unary operations

Johan G. F. Belinfante
2003 May 7

```
In[1]:= << goedel52.r63; << tools.m

        :Package Title: goedel52.r63         2003 May 6 at 5:50 a.m.

        It is now:  2003 May 12 at 12:22

        Loading Simplification Rules

        TOOLS.M                       Revised 2003 May 11

        weightlimit = 40
```

■ unary operations

A unary operation is a mapping with the same domain and codomain, that is, a function whose range is contained in its domain. Here are two equivalent ways to describe the class of unary operations:

```
In[2]:= class[x, and[FUNCTION[x], subclass[range[x], domain[x]]]]

Out[2]= intersection[FUNS, image[inverse[DORA], inverse[S]]]

In[3]:= class[x, exists[y, member[x, map[y, y]]]]

Out[3]= intersection[FUNS, image[inverse[DORA], inverse[S]]]
```

In this notebook a more compact formula is derived for this class, using the function **MAP**. Why is this important? The success of automated reasoning depends critically on the weights of the expressions encountered. Elegance is not a luxury in automated theorem proving; it is of critical importance.

■ the MAP formula

A quick way to derive the **MAP** formula for the class of unary operations is this:

```
In[4]:= ImageComp[id[FUNS], intersection[composite[inverse[IMAGE[FIRST]], FIRST],
           composite[inverse[IMAGE[SECOND]], inverse[S], SECOND]], Id] // Reverse

Out[4]= intersection[FUNS, image[inverse[DORA], inverse[S]]] == U[image[MAP, Id]]

In[5]:= intersection[FUNS, image[inverse[DORA], inverse[S]]] := U[image[MAP, Id]]
```

More generally:

```
In[6]:= ImageComp[id[FUNS], intersection[composite[inverse[IMAGE[FIRST]], FIRST],
          composite[inverse[IMAGE[SECOND]], inverse[S], SECOND]], x] // Reverse

Out[6]= intersection[FUNS, image[inverse[DORA], composite[inverse[S], x]]] == U[image[MAP, x]]

In[7]:= intersection[FUNS, image[inverse[DORA], composite[inverse[S], x_]]] := U[image[MAP, x]]
```

The formula for the class of unary operations becomes more useful if one adds a corresponding membership rule:

```
In[8]:= SubstTest[member, x, intersection[u, v],
            {u -> FUNS, v -> image[inverse[DORA], composite[inverse[S], y]]}]

Out[8]= member[x, U[image[MAP, y]]] == and[FUNCTION[x], member[x, V],
          member[pair[domain[x], range[x]], composite[inverse[S], y]]]

In[9]:= member[x_, U[image[MAP, y_]]] := and[FUNCTION[x], member[x, V],
          member[pair[domain[x], range[x]], composite[inverse[S], y]]]
```

Specializing to the case of interest:

```
In[10]:= member[x, U[image[MAP, Id]]]

Out[10]= and[FUNCTION[x], member[x, V], subclass[range[x], domain[x]]]
```

The class of unary operations now comes out in this compact form:

```
In[11]:= class[x, and[FUNCTION[x], subclass[range[x], domain[x]]]]

Out[11]= U[image[MAP, Id]]
```

■ an example: id[x]

Identity maps provide examples of unary operations:

```
In[12]:= member[id[x], U[image[MAP, Id]]]

Out[12]= member[x, V]
```

This fact can be rewritten without variables:

```
In[13]:= subclass[P[Id], U[image[MAP, Id]]] // AssertTest

Out[13]= subclass[P[Id], U[image[MAP, Id]]] == True

In[14]:= subclass[P[Id], U[image[MAP, Id]]] := True
```

In particular, the empty set is a unary operation:

```
In[15]:= member[0, U[image[MAP, Id]]]

Out[15]= True
```

■ another example: RC[x]

The relative complement functions **RC[x]** provide another collection of unary operations:

```
In[16]:= member[RC[x], U[image[MAP, Id]]]

Out[16]= True
```

This fact can also be written in variable−free form, using the function **RCF** defined by

```
In[17]:= lambda[x, RC[x]]

Out[17]= RCF
```

