A Refuting Conjectures 1 and 3

Inspired by an example of [15] we have found that the presented denotational treatment of recursive let-expressions is not consistent with the operational behavior. More precisely, the denotation of expressions that contain recursive let-expressions may consist of more results than it is supposed to. Let us demonstrate this by considering the following expression, which is actually very similar to an example of [14, to show that rule (VarExp) of [2] is inappropriate]:

$$\textbf{let}\ b = \mathsf{True}\ ?\ \textbf{case}\ b\ \textbf{of}\ \{\mathsf{True} \to \mathsf{False}; \mathsf{False} \to \mathsf{False}\}\ \ \textbf{in}\ b$$

Evaluating this expression in KiCSi yields True as first result. Asking for more results leads to nontermination. This is the intended behavior in the presence of call-time choice: since b is a variable it can only be bound to one deterministic choice. Therefore, the evaluation of the term above should yield the union of the results of the evaluation of $\textbf{let}\ b = \mathsf{True}\ \textbf{in}\ b$ and $\textbf{let}\ b = \textbf{case}\ b\ \textbf{of}\ \{\mathsf{True} \to \mathsf{False}; \mathsf{False} \to \mathsf{False}\}\ \ \textbf{in}\ b$, i.e., denotationally the union of $\{\mathit{True}\}$ and $\emptyset$. But the denotational semantics we presented additionally yields the result *False*. Let us examine the corresponding calculation in a bit more detail:

$$\begin{aligned}
&[\![\textbf{let}\ b = \mathsf{True}\ ?\ \textbf{case}\ b\ \textbf{of}\ \{\mathsf{True} \to \mathsf{False}; \mathsf{False} \to \mathsf{False}\}\ \ \textbf{in}\ b]\!]_{\emptyset,\emptyset}\\
= &\bigsqcup\nolimits_{\mathbf{t} \in \mathbf{T}_{b=\mathsf{True}?\textbf{case}\ b\ \textbf{of}\ \{\mathsf{True}\to\mathsf{False};\mathsf{False}\to\mathsf{False}\}}} [\![b]\!]_{\emptyset,[b\mapsto\mathbf{t}]}\\
&\text{with } \mathbf{T}_{b=\mathsf{True}?\textbf{case}\ b\ \textbf{of}\ \{\mathsf{True}\to\mathsf{False};\mathsf{False}\to\mathsf{False}\}}\\
&\qquad = \min\{\mathbf{t} \mid \mathbf{t} \in\\
&\qquad\qquad \max(([\![\mathsf{True}\ ?\ \textbf{case}\ b\ \textbf{of}\ \{\mathsf{True} \to \mathsf{False}; \mathsf{False} \to \mathsf{False}\}]\!]_{\emptyset,[b\mapsto\mathbf{t}]})_\perp)\}\\
&\qquad = \min\{\mathbf{t} \mid \mathbf{t} \in \max((\{\mathit{True}\} \cup \begin{cases}\emptyset & \text{if } \mathbf{t} = \perp\\ \{\mathit{False}\} & \text{otherwise}\end{cases})_\perp)\}\\
&\qquad = \{\mathit{True}, \mathit{False}\}\\
= &\{\mathit{True}, \mathit{False}\}
\end{aligned}$$

The problem becomes visible best in the third-last line of the calculation. Let us assume that the result that originates from the non-recursive part of the right-hand side of the variable binding, namely $\{\mathit{True}\}$, is not present. In this case possible values for $\mathbf{t}$, over which to minimize, are exactly $\perp$ and *False*, because $\perp \in \max(\emptyset_\perp)$ and $\mathit{False} \in \max(\{\mathit{False}\}_\perp)$, but $\mathit{True} \notin \max(\{\mathit{False}\}_\perp)$. After minimization only $\perp$ remains. If we, however, reconsider the original situation where $\{\mathit{True}\}$ *is* present, $\perp$ does not even take part in the minimization, because $\perp \notin \max((\{\mathit{True}\} \cup \emptyset)_\perp)$. Due to $\mathit{True}, \mathit{False} \in \max((\{\mathit{True}\} \cup \{\mathit{False}\})_\perp)$ we now have to minimize over the set $\{\mathit{True}, \mathit{False}\}$ rather than over the set $\{\perp, \mathit{False}\}$, and thus *False* "survives".

