lambda and VERTSECT

Johan G. F. Belinfante
2004 January 7

In[1]:= << goedel53.06b; << tools.m

Introduction

The function corresponding to a unary functor \( F[x] \) is \( \text{lambda}[x, F[x]] \). In general, it is faster to use the equivalent construction \( \text{VERTSECT} \text{reify}[x, F[x]] \). Compare, for example:

In[2]:= \( \text{lambda}[x, \text{rotate}[x]] \) // Timing

Out[2]= \{6.88 \text{ Second}, \text{IMAGE[ROT]}\}

In[3]:= \( \text{VERTSECT} \text{reify}[x, \text{rotate}[x]] \) // Timing

Out[3]= \{0.06 \text{ Second}, \text{IMAGE[ROT]}\}

Since \( \text{VERTSECT}[x] \) is always a proper class when \( x \) is a set, the function corresponding to this functor is empty. To remedy this, it is useful to work with the function \( \text{VS} \) that corresponds to the restriction of \( \text{VERTSECT}[x] \) to the domain of \( x \).

In[4]:= \text{image} [\text{VS}, \text{singleton}[x]]

Out[4]= \text{intersection} [\text{image}[\text{V}, \text{singleton}[x]], \text{singleton}[\text{composite}[\text{VERTSECT}[x], \text{id[domain}[x]]]]]

Currently neither \( \text{lambda}[x, \text{composite}[\text{VERTSECT}[x], \text{id[domain}[x]]]] \) or the reification counterpart currently yields a simple expression. In this notebook, new rewrite rules are derived to permit the simulated \( \text{lambda} \) expression to simplify to \( \text{VS} \).

Simulated Lambda Rules

The expression currently obtained by applying \( \text{reify} \) involves the rotation–invariant function \( \text{RIF} \).
This further simplifies the expression to the following:

```
In[13]:= reify[x, composite[VERTSECT[x], id[domain[x]]]]
Out[13]= union[composite[inverse[E], COMPOSE, intersection[composite[inverse[FIRST], VS], composite[inverse[SECOND], IMAGE[DUP]], IMAGE[FIRST]]],
             composite[inverse[SECOND], IMAGE[DUP]], IMAGE[FIRST]]
```

The next step is to identify a part of this as a fancy way to write the empty set. For this, the following general rule can be used:

```
In[11]:= intersection[composite[inverse[FIRST], x], composite[RIGHT[y], z]] // RelnNormality
Out[11]= composite[RIGHT[y], intersection[x, z]]
```

This further simplifies the `reify` expression to the following:

```
In[13]:= reify[x, composite[VERTSECT[x], id[domain[x]]]]
Out[13]= composite[inverse[E], COMPOSE, intersection[composite[inverse[FIRST], VS], composite[inverse[SECOND], IMAGE[DUP]], IMAGE[FIRST]]]
```
The final step needed to simplify this is a rule that says that restricting the restriction \( \text{composite}[\text{VERTSECT}[x], \text{id}[\text{domain}[x]]] \) to \( \text{domain}[x] \) cuts out nothing. This rule can be derived as follows:

\[
\text{In}[14] := \text{Map}[
\text{VERTSECT}, \text{SubstTest}@\text{reify}, x, \text{composite}[y, \text{VERTSECT}[x]], y \rightarrow \text{id}[\text{complement}[\text{singleton}[0]]]])
\]

\[
\text{Out}[14] = \text{composite}[\text{COMPOSE}, \text{intersection}[\text{composite}[\text{inverse}[\text{FIRST}], \text{VS}], \text{composite}[\text{inverse}[\text{SECOND}], \text{IMAGE}[\text{DUP}], \text{IMAGE}[\text{FIRST}]]]] = \text{VS}
\]

\[
\text{In}[15] := \text{composite}[\text{COMPOSE}, \text{intersection}[\text{composite}[\text{inverse}[\text{FIRST}], \text{VS}], \text{composite}[\text{inverse}[\text{SECOND}], \text{IMAGE}[\text{DUP}], \text{IMAGE}[\text{FIRST}]]]] := \text{VS}
\]

This achieves our goal of simplifying the simulated \texttt{lambda} expression.

\[
\text{In}[16] := \text{VERTSECT}@\text{reify}[x, \text{composite}[\text{VERTSECT}[x], \text{id}[\text{domain}[x]]])
\]

\[
\text{Out}[16] = \text{VS}
\]

**lambda for the unrestricted VERTSECT functor**

The function corresponding to \( \text{VERTSECT}[x] \) is empty, just as for the related function \( \text{IMAGE}[x] \), to which it is related. Again, some new rules are needed to see that this is the case. At this stage one finds:

\[
\text{In}[17] := \text{VERTSECT}@\text{reify}[x, \text{VERTSECT}[x]])
\]

\[
\text{Out}[17] = \text{composite}[\text{CUP}, \text{intersection}[\text{composite}[\text{inverse}[\text{FIRST}], \text{VS}], \text{composite}[\text{inverse}[\text{SECOND}], \text{VERTSECT}[\text{composite}[\text{RIGHT}[0], \text{complement}[\text{inverse}[\text{E}]]], \text{IMAGE}[\text{FIRST}]]]]
\]

Part of this expression is the empty set in disguise:

\[
\text{In}[18] := \text{VERTSECT}@\text{composite}[\text{RIGHT}[0], \text{complement}[\text{inverse}[\text{E}]]]) // \text{RelnNormality}
\]

\[
\text{Out}[18] = \text{VERTSECT}@\text{composite}[\text{RIGHT}[0], \text{complement}[\text{inverse}[\text{E}]]]) = 0
\]

\[
\text{In}[19] := \text{VERTSECT}@\text{composite}[\text{RIGHT}[0], \text{complement}[\text{inverse}[\text{E}]]]) := 0
\]

With this rule in place, one does find the expected result:

\[
\text{In}[20] := \text{VERTSECT}@\text{reify}[x, \text{VERTSECT}[x]])
\]

\[
\text{Out}[20] = 0
\]

**appendix**

The rule used above to simplify an expression involving \texttt{RIGHT} has a \texttt{LEFT} counterpart. No application for this is contemplated, but it will also be added as a permanent new rule.

\[
\text{In}[21] := \text{intersection}[\text{composite}[\text{inverse}[\text{SECOND}], x], \text{composite}[\text{LEFT}[y], z]) // \text{RelnNormality}
\]

\[
\text{Out}[21] = \text{intersection}[\text{composite}[\text{inverse}[\text{SECOND}], x], \text{composite}[\text{LEFT}[y], z)] = \text{composite}[\text{LEFT}[y], \text{intersection}[x, z]]
\]
In[22]:= intersection[composite[inverse[SECOND], x_], composite[LEFT[y_], z_]] :=
    composite[LEFT[y], intersection[x, z]]

comment

In general \textsc{VERTSECT[reify[x, f[x]]]} works faster and gives better results than does \texttt{lambda[x, f[x]]} for unary functors. For binary functors, \texttt{lambda[pair[x,y], f[x,y]]} can be replaced with the equivalent construction \texttt{composite[-VERTSECT[reify[x, f[first[x], second[x]]]], id[cart[V, V]]]}. For example:

In[25]:= lambda[pair[x, y], composite[x, y]] // Timing

Out[25]= {14.55 Second, COMPOSE}

In[26]:= composite[VERTSECT[reify[x, composite[first[x], second[x]]]], id[cart[V, V]]] // Timing

Out[26]= {0.48 Second, COMPOSE}