definition of LISTS

Johan G. F. Belinfante
2003 May 14

```
In[1]:= << goedel52.r69; << tools.m

        :Package Title: goedel52.r69         2003 May 11 at 9:30 p.m.

        It is now:  2003 May 16 at 14:12

        Loading Simplification Rules

        TOOLS.M                       Revised 2003 May 14

        weightlimit = 40
```

■ summary

A list is defined here to be a function whose domain is a natural number. The class **LISTS** of all lists is introduced in this notebook, and some of its more elementary properties are derived. It is intended that more interesting properties will be added later. The definition given here is not the only possible way to define a list.. Another possibility would be to treat lists as ordered pairs in which the first coordinate is a list of all but the last element, and the second coordinate is the final item listed. This ordered−pairapproach would make it difficult to define the empty list, which is not a problem with the definition adopted here. Defining lists as functions, as is done here, does have the disadvantage of making it impossible to form lists of proper classes. One could conceivably define lists of proper classes to be disjoint unions, that is, relations whose domains are natural numbers and whose vertical sections are the classes being listed. But defining lists as disjoint unions would make it difficult to form lists in which the empty set is one of the items listed. Moreover, any such disjoint−union list of proper classes would not be a set, and so there would be no such thing as a class of such lists.

■ definition

The definition of the class of all lists is:

```
In[2]:= member[x_, LISTS] := and[FUNCTION[x], member[domain[x], omega]]
```

For work in **Otter**, the membership rule would be deduced from the following definition:

```
In[3]:= LISTS // Normality // Reverse

Out[3]= intersection[FUNS, image[inverse[IMAGE[FIRST]], omega]] == LISTS

In[4]:= intersection[FUNS, image[inverse[IMAGE[FIRST]], omega]] := LISTS
```

Note that the empty set is a list:

```
In[5]:= member[0, LISTS]

Out[5]= True
```

■ some obvious facts

It will be useful to add some rewrite rules expressing some obvious consequences of this definition:

```
In[6]:= subclass[LISTS, FUNS] // AssertTest

Out[6]= subclass[LISTS, FUNS] == True

In[7]:= subclass[LISTS, FUNS] := True
```

The following rule is temporary, and will be improved shortly:

```
In[8]:= SubstTest[subclass, image[x, intersection[y, z]], image[x, z],
            {x -> IMAGE[FIRST], y -> FUNS, z -> image[inverse[IMAGE[FIRST]], omega]}]

Out[8]= subclass[image[IMAGE[FIRST], LISTS], omega] == True

In[9]:= subclass[image[IMAGE[FIRST], LISTS], omega] := True
```

In a later section, the reverse inclusion is derived, and this rule is replaced by an equation.

■ some examples of lists

The length of a list is its domain. The idea is to show that there are lists of any length by explicitly exhibiting one of each length. Obviously, one can get a list of length **n** by listing all the numbers from **0** to **n−1** According to our definition, this list of the first **n** natural numbers is just the identity function **id[n]**. The definition suffices to conclude that these are indeed lists:

```
In[10]:= member[id[x], LISTS]

Out[10]= member[x, omega]
```

The class of all such lists is:

```
In[11]:= class[x, exists[y, and[member[y, omega], equal[x, id[y]]]]]

Out[11]= image[IMAGE[DUP], omega]
```

To show that this is a subclass of **LISTS**, one needs to show it is a subclass of **FUNS** and of **image[inverse[IMAGE[-FIRST]],omega]**. For these facts, there is no need to restrict oneself to natural numbers:

```
In[12]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
             {u -> image[IDP, x], v -> range[IDP], w -> FUNS}]

Out[12]= subclass[image[IMAGE[DUP], x], FUNS] == True

In[13]:= subclass[image[IMAGE[DUP], x_], FUNS] := True

In[14]:= ImageComp[IMAGE[FIRST], IMAGE[DUP], x] // Reverse

Out[14]= image[IMAGE[FIRST], image[IMAGE[DUP], x]] == x
```

```
In[15]:= image[IMAGE[FIRST], image[IMAGE[DUP], x_]] := x
```

The desired conclusion now follows:

```
In[16]:= subclass[image[IMAGE[DUP], omega], LISTS] // AssertTest

Out[16]= subclass[image[IMAGE[DUP], omega], LISTS] == True

In[17]:= subclass[image[IMAGE[DUP], omega], LISTS] := True
```

■ lists can have any length

The examples introduced in the preceding section provide lists of any length.

```
In[18]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
             {u -> image[IMAGE[DUP], omega], v -> LISTS, w -> IMAGE[FIRST]}]

Out[18]= subclass[omega, image[IMAGE[FIRST], LISTS]] == True

In[19]:= subclass[omega, image[IMAGE[FIRST], LISTS]] := True
```

Combining this inclusion with one derived earlier yields an equation:

```
In[20]:= SubstTest[and, subclass[u, v], subclass[v, u],
           {u -> image[IMAGE[FIRST], LISTS], v -> omega}]

Out[20]= True == equal[omega, image[IMAGE[FIRST], LISTS]]

In[21]:= image[IMAGE[FIRST], LISTS] := omega
```