Christian-Albrechts-Universität zu Kiel
Institut für Informatik und Praktische Mathematik
Prof. Dr. W.-P. de Roever
Harald Fecher, Martin Steffen

Fool

Sommersemester 2005

Serie 3

April 29, 2005

Thema: ς-calc. (Aufgaben mit Lösungshinweisen)

Ausgabetermin: April 29, 2005

Abgabe:

Aufgabe 1 (Free variables) Ponder the definition of free variables in [1, p. 61] ("object scoping"). During the lecture, there had been shortly an "optimized" definition of that function in the case of *method update* on the blackboard. It looked as follows:

$$fv(b.l_j{:=}\varsigma(y).b) \triangleq fv(b) \setminus \{y\} \cup \bigcup_{i \in \{1,\ldots,n\} \setminus \mathbf{j}} (fv(b_i) \setminus x_i)$$

where the **j** is removed. What are the consequences of that optimized definition? Especially in connection with the substitution operation defined immediately afterwards in the book.

Lösung: The real definition is given at page 61:

$$fv((a.l{:=}\varsigma(y).b) = fv(a) \cup fv(\varsigma(y)b) \ .$$

Intuitively, the definition "carefully" takes into account all potentially free variables, i.e., it does not rely on the "fact" that the method l in object a is updated "anyway". That's in general a bad idea to mix up the notion of free variable (or similar notions) with the dynamic notion of reduction. It would be like a statement

$y \notin fv((\lambda x.t)y)$, if $x \notin fv(t)$

The intuition is the same: we know that y is thrown away and thus we conclude that it does not occur free. First of all, that's nonsense.[1] Second of all it can change the semantics, because we assume that as the very next step, y is fed into the argument.

Here is an argument why it is a bad definition. One would like to have a statement like the following:

[1] Well, perhaps not nonsense. In some sense, arguing this way throws in flow-analysis.

> The semantics of a program depends *only* on the values of its (finitely many) free variables (and the text of the program itself, of course)

That's very typical of syntax (of all kinds) where one has (free) variables.[2] More formally,

$$\text{if} \quad x \notin fv(t), \quad \text{then} \quad t\{\!|a/x|\!\} \equiv t\{\!|b/x|\!\} \ , \tag{1}$$

for all a and b, where by $\equiv$ we just mean an informal notion of equivalence. If we don't count y in the λ-example or the free variables of the replaced method in the oc-example among the free variables, this intuitive equation does no longer hold, if we substitute the *diverging* term.[3]

That was a more or less informal reasoning why "optimizing" the definition of free variables means we are heading for trouble. We should consult the real definition of [1] to see the consequences for the official semantics. The above observation also concerns the reduction semantics, in particular the non-deterministic $\rightarrow$, respectively *substitution* which forms the core of the computation mechanism. Let's look at the relevant case of substitution:

$$(a.l{:=}\varsigma(s).b)\{\!|x/c|\!\} \triangleq (a\{\!|c/x|\!\}).l{:=}(\varsigma(s).b)\{\!|x/c|\!\} \tag{2}$$

This means the substitution concerns the full a; i.e., even if a is of the form $\{\ldots, l(s) = d(x), \ldots\}$, where the method body d contains x, the x is replaced by c. So if we take the $\rightarrow$-semantics, the definition is "*wrong*", however its a bit tricky, since the substitution does not "directly" make use of the definition of free variables. Let's take the following example:

$$a = \{l(s) = (b.l'{:=}\varsigma(y).y)\} \quad \text{where} \quad b = \{l'(s') = f(x,z)\} \tag{3}$$

