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

Kevin Kappelmann 27. März 2017

Big-Step

1 Theorie

Big-Step ist eine Art der operationellen Semantik, im Gegensatz zum wp-Kalkül, dass der axiomatischen Semantik zuzuordnen ist. Es werden Axiome und Regeln festgelegt, die die abstrakte Ausführung eines Programmes beschreiben. Dabei werden nicht, wie in einem Computer, sequentiell alle Schritte einzeln ausgeführt, bis das Programmende erreicht wird (das wäre die Small-Step-Semantik), sondern es wird das ganze Programm als ein einziger großer Schritt betrachtet. Ein Schritt bedeutet hierbei so viel wie "ein gültiger Ableitungsbaum aus den Axiomen und Regeln".
Wir betrachten die Big-Step-Regeln aus der Vorlesung.

Axiome: $v \Rightarrow v$ für jeden Wert v

Tupel: $$\frac{e_1 \Rightarrow v_1 \quad \dots \quad e_k \Rightarrow v_k}{(e_1, \dots, e_k) \Rightarrow (v_1, \dots, v_k)} \ (T)$$

Listen: $$\frac{e_1 \Rightarrow v_1 \quad e_2 \Rightarrow v_2}{e_1 :: e_2 \Rightarrow v_1 :: v_2} \ (L)$$

Globale Definitionen: $$\frac{f = e \quad e \Rightarrow v}{f \Rightarrow v} \ (GD)$$

Lokale Definitionen: $$\frac{e_1 \Rightarrow v_1 \quad e_0[v_1/x] \Rightarrow v_0}{\texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_0 \Rightarrow v_0} \ (LD)$$

Funktionsaufrufe: $$\frac{e_1 \Rightarrow \texttt{fun}\ x\ \texttt{->}\ e_0 \quad e_2 \Rightarrow v_2 \quad e_0[v_2/x] \Rightarrow v_0}{e_1\ e_2 \Rightarrow v_0} \ (App)$$

Funktionsaufrufe mit mehreren Argumenten: $$\frac{e_0 \Rightarrow \texttt{fun}\ x_1 \dots x_k\ \texttt{->}\ e \quad e_1 \Rightarrow v_1 \dots e_k \Rightarrow v_k \quad e[v_1/x_1 \dots v_k/x_k] \Rightarrow v}{e_0\ e_1 \dots e_k \Rightarrow v} \ (App')$$

Pattern Matching: $$\frac{e_0 \Rightarrow v' \equiv p_i[v_1/x_1, \dots, v_k/x_k] \quad e_i[v_1/x_1, \dots, v_k/x_k] \Rightarrow v}{\texttt{match}\ e_0\ \texttt{with}\ p_1\ \texttt{->}\ e_1\ \texttt{|}\ \dots\ \texttt{|}\ p_m\ \texttt{->}\ e_m \Rightarrow v} \ (PM)$$

— sofern v' auf keines der Muster $p_1, \dots, p_{i-1}$ passt

Eingebaute Operatoren: $$\frac{e_1 \Rightarrow v_1 \quad e_2 \Rightarrow v_2 \quad v_1\ \texttt{op}\ v_2 \Rightarrow v}{e_1\ \texttt{op}\ e_2 \Rightarrow v} \ (Op)$$

— Unäre Operatoren werden analog behandelt.

Substitutionslemma

$$\frac{e_1 = e_2}{e[e_1/x] = e[e_2/x]}$$

Abbildung 1: Big-Step-Regeln

Da fragt man sich vielleicht zunächst, was dieses "⇒" bedeutet und was der Unterschied zu einem "=" sein soll.

- $\underline{\Rightarrow}$: Lesen wir $e \Rightarrow v$ für einen Ausdruck e und eine Wert v, so bedeutet dies "e wertet sich zu v aus". Ein Wert ist dabei, etwas salopp gesagt, nichts anderes als ein Ausdruck, der nicht weiter durch die oben genannten Regeln vereinfacht werden kann.
  Beispielsweise können wir $1 + 2 + 3 \Rightarrow 6$ oder $1 \Rightarrow 1$ oder auch $fun\ x \to x + 1 \Rightarrow fun\ x \to x + 1$ schreiben. Hingegen ist $fun\ a\ b \to a + b \Rightarrow fun\ a\ b \to b + a$ **nicht** gültig.

- $\underline{=}$: Lesen wir $e_1 = e_2$ für zwei Ausdrücke e_1 und e_2, so bedeutet dies - erneut etwas salopp formuliert - "e_1 und e_2 verhalten sich äquivalent". Dies bedeutet insbesondere, dass aus $e \Rightarrow v$ sofort $e = v$ folgt.
  Beispielsweise können wir $1 + 2 + 3 = 6$ oder $fun\ a\ b \to a + b = fun\ a\ b \to b + a$ oder $(let\ rec\ f\ x = f\ x) = (let\ rec\ g\ x\ y = g\ 1\ 2)$ schreiben. Im letzten Beispiel tritt das Gleichheitssymbol sowohl als Definitionsoperator von OCaml ("="), als auch als Gleichheitssymbol für unsere Big-Step-Semantik ("=") auf und die beiden Terme sind äquivalent, da sie beide nicht terminieren und sich somit äquivalent verhalten.

2 Fallbeispiele

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

2.1 Einfacher Ableitungsbaum

Gegeben sind folgende OCaml-Definitionen

$$\begin{aligned} let\ w\ l = match\ l\ with\ []\ &\to 0 \mid x :: xs\ \to x \\ let\ r = match\ w\ [1;2]\ with\ 0\ &\to 42 \mid x\ \to x + 10 * 2 \end{aligned}$$

Zeigen Sie, dass r=21 gilt.