Contrary to the denotational semantics, the natural semantics does yield the same results as KiCSi for the above expression, as we will show now. To save space we abbreviate $\textbf{case}\ b\ \textbf{of}\ \{\mathsf{True} \to \mathsf{False}; \mathsf{False} \to \mathsf{False}\}$ by $\mathbf{seq}^b_{\mathsf{False}}$.

The following derivation is the only successful derivation for the expression in question:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\emptyset : \mathsf{True} \Downarrow \emptyset : \mathsf{True}}\ (\textsc{Val})}{\emptyset : \mathsf{True} \mathbin{?} \mathbf{seq}^b_{\mathsf{False}} \Downarrow \emptyset : \mathsf{True}}\ (\textsc{Or}_1)}{\{b \mapsto \mathsf{True} \mathbin{?} \mathbf{seq}^b_{\mathsf{False}}\} : b \Downarrow \{b \mapsto \mathsf{True}\} : \mathsf{True}}\ (\textsc{Lookup})}{\emptyset : \mathbf{let}\ b = \mathsf{True} \mathbin{?} \mathbf{seq}^b_{\mathsf{False}}\ \mathbf{in}\ b \Downarrow \{b \mapsto \mathsf{True}\} : \mathsf{True}}\ (\textsc{Let})$$

Crucially, choosing ($\textsc{Or}_2$) instead of ($\textsc{Or}_1$) leads to a partial derivation that cannot be completed:

$$\dfrac{\dfrac{\emptyset : b \Downarrow ??? \qquad ??? \Downarrow ???}{\emptyset : \mathbf{case}\ b\ \mathbf{of}\ \{\mathsf{True} \to \mathsf{False}; \mathsf{False} \to \mathsf{False}\} \Downarrow}\ (???)}{\emptyset : \mathsf{True} \mathbin{?} \mathbf{seq}^b_{\mathsf{False}} \Downarrow}\ (\textsc{Or}_2)$$

For the rule (???) we could try to choose ($\textsc{LSelect}_1$), ($\textsc{LSelect}_2$), ($\textsc{LGuess}_1$) or ($\textsc{LGuess}_2$), but in the left branch we would always end up asking the empty heap for the value of b, thus getting stuck.

The example presented above proves that Conjecture 1 is false (and, more specifically, Conjecture 3). From our current perspective that flaw seems to be unfixable in any approach to a set-valued denotational semantics. To define such a semantics for a recursive let-expression it is simply not sufficient to know the sets which would be assigned to the right-hand sides of variable bindings. Instead, it needs to be known wherefrom the elements in such a set arise. And that information is not accessible in general.

We still think that in the absence of recursive let-expressions Conjecture 3 and (thus, by Theorem 1) Conjecture 1 hold. Also, we think that Conjecture 2 holds even in the presence of recursive let-expressions, though it is doubtful how useful that is in practice, given that we now know that the part of our denotational semantics concerning recursive let-expressions is not really adequate for full Curry. The fragment that remains when we allow only non-recursive let-expressions is still powerful enough to model an interesting part of the language. Hence, our semantics remains a suitable choice for equational reasoning and as a foundation for formally carrying over relational parametricity arguments to functional logic languages.

References

14. Braßel, B., Huch, F.: On the tighter integration of functional and logic programming. Technical Report 0710, Department of Computer Science, University Kiel (2007)
15. Schmidt-Schauß, M., Machkasova, E., Sabel, D.: Counterexamples to simulation in non-deterministic call-by-need lambda-calculi with letrec. Technical Report Frank-38, Institute of Computer Science, University Frankfurt (2009)