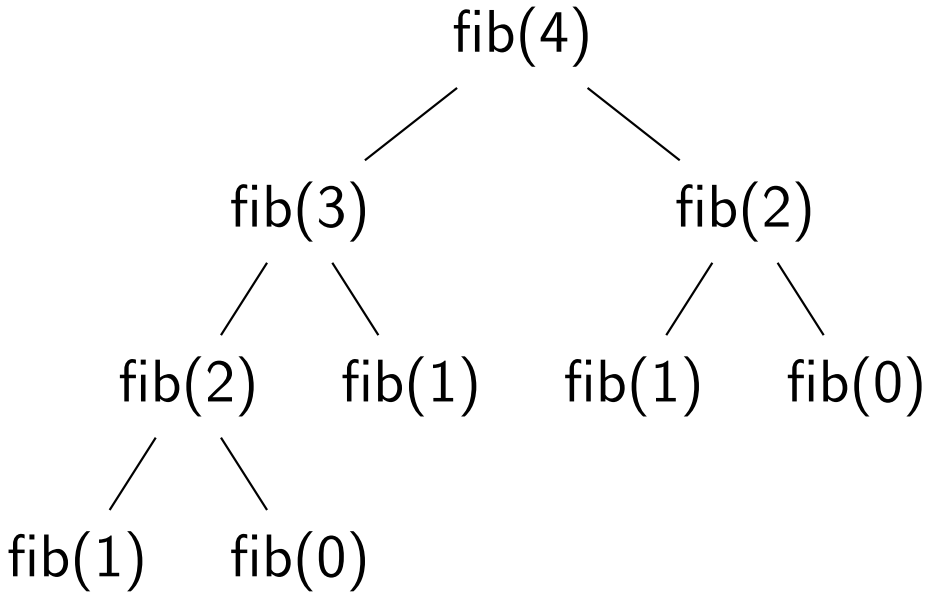


Abbildung:
depth(fak(3))=3

Abbildung:
depth(fib(4))=
1+max(depth(fib(3),fib(2)))=...=
1+max(2,1)=3

Endliche Rekursionstiefe $\Leftrightarrow$ Terminierung

- Wenn eine rekursive Funktion terminiert hat sie eine endliche Rekursionstiefe und umgekehrt.
- Ein Ansatz um herauszufinden ob eine rekursive Funktion f aufgerufen mit Parameter x terminiert wäre also eine Funktion die uns in Abhängigkeit von eben diesem x die Rekursionstiefe von `f x` angibt, oder wenigstens eine obere Schranke dafür.
- Diese Funktion wollen wir jetzt formalisieren.

Abstiegsfunktion I

- Sei $f : A \to B$ eine rekursive Funktion, die auf der Teilmenge $A' \subseteq A$ terminieren soll.
- f terminiert genau dann für alle Werte aus A', wenn es eine Funktion $m : A' \to \mathbb{N}$ gibt[2], die für jedes $x \in A'$ eine obere Schranke für die Rekursionstiefe depth(f(x)) zurückgibt.
- Wie können wir aber überprüfen, dass m seine Spezifikation erfüllt?

[2] Die Funktion kann auch Ursprung A haben, muss aber nur für $x \in A'$ definiert sein

Test auf Abstieg

- Ein möglicher Test ist folgende Beobachtung: Wenn $m(x) = n$ sollte $m(y) < n$ bzw $m(y) \leq n - 1$ für alle rekursiven Aufrufe `f y` die bei der Auswertung von `f x` getätigt werden.
- Denn: Bei einem getätigten Schritt in die Rekursionstiefe sollte die Abschätzung für die verbleibende Tiefe mindestens um 1 verringert werden.
- Wäre die Abschätzung bei einem rekursiven Aufruf größer oder gleich, so wäre die ursprüngliche Abschätzung falsch gewesen.
- Veranschaulichung: Keine Abstiegsfunktion ist z.B. die Anzeige der Restwartezeit auf öffentliche Verkehrsmittel: Wenn nach einer Minute die Wartezeit immer noch 20 Minuten beträgt oder gar 21, kann die Anzeige von 20 Minuten vor einer Minute nicht gestimmt haben.

- Wir haben also einen geeigneten Test auf Abstieg gefunden. Damit formalisieren wir die Abstiegsfunktion neu.
- Um das Bild abzurunden schreiben wir diese neue Formalisierung zusammen mit denen der Definiertheit und Abgeschlossenheit auf.

Definiertheit rekursiver Funktionen

Sei $f : A \to B$, $A' \subseteq A$. Wollen wir zeigen, dass für alle $x \in A'$ `f x` definiert ist, muss sie folgende 3 Eigenschaften erfüllen:

Abgeschlossenheit

- Für alle $x \in A'$: `f x` tätigt im Rumpf rekursive Aufrufe nur mit Parametern $y \in A'$.

Definiertheit

- Für alle $x \in A'$: Unter der Annahme dass alle rekursiven Aufrufe die von `f x` getätigt werden definiert sind, muss der Rumpf an sich definiert sein, d.h. wenn partielle Funktionen g mit Parameter(n) z aufgerufen werden dann gilt immer $z \in dom(g)$

Abstiegsfunktion

- Es gibt eine Funktion $m : A' \to \mathbb{N}$: Für alle $x \in A'$: wenn bei der Auswertung von `f x` f rekursiv mit Parameter(n) y aufgerufen wird gilt $m(y) < m(x)$ bzw. $m(y) \leq m(x) - 1$