assignment08

July 3, 2019

1 Functional Programming SS19

2 Assignment 08 Solutions

2.0.1 Exercise 1 ($\beta\delta - Reduction$)

Recall the notation of "δ-reduction", which is defined as follows:

A set δ of rules of the form $ct_1...t_n \to r$ with $c \in C, t_1, ..., t_n, r \in \Lambda$ is called a delta-rule if

1. *$t_1, ..., t_n, r$ are closed lambda terms*
2. *all t_i are in $\to_\beta$-normal form*
3. *the t_i do not contain any left-hand side of a rule from δ*
4. *in δ there exist no two different rules $ct_1...t_n \to r$ and $ct_1...t_m \to r'$ with $m \geq n$*

For such a set δ we define the relation $\to_\delta$ as the smallest relation with

- *$l \to_\delta r$ for all $l \to r \in \delta$*
- *if $t_1 \to_\delta t_2$ then also $(t_1\ r) \to_\delta (t_2\ r)$, $(r\ t_1) \to_\delta (r\ t_2)$, and $\lambda y.t_1 \to_\delta \lambda y.t_2$ for all $r \in \Lambda, y \in V$.*

We denote the combination of β- and δ-reduction by $\to_{\beta\delta}$, i.e. $\to_{\beta\delta} = \to_\beta \cup \to_\delta$.

The four conditions (1), (2), (3), (4) are required in order to ensure that $\to_{\beta\delta}$ is confluent. Now assume that we replace condition (4) by the following condition:

in δ there exist no two different rules $ct_1...t_n \to r$ and $ct_1...t_m \to r'$ with $m \geq n$ and $r \neq r'$.

Would $\to_{\beta\delta}$ still be confluent? Please explain your answer.

A $\to_{\beta\delta}$ reduction with the modified condition (4) is not confluent in general. In particular, the modified condition allows rules $ct_1...t_n \to r$ and $ct_1...t_m \to r$ with $m \geq n$; depending on the evaluation strategy, this might lead to a terminating reduction sequence when applying one of the rules, but a non-terminating reduction when applying another rule.

To show this, let us consider an example in which we have the following rules in δ:

$$1\text{: c True} \to \lambda x.x$$
$$2\text{: c True False} \to \lambda x.x$$

If we then need to reduce the expression exp $=$ c True $(\lambda y.y\ y)(\lambda y.y\ y))$, the reduction using rule 1 terminates since the resulting term is in weak head normal order:

$$\begin{aligned} &\text{c True } (\lambda y.y\ y)(\lambda y.y\ y) \\ \rightarrow_{\delta_1}\ &(\lambda x.x)(\lambda y.y\ y)(\lambda y.y\ y) \end{aligned}$$

On the other hand, the reduction using rule 2 does not terminate since the term $(\lambda y.y\ \ y)(\lambda y.y\ \ y)$ has to be reduced for the rule to be applied, but the β-reduction on it does not terminate.

The modified condition may however be confluent in some cases. As another example, let us consider the following rules in δ:

$$\begin{aligned} &1\text{: c True} \rightarrow \text{True} \\ &2\text{: c True True} \rightarrow \text{True} \\ &3\text{: True True} \rightarrow \text{True} \end{aligned}$$

If we need to reduce exp $=$ c True True, the reduction rule 1 results in True True, which can then be reduced to True by applying rule 3. Similarly, applying rule 2 on exp results in True. In this case, both rules can be reduced to the same expression.