With the alternative definition we have $x \notin fv(a)$. That is however, not relevant. What breaks the neck of the definition of substitution is that z is not counted as free (the status of x is not so important, even if we could have suspected that after the above discussion)! Let's calculate:

$$\begin{array}{rcll} a\{\!|c/x|\!\} & = & \{l(\mathbf{z}) = ((b.l'{:=}\varsigma(y).y)\{\!|z/s|\!\}\{\!|c/x|\!\}\} & \mathbf{z} \notin fv(\varsigma(s).(b.l'{:=}\varsigma(y).y), c, x) \\ & = & \{l(\mathbf{z}) = ((b.l'{:=}\varsigma(y').y')\{\!|c/x|\!\}\} & \\ & = & \{l(\mathbf{z}) = (((b\{\!|c/x|\!\}).l'{:=}\varsigma(y'').y'')\} & \\ & = & \{l(\mathbf{z}) = ((((\{l'(s'') = f(c,\mathbf{z})\}).l'{:=}\varsigma(y'').y'')\} \ . & \end{array}$$

So here we have again the effect of parasitary or dynamic binding. Since we did not count z to occur free, z is now *captured.* Still, if it is assured that the method update is evaluated *before* anything involving z is evaluated, that's ok. For the $\rightsquigarrow$-semantics, it seems acceptable. For the $\rightarrow$-semantics, the captured binding can be used to construct a un-semantical example, involving divergence. For instance, choosing to be a method call

$$f(x,z) = z.l$$

[2] One can even use it as a "syntax-free" definition of free variables.

[3] Note however, that the definition of substitution makes only indirect use of the free variables, i.e., it substitution is not formalized in stating "replace all free occurrences ... such that ...".

The capture of z by the binding suddenly makes $z.l$ to a *recursive call* to the outermost object. This certainly is *wrong* as it does not match the intended meaning of the program from Equation (3). The opinion, that "maybe the behavior is exactly what the programmer had in mind" is not tenable, since it was simply *by accident* namely by renaming s to a variable z which was considered to be available, that the diverging behavior occurred. □

Aufgabe 2 (Divergenz) Geben Sie einen ς-Term an, der terminierende und nicht terminierende Reduktionsschritte bzgl. $\longrightarrow$ hat.

Lösung: That's simple. First define a non-terminating term. It's quite easier than in the λ-calculus, we do not have to come up with strange things such as Y-combinators ... Instead we can directly write down *recursion*:

$$\omega \triangleq \{l = \varsigma(s).\ s.l\}\ . \tag{4}$$

It's easy to check that

$$\omega.l \to_\beta (s.l)\{\!|\omega/s|\!\} = (s.l)\{\!|\omega|\!\} = \omega.l \ldots$$

While at it, let's compare it also with the way the weak-reduction, big-step semantics handles the situation:

$$\dfrac{\dfrac{\vdots}{(s.l)\{\!|o/s|\!\} = o.l \rightsquigarrow}\ \textsc{Call}}{o.l \rightsquigarrow}\ \textsc{Call}$$

We see that there is is no derivation such that $o.l \rightsquigarrow t$. This is typical: divergence (or also other failures) are represented by absence of a derivation.

Now we simply need an object with a method that throws away its argument.

$$o \triangleq \{l = \varsigma(s).[]\}$$

Then $o.l(\omega)$ may or may not diverge. Note the contextual definition. A *context* is a "term with a hole", written $C[]$. For instance putting the hole "around" ω gives

$$o.l([\omega])$$

which diverges. If we use the empty context, i.e., put the whole around the complete term, yielding $[o.l(\omega)]$, the reduction rule throws away the ω. □

Aufgabe 3 (Soundness of weak reduction) Beweisen Sie Proposition 6.2-3 aus [1].

Lösung:

Aufgabe 4 Definieren Sie *induktiv* die Menge der ς-Terme (auch als Normalformen bezeichnet) bei denen keine $\to$ Schritte mehr möglich sind.[4] Zeigen Sie, daß kein ς-Term zu zwei verschiedenen Normalformen reduzieren kann. Benutze dafür den Satz von Church Rosser (Theorem 6.2-2 auf Seite 62).

Lösung: The definition is not 100% context free but reasonably so. We just must form terms where no $\{\}$

[4] $\to$ ist die *"Einschrittdefinition im Kontext"* die auf Seite 62 definiert ist.

Literatur

[1] Martín Abadi and Luca Cardelli. *A Theory of Objects*. Monographs in Computer Science. Springer, 1996.