Einführung in die Informatik 2 - Repetitorium WS 2016/17
Technische Universität München

Kevin Kappelmann 29. März 2017

Äquivalenzbeweise

1 Theorie

Wie wir vielleicht bemerkt haben, sind Big-Step-Ableitungsbäume kein Spaß zu zeichnen und werden nicht selten sehr groß. Glücklicherweise können wir die Semantik unserer Programme auch durch eine etwas lockerere Art beweisen, indem wir einfach Definitionen und Substitutionen, wie in einem klassischen mathematischen Beweis, verwenden, um zwei Ausdrücke schrittweise ineinander umzuführen.
Zum Beispiel macht es doch sehr Sinn, zum Beweis von $(let\ f = 3\ in\ f * (1 + 2)) = 9$ nicht mühselig einen Big-Step-Ableitungsbaum zu zeichnen, sondern den Beweis wie folgt zu führen

$$f * (1 + 2) \overset{Def.\ f}{=} 3 * (1 + 2) = 3 * (3) = 9$$

Ja, das sieht doch gut aus! Aber... warum machen wir dann überhaupt Big-Step-Beweise? Zur Knechtschaft der Studenten? Gewiss! Aber natürlich auch aus einem anderen Grund. Betrachten wir folgende Definitionen

$$let\ rec\ f\ x = f\ x$$
$$let\ g\ x = 0$$

Wir wollen nun $g\ (f\ 1) = 0$ beweisen. Also schreiben wir

$$g\ (f\ 1) \overset{Def.\ g}{=} 0$$

Halt, nicht so schnell! Die Funktion f terminiert offensichtlich nicht. Geben wir Definitionen wie oben in den OCaml-Interpreter ein, so erhalten wir auch, dass der Aufruf $g\ (f\ 1)$ nicht terminiert. Unser Beweis ist also falsch. Wir haben die Schwachstelle unserer Vereinfachung gefunden. **Das ganze Vorhaben funktioniert nur unter dem Vorbehalt, dass alle Ausdrücke terminieren!**. Deswegen wird auch in Übungen explizit der Zusatz "Sie dürfen annehmen, dass alle Aufrufe terminieren" genannt, sollte ein Äquivalenzbeweis ohne Big-Step-Ableitungsbaum verlangt sein.

2 Fallbeispiel

Wie können wir nun das Ganze nutzen, um die Semantik unserer Programme zu überprüfen? Dies sollen die folgenden zwei Beispiele erläutern.

2.1 Schon wieder Gauß?

Gegeben sind folgende OCaml-Definitionen

$$let\ rec\ f\ n\ = if(n <= 1)\ then\ n\ else\ n + f\ (n - 1) \tag{1}$$

Zeigen Sie, dass für alle $n \in \mathbb{N}_0$ gilt, unter der Voraussetzung, dass alle Aufrufe terminieren:

$$f\ n = \frac{n * (n + 1)}{2}$$

Beweis. Wir sehen, dass die Definition von f rekursiv von n abhängt und n in jedem Schritt kleiner wird. Daher zeigen wir die Aussage mit Induktion über n.

- Induktionsbasis: Sei n=0, dann
$$f\ 0 \overset{Def.\ f}{=} 0 = \frac{0 * (0+1)}{2}$$
- Induktionsschritt: Sei $n \in \mathbb{N}_0$ beliebig fixiert
  - Induktionshypothese: Es gelte $f\ n = \frac{n*(n+1)}{2}$
  - Induktionsbehauptung: Dann gilt auch $f\ (n+1) = \frac{(n+1)*((n+1)+1)}{2}$
  - Beweis:
    * Fall $n = 1$:
$$f\ 1 \overset{Def.\ f}{=} 1 = \frac{1 * (1+1)}{2}$$
    * Fall $n > 1$:
$$f\ (n+1) \overset{Def.\ f}{=} n+1+f\ n \overset{I.H.}{=} n+1+\frac{n*(n+1)}{2} = n+1+\frac{n^2+n}{2} = \frac{2n+2+n^2+n}{2}$$
$$= \frac{(n^2+2n+1)+n+1}{2} = \frac{(n+1)^2+n+1}{2} = \frac{(n+1)*((n+1)+1)}{2}$$

□

2.2 Die Kunst der Abstrahierung

Gegeben sind folgende OCaml-Definitionen

$$\begin{aligned} &let\ rec\ append\ a\ b = match\ a\ with\ [] \to b \mid x :: xs \to x :: append\ xs\ b \\ &let\ (@) = append \\ &let\ rec\ aux\ a = function\ [] \to a \mid x :: xs \to aux\ (x :: a)\ xs\ in\ aux\ []\ l \\ &let\ rec\ rev = function\ [] \to [] \mid x :: xs \to rev\ xs\ @\ [x] \end{aligned}$$

und außerdem folgendes Lemma

$$a\ @\ (b\ @\ c) = (a\ @\ b)\ @\ c \tag{2}$$

Zeigen Sie, unter der Voraussetzung, dass alle Funktionsaufrufe terminieren, dass für alle Listen l gilt:

$$aux\ []\ l = rev\ l$$

Beweis. Wir sehen, dass die Definitionen von rev und aux rekursiv von der Liste l abhängen und diese in jedem Schritt verkleinert wird. Daher zeigen wir die Aussage mit Induktion über die Länge der Liste mit $n := length\ l$.