The derivation makes use of inverse images, a commonly used technique for this sort of job.

```
In[18]:= image[inverse[RCF], U[image[MAP, Id]]] // Normality

Out[18]= image[inverse[RCF], U[image[MAP, Id]]] == V

In[19]:= image[inverse[RCF], U[image[MAP, Id]]] := V
```

The tool **ImageComp** is used to eliminate the inverse images:

```
In[20]:= ImageComp[RCF, inverse[RCF], U[image[MAP, Id]]]

Out[20]= intersection[range[RCF], U[image[MAP, Id]]] == range[RCF]

In[21]:= intersection[range[RCF], U[image[MAP, Id]]] := range[RCF]
```

This can be rewritten more concisely as follows:

```
In[22]:= SubstTest[equal, intersection[u, v],
           u, {u -> range[RCF], v -> U[image[MAP, Id]]}] // Reverse

Out[22]= subclass[range[RCF], U[image[MAP, Id]]] == True

In[23]:= subclass[range[RCF], U[image[MAP, Id]]] := True
```

■ an example from arithmetic

In this section it is shown how to formulate without variables the fact that the non−negative integers are unary operations.

```
In[24]:= member[plus[x], U[image[MAP, Id]]]

Out[24]= True
```

The class of all such functions is the set **range[PLUS]** of non−negative integers:

```
In[25]:= member[plus[x], range[PLUS]]

Out[25]= member[x, omega]
```

The technique again is to make use of inverse images. (It takes a while to derive the needed rule to do this.)

```
In[26]:= member[x, image[inverse[PLUS], y]] // AssertTest

Out[26]= member[x, image[inverse[PLUS], y]] ==
          and[member[x, omega], member[composite[NATADD, RIGHT[x]], y]]

In[27]:= member[x_, image[inverse[PLUS], y_]] :=
          and[member[x, omega], member[composite[NATADD, RIGHT[x]], y]]
```

From here on, the derivation goes quickly:

```
In[28]:= image[inverse[PLUS], U[image[MAP, Id]]] // Normality

Out[28]= image[inverse[PLUS], U[image[MAP, Id]]] == omega

In[29]:= image[inverse[PLUS], U[image[MAP, Id]]] := omega

In[30]:= ImageComp[PLUS, inverse[PLUS], U[image[MAP, Id]]]

Out[30]= intersection[range[PLUS], U[image[MAP, Id]]] == range[PLUS]

In[31]:= intersection[range[PLUS], U[image[MAP, Id]]] := range[PLUS]

In[32]:= SubstTest[equal, intersection[u, v],
           u, {u -> range[PLUS], v -> U[image[MAP, Id]]}] // Reverse

Out[32]= subclass[range[PLUS], U[image[MAP, Id]]] == True

In[33]:= subclass[range[PLUS], U[image[MAP, Id]]] := True
```

The negative integers are not unary operations:

```
In[34]:= member[inverse[plus[x]], U[image[MAP, Id]]]

Out[34]= or[equal[0, x], not[member[x, omega]]]
```

■ multiplication example

Left−multiplication by a natural number is another unary operation that comes up in the arithmetic of natural numbers. To verify that left multiplication is a unary operation, one needs to add a few simplification rules.

```
In[35]:= member[composite[NATMUL, LEFT[x]], V] // AssertTest

Out[35]= member[composite[NATMUL, LEFT[x]], V] == True

In[36]:= member[composite[NATMUL, LEFT[x_]], V] := True
```

The divisibility relation **DIV** also comes up briefly in this application.

```
In[37]:= class[pair[u, v], exists[w, member[pair[pair[u, w], v], x]]] /. x → NATMUL

Out[37]= DIV
```

Only one elementary fact about this relation is needed for the current endeavor.

```
In[39]:= SubstTest[subclass, image[z, x], range[z], z → DIV]

Out[39]= subclass[image[DIV, x], omega] == True

In[40]:= subclass[image[DIV, x_], omega] := True
```

The desired result now follows:

```
In[41]:= member[composite[NATMUL, LEFT[x]], U[image[MAP, Id]]]

Out[41]= True
```