generic derivations for functions

Johan G. F. Belinfante
2004 May 17

```
In[1]:= << goedel57.16a; << tools.m

        :Package Title: goedel57.16a      2004 May 16 at 10:05 p.m.

        It is now:  2004 May 17 at 14:53

        Loading Simplification Rules

        TOOLS.M                       Revised 2004 May 14

        weightlimit = 40
```

summary

In this notebook a general method is presented that can serve as a template for proofs that certain membership rules yield functions. The idea is based on the technique used recently for the specific case of the functon **EQUIV=lambda[x,f[x]]**, but the method should work for any unary constructor **f[x]**. The key new rewrite rule that makes the preent method work is was only recently added to the **GOEDEL** program:

```
In[2]:= member[pair[x, y], composite[Di, z]]

Out[2]= and[member[y, V], not[subclass[image[z, singleton[x]], singleton[y]]]]
```

a generic definition for functions

Fiven a functor **f**, the following membership rule can be used to define the function **L[f] = lambda[x, f[x]]**:

```
In[3]:= member[x_, L[f_]] := and[member[first[x], V], equal[second[x], f[first[x]]]]
```

In practice if may sometimes be useful to replace the first literal on the right side with the more general condition **member[first[x], y]**, which has little effect on the overall procedure. The first step is to derive that fact that **L[f]** is a relation. This can be done in two steps as follows:

```
In[4]:= Map[equal[0, #] &, dif[L[f], cart[V, V]] // Normality]

Out[4]= subclass[L[f], cart[V, V]] == True

In[5]:= subclass[L[f_], cart[V, V]] := True

In[6]:= equal[composite[Id, L[f]], L[f]]

Out[6]= True

In[7]:= composite[Id, L[f_]] := L[f]
```

Tjhe second step is to derive a vertical section rule:

```
In[8]:= image[L[f], singleton[x]] // Normality

Out[8]= image[L[f], singleton[x]] == intersection[image[V, singleton[x]],
          singleton[f[union[x, complement[image[V, singleton[x]]]]]]]

In[9]:= image[L[f_], singleton[x_]] := intersection[image[V, singleton[x]],
          singleton[f[union[x, complement[image[V, singleton[x]]]]]]]
```

Wirh these rules in place one quickly derives the fact that **L[f]** is a function:

```
In[10]:= Map[equal[0, #] &, dif[L[f], funpart[L[f]]] // RelnNormality]

Out[10]= FUNCTION[L[f]] == True

In[11]:= FUNCTION[L[f_]] := True
```

The reification of **f** is the class

```
In[16]:= class[pair[x_, y_], member[y_, f_[x_]]] := R[f]
```

The function **L[f]** can be identified as its **VERTSECT**.

```
In[23]:= L[f] // VSNormality // Reverse

Out[23]= VERTSECT[R[f]] == L[f]

In[24]:= VERTSECT[R[f_]] := L[f]

In[27]:= SubstTest[fix, composite[LB[x], VERTSECT[x]], x → R[f]]

Out[27]= fix[composite[LB[R[f]], L[f]]] == domain[L[f]]

In[28]:= fix[composite[LB[R[f_]], L[f_]]] := domain[L[f]]
```

The domain of the function **L[f]** can be computed as the class of all sets **x** for which **f[x]** is a set:

```
In[29]:= class[x, member[f[x], V]]

Out[29]= domain[L[f]]
```

an example

In this section, the general theory is illustrated with the special case **f = trv**.

```
In[19]:= FUNCTION[L[trv]]

Out[19]= True
```

The reification rules suffice to compute **R[trv]**.

```
In[36]:= reify[x, trv[x]]

Out[36]= composite[inverse[E], HULL[TRV], inverse[S]]
```

This result can be independently verified:

```
In[22]:= SubstTest[class, pair[x, y], member[y, f[x]], f → trv] // Reverse

Out[22]= R[trv] == composite[inverse[E], HULL[TRV], inverse[S]]

In[30]:= R[trv] := composite[inverse[E], HULL[TRV], inverse[S]]

In[32]:= SubstTest[VERTSECT, R[f], f → trv] // Reverse

Out[32]= L[trv] == composite[HULL[TRV], IMAGE[id[cart[V, V]]]]

In[33]:= L[trv] := composite[HULL[TRV], IMAGE[id[cart[V, V]]]]
```

In this case, the domain is already known to be **V**.

```
In[35]:= class[x, member[trv[x], V]] == domain[L[trv]]

Out[35]= True
```