APPLY and rotate[E]

Johan G. F. Belinfante
2003 May 12

```
In[1]:= << goedel52.r67; << tools.m

        :Package Title: goedel52.r67        2003 May 9 at 11:50 p.m.

        It is now:  2003 May 12 at 9:53

        Loading Simplification Rules

        TOOLS.M                     Revised 2003 May 11

        weightlimit = 40
```

■ introduction

The class **APPLY[x,y]** is a shorthand for

```
In[2]:= APPLY[x, y]

Out[2]= A[image[x, singleton[y]]]
```

This constructor is mainly useful for applying a function to an argument. Applying **lambda** to this expression yields a formula involving the rotated membership relation **rotate[E]**. This formula involves the function **BIGCAP** which corresponds to the constructor **A[x]**:

```
In[3]:= class[y, forall[z, implies[member[z, x], member[y, z]]]]

Out[3]= A[x]

In[4]:= lambda[x, A[x]]

Out[4]= BIGCAP
```

For function evaluation, the purpose of **BIGCAP** is to extract the value of a function from the singleton produced by taking the vertical section **image[x,singleton[y]]** of a function **x** at an argument **y**. The unary intersection operation **A** is not the only way to extract an element from its singleton. It is often desirable, when dealing with functions, to replace the function **BIGCAP** with **inverse[SINGLETON]**. In this notebook some formulas are derived which show how to go about doing this. These formula involve the function **FUNPART**:

```
In[5]:= lambda[x, funpart[x]]

Out[5]= FUNPART
```

The constructor **funpart** is defined by

```
In[6]:= intersection[composite[Id, x], complement[composite[Di, x]]]

Out[6]= funpart[x]
```

This constructor is useful for eliminating **FUNCTION** hypotheses from clauses.

```
In[7]:= equal[x, funpart[x]]

Out[7]= FUNCTION[x]
```

One can therefore think of the expression **funpart[x]** as a generic function.

■ applying lambda to APPLY

Applying **lambda** to the constructor **APPLY** yields the binary function

```
In[8]:= lambda[pair[x, y], APPLY[x, y]]

Out[8]= composite[
         VERTSECT[complement[composite[complement[inverse[e]], rotate[e]]]], id[cart[V, V]]]
```

The rotated membership relation in this formula can be eliminated:

```
In[9]:= composite[complement[inverse[E]], BIGCAP, IMG, cross[Id, SINGLETON]] // VSTriNormality //
         Reverse

Out[9]= composite[complement[inverse[e]], rotate[e]] ==
         composite[complement[inverse[e]], BIGCAP, IMG, cross[Id, SINGLETON]]

In[10]:= composite[complement[inverse[e]], rotate[e]] :=
          composite[complement[inverse[e]], BIGCAP, IMG, cross[Id, SINGLETON]]
```

The formula for **lambda APPLY** now produces a more transparent result:

```
In[11]:= lambda[pair[x, y], APPLY[x, y]]

Out[11]= composite[BIGCAP, IMG, cross[Id, SINGLETON]]
```

■ special formulas for function evaluation

For the application to function evaluation, one can replace **BIGCAP** by **inverse[SINGLETON]**. In this section, some formulas are derived which show how to do this.

```
In[12]:= Map[VERTSECT[complement[#]] &, composite[complement[inverse[E]],
            BIGCAP, IMG, cross[FUNPART, SINGLETON]] // TriNormality]

Out[12]= composite[BIGCAP, IMG, cross[FUNPART, SINGLETON]] == union[cart[intersection[
              composite[inverse[SINGLETON], complement[image[inverse[IMG], range[SINGLETON]]]],
              composite[inverse[e], IMAGE[FIRST], FUNPART]], singleton[0]],
            composite[inverse[SINGLETON], IMG, cross[Id, SINGLETON]]]
```

The add−onappearing on the right side is the empty set in disguise:

```
In[13]:= SubstTest[composite, id[range[x]], x, x -> cart[intersection[
              composite[inverse[SINGLETON], complement[image[inverse[IMG], range[SINGLETON]]]],
              composite[inverse[⊕], IMAGE[FIRST], FUNPART]], singleton[0]]] // Reverse

Out[13]= cart[intersection[
            composite[inverse[SINGLETON], complement[image[inverse[IMG], range[SINGLETON]]]],
            composite[inverse[⊕], IMAGE[FIRST], FUNPART]], singleton[0]] == 0

In[14]:= cart[intersection[
            composite[inverse[SINGLETON], complement[image[inverse[IMG], range[SINGLETON]]]],
            composite[inverse[⊕], IMAGE[FIRST], FUNPART]], singleton[0]] := 0
```

The formula derived above now simplifies to

```
In[15]:= %12

Out[15]= composite[BIGCAP, IMG, cross[FUNPART, SINGLETON]] ==
          composite[inverse[SINGLETON], IMG, cross[Id, SINGLETON]]

In[16]:= composite[BIGCAP, IMG, cross[FUNPART, SINGLETON]] :=
          composite[inverse[SINGLETON], IMG, cross[Id, SINGLETON]]
```

Another interesting formula for this function is obtained by a triple rotation:

```
In[17]:= composite[rotate[⊕], cross[FUNPART, Id]] // TripleRotate

Out[17]= composite[rotate[⊕], cross[FUNPART, Id]] ==
          composite[inverse[SINGLETON], IMG, cross[Id, SINGLETON]]

In[18]:= composite[rotate[⊕], cross[FUNPART, Id]] :=
          composite[inverse[SINGLETON], IMG, cross[Id, SINGLETON]]
```