2.0.2 Exercise 2 (Weak Head Normal Order Reduction)

For each of the following terms please show the reduction steps of the WHNO-reduction with the $\rightarrow_{\beta\delta}$-relation up to weak head normal form. In each step, please indicate whether it is a $\rightarrow_\beta$- or $\rightarrow_\delta$-step. Also indicate if the reduction stops because no more $\beta\delta$-reduction is possible or if it stops only because the term is already in WHNO. Note that `Nil`, `Cons`, and `False` are constructors. Here the set δ contains the following rules:

$$\begin{aligned} \text{if False} &\rightarrow \lambda xy.y \\ \text{gt 22 23} &\rightarrow \text{False} \\ \text{mult 4 2} &\rightarrow 8 \\ \text{isa}_{\text{Cons}}\text{ Nil} &\rightarrow \text{False} \end{aligned}$$

a) $(\lambda x.\textit{mult}\ x\ 2)\ (\textit{if}\ (\textit{gt}\ 22\ 23)\ 1\ 4)$

$$\begin{aligned} (\lambda x.\text{mult } x\ 2)\ (\text{if (gt 22 23) 1 4}) &\rightarrow_\beta \text{mult (if (gt 22 23) 1 4) 2} \\ &\rightarrow_\delta \text{mult (if False 1 4) 2} \\ &\rightarrow_\delta \text{mult } ((\lambda x\ y.y)\ 1\ 4)\ 2 \\ &\rightarrow_\beta \text{mult } ((\lambda x\ y.y)\ 1\ 4)\ 2 \\ &\rightarrow_\beta \text{mult } ((\lambda y.y)\ 4)\ 2 \\ &\rightarrow_\beta \text{mult 4 2} \\ &\rightarrow_\delta 8 \end{aligned}$$

The term is in WHNO at this point, so the reduction stops.

b) $(\lambda x\ y.x\ y)\ ((\lambda z.mult\ 4\ z)\ 2)$

$$(\lambda x\ y.x\ y)\ ((\lambda z.\text{mult}\ 4\ z)\ 2) \rightarrow_\beta \lambda y.((\lambda z.\text{mult}\ 4\ z)\ 2)\ y$$

This term is already in WHNO, so the reduction stops even though additional $\beta\delta$-reduction steps are possible.

c) $(\lambda z.if\ (isa_{Cons}\ z)\ Nil\ (Cons\ ((\lambda x.mult\ 4\ x)\ 2)\ Nil))\ Nil$

$$\begin{aligned}(\lambda z.\text{if}\ (\text{isa}_{Cons}\ z)\ \text{Nil}\ (\text{Cons}\ ((\lambda x.\text{mult}\ 4\ x)\ 2)\ \text{Nil}))\ \text{Nil} &\rightarrow_\beta \text{if}\ (\text{isa}_{Cons}\ \text{Nil})\ \text{Nil}\ (\text{Cons}\ ((\lambda x.\text{mult}\ 4\ x)\ 2)\ \text{Nil})\\ &\rightarrow_\delta \text{if}\ (\text{False})\ \text{Nil}\ (\text{Cons}\ ((\lambda x.\text{mult}\ 4\ x)\ 2)\ \text{Nil})\\ &\rightarrow_\delta (\lambda x\ y.y)\ (\text{Nil}\ (\text{Cons}\ ((\lambda x.\text{mult}\ 4\ x)\ 2)\ \text{Nil}))\\ &\rightarrow_\beta (\lambda y.y)\ (\text{Cons}\ ((\lambda x.\text{mult}\ 4\ x)\ 2)\ \text{Nil})\\ &\rightarrow_\beta \text{Cons}\ ((\lambda x.\text{mult}\ 4\ x)\ 2)\ \text{Nil}\end{aligned}$$

At this point, there are no additional $\beta\delta$-reduction steps possible since Cons is a constructor, so the term is in WHNO.

2.0.3 Exercise 3 (Simple Haskell to Lambda Calculus)

Please translate the following Haskell-expression into a lambda term using $\mathcal{L}am$:

```
let length = \zs -> if isa_Nil zs then
                        0
                        else
                        1 + length (sel_2,2 (argof_Cons zs))
in length Nil
```

Hint: If you want, you can write `(plus x y)` instead of `(x + y)`, `i` instead of `isa_Nil`, etc.