Beweis. Wir definieren uns zunächst ein paar Kürzel, um den Beweisbaum übersichtlich zu halten

$$\begin{aligned} e_0 &:= match\ w\ [1;2]\ with\ 0\ \to 42 \mid x\ \to x + 10 * 2 \\ e_1 &:= match\ l\ with\ []\ \to 0 \mid x :: xs\ \to x \\ f_0 &:= fun\ l \to e_1 \end{aligned}$$

und beginnen dann von unten den Ableitungsbaum zu zeichnen. Die Anwendung des Axiomes $v \Rightarrow v$ lassen wir zu Gunsten der besseren Übersicht weg.

$$(GD^1)\ \frac{r = e_0 \quad (PM^2)\ \dfrac{(APP^3)\ \dfrac{(GD^4)\ \dfrac{w = f_0}{w \Rightarrow f_0} \quad (L^5)\ \dfrac{(L^6)\ \overline{[2] \Rightarrow [2]}}{[1;2] \Rightarrow [1;2]} \quad (PM^7)\ \dfrac{(L^8)\ \dfrac{(L^9)\ \overline{[2] \Rightarrow [2]}}{[1;2] \Rightarrow [1;2] \equiv 1 :: [2]} \qquad x[1/x] \Rightarrow 1}{e_1[[1;2]/l] \Rightarrow 1}}{w\ [1;2] \Rightarrow 1 \equiv x[1/x]} \quad \dfrac{\dfrac{10 * 2 \Rightarrow 20}{10 * 2 \Rightarrow 20}\ (OP^{11}) \qquad 1 + 20 \Rightarrow 21}{1 + 10 * 2 \Rightarrow 21}\ (OP^{10})}{e_0 \Rightarrow 21}}{r \Rightarrow 21}$$

Im Superskript der angewendeten Regel steht hier jeweils der Zeitpunkt, zu dem die Regel angewandt wurde, um euch den Ablauf klarer zu machen. Im Prinzip schreibt man sich die zu behauptende Aussage unten auf sein Blatt auf und versucht dann bottom-up mit Hilfe der Regeln aus Abb. 1 einen gültigen Ableitungsbaum zu finden. Dabei arbeitet man sich bei jedem Schritt von links nach rechts voran. Wenn man dann einen Wert nach oben berechnet hat, kann man diesen als Ergebnis der Auswertung nach unten einsetzen.

□

2.2 Induktion - Terminierung

Gegeben sind folgende OCaml-Definitionen

$$let\ rec\ f\ l = match\ l\ with\ []\ \to 0 \mid x :: xs\ \to x + f\ xs$$

Zeigen Sie, dass für alle Integer-Listen l der Aufruf f l terminiert.

Beweis. Wir zeigen die Behauptung mit Induktion.
Wie wir auf dem Grundlagenblatt für Induktion gelernt haben, brauchen wir für eine Induktion eine partielle Ordnung $\preceq$, die keine unendlich absteigende Ketten besitzt. Nun lässt sich diese einfach für den Datentyp der Listen festlegen, indem wir für zwei Listen l1 und l2 definieren

$$l1 \preceq l2 : \iff \exists y.\ l2 = y@l1$$

Somit können wir vereinfacht ausgedrückt sagen: Wir führen die Induktion über die Länge der Liste $n \coloneqq length\ l \in \mathbb{N}_0$.
Wir setzen nun

$$e_0 \coloneqq match\ l\ with\ []\ \to 0 \mid x :: xs\ \to x + f\ xs$$
$$f_0 \coloneqq fun\ l \to e_0$$

Die Anwendung des Axiomes $v \Rightarrow v$ lassen wir zu Gunsten der besseren Übersicht weg.

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

$$\text{(APP)}\ \frac{\text{(GD)}\ \dfrac{f = f_0}{f \Rightarrow f_0}\quad \text{(PM)}\ \dfrac{[] \Rightarrow [] \equiv []\qquad 0 \Rightarrow 0}{e_0[[]/l] \Rightarrow 0}}{f\ [] \Rightarrow 0}$$

  Somit terminiert der Basisfall.

- Induktionsschritt: Sei $n \in \mathbb{N}_0$ beliebig fixiert, l eine beliebige Liste mit $n = length\ l$ und $x \in \mathbb{Z}$ beliebig, sodass $n + 1 = length\ (x :: l)$.
  - Induktionshypothese: Es gelte, dass $f\ l$ terminiert.
  - Induktionsbehauptung: Dann gilt auch, dass $f\ (x :: l)$ terminiert.
  - Beweis:

$$\text{(APP)}\ \frac{\text{(GD)}\ \dfrac{f = f_0}{f \Rightarrow f_0}\quad \text{(L)}\ \dfrac{}{x :: l \Rightarrow x :: l}\quad \dfrac{x :: l \Rightarrow x :: l \equiv x :: xs[x/x, l/xs]\qquad \text{(OP)}\ \dfrac{\text{(I.H.)}\ \dfrac{}{f\ l \Rightarrow v}\qquad x + v \Rightarrow v'}{x + f\ l \Rightarrow v'}}{e_0[(x :: l)/l] \Rightarrow v'}\ \text{(PM)}}{f\ (x :: l) \Rightarrow v'}$$

□

Grundsätzlich hätten wir auch strenger zeigen können, dass die Funktion f nicht nur terminiert, sondern wirklich die Summe der Zahlen der Liste berechnet. Einfacher und übersichtlicher ist es aber meist, mit Big-Step die Terminierung einer Funktion zu zeigen und anschließend die Semantik des Programmes mit einem gelockerten Formalismus zu beweisen (siehe equivalence_proofs).

3 Übungen

Semester 2016/17

- Blatt 12

GitHub

- 2016/big_step

Einfach selber Programme schreiben und diese beweisen! :)