- Induktionsbasis: Sei n=0, dann gilt $l = []$ und
$$aux\ []\ [] \overset{Def.\ aux}{=} [] \overset{Def.\ rev}{=} rev\ []$$
- Induktionsschritt: Sei $n \in \mathbb{N}_0$ beliebig fixiert, l eine beliebige Liste mit $n = length\ l$ und x ein beliebiges Element, sodass $n + 1 = length\ (x :: l)$.
  - Induktionshypothese: Es gelte $aux\ []\ l = rev\ l$
  - Induktionsbehauptung: Dann gilt auch $aux\ []\ (x :: l) = rev\ (x :: l)$
  - Beweis:
$$aux\ []\ (x :: l) \overset{Def.\ aux}{=} aux\ [x]\ l \quad \lightning$$
    Hier stehen wir vor einem Problem. Unsere Induktionshypothese gilt nur für $aux\ []\ l = rev\ l$, nicht aber für $aux\ [x]\ l$. Dies passiert häufig in solchen Beweisen. Wir müssen die Hypothese verallgemeinern, um den Beweis durchführen zu können.

Neuer Versuch: Wir zeigen zunächst $aux\ a\ l = rev\ l\ @\ a$ über die Länge der Liste mit $n := length\ l$ und beliebigen Listen a.

- Induktionsbasis: Sei n=0, dann gilt $l = []$ und

$$aux\ a\ [] \overset{Def.\ aux}{=} a \overset{Def.\ @}{=} [] @ a \overset{Def.\ rev}{=} rev\ [] @ a$$

- Induktionsschritt: Sei $n \in \mathbb{N}_0$ beliebig fixiert, l eine beliebige Liste mit $n = length\ l$ und x ein beliebiges Element, sodass $n + 1 = length\ (x :: l)$.
  - Induktionshypothese: Es gelte $aux\ a\ l = rev\ l\ @\ a$
  - Induktionsbehauptung: Dann gilt auch $aux\ a\ (x :: l) = rev\ (x :: l)\ @\ a$
  - Beweis:

$$\begin{aligned} aux\ a\ (x :: l) &\overset{Def.\ aux}{=} aux\ (x :: a)\ l \overset{I.H.}{=} rev\ l\ @\ (x :: a) \\ &\overset{Def.\ @}{=} rev\ l\ @\ (x :: [] @ a) \overset{Def.\ @}{=} rev\ l\ @\ ([x] @ a) \\ &\overset{Lemma\ (2)}{=} (rev\ l\ @\ [x]) @ a \overset{Def.\ rev}{=} rev\ (x :: l) @ a \end{aligned}$$

Mit dem soeben gezeigten Lemma folgt mit $a = []$ und beliebigen Listen l

$$aux\ []\ l = rev\ l\ @\ [] = rev\ l$$

Der Beweis der letzten Gleichheit und Lemmas (2) überlassen wir dem Leser als Übung. □

3 Zusammenfassung und Tipps

Eine kleine Zusammenfassung, zu den gerade gesehenen Beweisen und ein paar Tipps:

- Wir nehmen bei dieser Art Beweis immer an, dass alle Aufrufe terminieren
- Um die Beweise durchführen zu können, benötigen wir häufig Induktion. Über welche Variable dabei die Induktion durchgeführt wird, liegt daran, worüber die Funktion, über die der Beweis eine Aussage trifft, definiert ist. Die Induktionsvariable muss dabei immer kleiner werden, ansonsten können wir nicht induktiv argumentieren.
- Bei Funktionen mit mehreren Parametern ist zunächst herauszufinden, über welche der Parameter die Funktion rekursiv definiert ist. Sollte es mehr als ein Parameter sein, so muss man etwas tricksen und z.B. die Induktion über die Summe der (Längen) der beiden Parameter führen.
- Manchmal müssen wir die übrigen Parameter beliebig setzen, um die Induktionshypothese im Induktionsschritt verwenden zu können (vgl. Beispiel 2.2).
- Definitionen von Funktionen können vorwärts als auch rückwärts verwendet werden ("=" ist schließlich symmetrisch).
- Manchmal ist es hilfreich, nicht zwanghaft zu versuchen, von einer Seite auf die andere umzuformen, sondern beide Seiten getrennt so weit wie möglich zu vereinfachen, um sich dann in der Mitte zu treffen.
- Sachen, die bereits bewiesen wurden, können wir wiederverwenden. Dies ist vor allem in Klausuren wichtig, in denen Lemmata gegeben sind und nach mehreren Beweise gefragt wird. Meist muss man in einem Schritt dann ein vorher erzieltes Ergebnis (oder ein gegebenes Lemma) verwenden.

4 Übungen

2016/17

- Blatt 13

2015/16

- Blatt 12

GitHub

- 2016/equivalence_proofs

In den Vorlesungsfolien sind auch 2, 3 Beweise.
Ansonsten: Einfach selber Programme schreiben und diese beweisen! :)