Let exp = \zs -> if isa_Nil zs then 0 else 1 + length (sel_2,2 (argof_Cons zs)). We then have

$$\begin{aligned}&\mathcal{L}am(\text{let length = exp in length Nil})\\ &= \mathcal{L}am(\text{length Nil})[\text{length / fix}(\lambda\ \text{length}.\mathcal{L}am(\text{exp}))]\end{aligned}$$

We have that

$$\begin{aligned}\mathcal{L}am(\text{exp}) &= \mathcal{L}am(\backslash\text{zs -> if isa\_Nil zs then 0 else 1 + length (sel\_2,2 (argof\_Cons zs)))}\\ &= \lambda\ \text{zs}.\mathcal{L}am(\text{if isa\_Nil zs then 0 else 1 + length (sel\_2,2 (argof\_Cons zs)))}\\ &= \lambda\ \text{zs}.(\text{if}\ \mathcal{L}am(\text{isa\_Nil zs})\ \mathcal{L}am(0)\ \mathcal{L}am(\text{1 + length (sel\_2,2 (argof\_Cons zs))}))\\ &= \lambda\ \text{zs}.(\text{if}\ (\mathcal{L}am(\text{isa\_Nil})\ \mathcal{L}am(\text{zs}))\ 0\ \mathcal{L}am(\text{plus 1 length (sel\_2,2 (argof\_Cons zs))}))\\ &= \lambda\ \text{zs}.(\text{if (isa\_Nil zs) 0}\ (\mathcal{L}am(\text{plus 1})\ \mathcal{L}am(\text{length (sel\_2,2 (argof\_Cons zs))})))\\ &= \lambda\ \text{zs}.(\text{if (isa\_Nil zs) 0}\ (\mathcal{L}am(\text{plus})\ \mathcal{L}am(1)\ (\mathcal{L}am(\text{length})\ \mathcal{L}am(\text{sel\_2,2 (argof\_Cons zs)}))))\\ &= \lambda\ \text{zs}.(\text{if (isa\_Nil zs) 0 (plus 1 (length}\ (\mathcal{L}am(\text{sel\_2,2})\ \mathcal{L}am(\text{argof\_Cons zs})))))\\ &= \lambda\ \text{zs}.(\text{if (isa\_Nil zs) 0 (plus 1 (length (sel\_2,2}\ (\mathcal{L}am(\text{argof\_Cons})\ \mathcal{L}am(\text{zs}))))))\\ &= \lambda\ \text{zs}.(\text{if (isa\_Nil zs) 0 (plus 1 (length (sel\_2,2 (argof\_Cons zs)))))}\\ &= \lambda\ \text{zs}.(\text{if isa\_Nil zs 0 (plus 1 (length (sel\_2,2 (argof\_Cons zs)))))}\end{aligned}$$

Therefore,

$\mathcal{L}am$(let length = exp in length Nil)
$=$ $\mathcal{L}am$(length Nil)[length / fix(λ length zs.(if isa_Nil zs 0 (plus 1 (length (sel_2,2 (argof_Cons zs))))))]
$=$ (fix(λ length zs.(if isa_Nil zs 0 (plus 1 (length (sel_2,2 (argof_Cons zs))))))) Nil

Let us denote (fix(λ length zs.(if isa_Nil zs 0 (plus 1 (length (sel_2,2 (argof_Cons zs))))))) by r. Using this notation, we can, for completeness, perform a β-reduction to evaluate the expression:

fix(r) Nil
$\rightarrow_\delta$ r (fix(r)) Nil
$\rightarrow_\beta$ (λ zs.(if isa_Nil zs 0 (plus 1 (fix(r) (sel_2,2 (argof_Cons zs)))))) Nil
$\rightarrow_\beta$ if isa_Nil Nil 0 (plus 1 (fix(r) (sel_2,2 (argof_Cons Nil))))

which evaluates to 0 since if isa_Nil evaluates to true.