the class UNOPS

Johan G. F. Belinfante and Lee Martie
2006 June 9

```
In[1]:= SetDirectory["l:"]; << goedel82.09a; << tools.m

        :Package Title: goedel82.09a         2006 June 9 at 6:50 a.m.

        It is now:  2006 Jun 9 at 18:13

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 June 6

        weightlimit = 40
```

summary

The class **UNOPS** of unary operations is defined by a membership rule wrapped with **class**:

```
In[2]:= Begin["Goedel`Private`"];

In[3]:= InfoMatch[class[w_, member[x_, HoldPattern[UNOPS]]]]

Out[3]//TableForm=
        class[w_, member[u_, UNOPS]] := Module[{v = Unique[]}, class[w, exists[v, member[u, ma
```

In this notebook, basic facts and examples concerning this class are derived. There is some overlap with the tutorial notebook **unops.nb**, but several new results are derived here.

normalization

The basic normalization result is:

```
In[4]:= UNOPS // Normality // Reverse

Out[4]= U[image[MAP, Id]] == UNOPS

In[5]:= U[image[MAP, Id]] := UNOPS
```

intersections and inclusions

Theorem.

```
In[6]:= intersection[FUNS, UNOPS] // Renormality
Out[6]= intersection[FUNS, UNOPS] == UNOPS
In[7]:= intersection[FUNS, UNOPS] := UNOPS
```

Corollary.

```
In[8]:= SubstTest[subclass, intersection[x, y], x, {x -> FUNS, y -> UNOPS}]
Out[8]= subclass[UNOPS, FUNS] == True
In[9]:= subclass[UNOPS, FUNS] := True
```

A related result:

```
In[10]:= AssInt[P[cart[V, V]], FUNS, UNOPS]
Out[10]= intersection[UNOPS, P[cart[V, V]]] == UNOPS
In[11]:= intersection[UNOPS, P[cart[V, V]]] := UNOPS
```

a replacement rule

An existing rewrite rule is removed, and will be replaced with a new one.

```
In[12]:= intersection[FUNS, image[inverse[DORA], inverse[S]]] =.
```

The replacement rule is:

```
In[13]:= UNOPS // Renormality // Reverse
Out[13]= intersection[FUNS, image[inverse[DORA], inverse[S]]] == UNOPS
In[14]:= intersection[FUNS, image[inverse[DORA], inverse[S]]] := UNOPS
```

Corollary.

```
In[15]:= SubstTest[subclass, intersection[x, y],
          y, {x -> FUNS, y → image[inverse[DORA], inverse[S]]}]
Out[15]= subclass[image[DORA, UNOPS], inverse[S]] == True
In[16]:= subclass[image[DORA, UNOPS], inverse[S]] := True
```

U[UNOPS]

A general result.

```
In[17]:= ImageComp[inverse[E], composite[inverse[E], MAP], x] // Reverse

Out[17]= U[U[image[MAP, x]]] == composite[inverse[E], x, E]

In[18]:= U[U[image[MAP, x_]]] := composite[inverse[E], x, E]
```

Corollary.

```
In[19]:= SubstTest[U, U[image[MAP, x]], x -> Id]

Out[19]= U[UNOPS] == cart[V, V]

In[20]:= U[UNOPS] := cart[V, V]
```

membership rules

To facilitate reasoning about unary operations, it is desirable to avoid automatically expanding the statement **member[x, UNOPS]**. Instead, the basic facts are presented here as statements of fact. One of these facts is that unary operations are functions:

```
In[21]:= Map[implies[#, FUNCTION[x]] &, SubstTest[member, x, U[image[MAP, y]], y → Id]]

Out[21]= or[FUNCTION[x], not[member[x, UNOPS]]] == True

In[22]:= or[FUNCTION[x_], not[member[x_, UNOPS]]] := True
```

Theorem.

```
In[23]:= Map[implies[#, subclass[range[x], domain[x]]] &,
          SubstTest[member, x, U[image[MAP, y]], y → Id]]

Out[23]= or[not[member[x, UNOPS]], subclass[range[x], domain[x]]] == True

In[24]:= or[not[member[x_, UNOPS]], subclass[range[x_], domain[x_]]] := True
```

Conversely:

```
In[25]:= Map[implies[and[FUNCTION[x], member[x, y], subclass[range[x], domain[x]]], #] &,
          SubstTest[member, x, U[image[MAP, z]], z → Id]]

Out[25]= or[member[x, UNOPS], not[FUNCTION[x]],
           not[member[x, y]], not[subclass[range[x], domain[x]]]] == True

In[26]:= or[member[x_, UNOPS], not[FUNCTION[x_]],
           not[member[x_, y_]], not[subclass[range[x_], domain[x_]]]] := True
```

some examples of unary operations

Some special cases of interest are listed in this section, starting with the empty function.

```
In[27]:= member[0, UNOPS] // AssertTest
Out[27]= member[0, UNOPS] == True

In[28]:= member[0, UNOPS] := True
```

Identity functions are unary operations.

```
In[29]:= member[id[x], UNOPS] // AssertTest
Out[29]= member[id[x], UNOPS] == member[x, V]

In[30]:= member[id[x_], UNOPS] := member[x, V]
```

The relative complement function takes any subset of **x** to its relative complement in **x**.

```
In[31]:= class[pair[u, v], and[subclass[u, x], equal[v, dif[x, u]]]]
Out[31]= RC[x]
```

These functions are unary operations. (When **x** is not a set, the function **RC[x]** is empty.)

```
In[32]:= member[RC[x], UNOPS] // AssertTest
Out[32]= member[RC[x], UNOPS] == True

In[33]:= member[RC[x_], UNOPS] := True
```

projections are unary operations

Projections are idempotent functions:

```
In[34]:= class[x, and[FUNCTION[x], equal[composite[x, x], x]]]
Out[34]= PROJ
```

Lemma.

```
In[35]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p2, p3], not[implies[p1, p3]], {p1 -> member[x, PROJ],
            p2 -> equal[fix[x], range[x]], p3 → subclass[range[x], domain[x]]}]]
Out[35]= or[not[equal[x, composite[x, x]]], not[FUNCTION[x]],
           not[member[x, V]], subclass[range[x], domain[x]]] == True

In[36]:= (% /. {x → x_}) /. Equal -> SetDelayed
```

Lemma.

```
In[37]:= Map[equal[V, #] &, SubstTest[class, x, implies[member[x, u], member[x, v]],
           {u → PROJ, v -> image[inverse[DORA], inverse[S]]}]] // Reverse
```

```
Out[37]= subclass[image[DORA, PROJ], inverse[S]] == True
```

```
In[38]:= subclass[image[DORA, PROJ], inverse[S]] := True
```

```
In[39]:= SubstTest[subclass, u, intersection[v, w],
          {u → PROJ, v -> FUNS, w -> image[inverse[DORA], inverse[S]]}]
```

```
Out[39]= subclass[PROJ, UNOPS] == True
```

```
In[40]:= subclass[PROJ, UNOPS] := True
```

Corollary.

```
In[41]:= equal[intersection[PROJ, UNOPS], PROJ]
```

```
Out[41]= True
```

```
In[42]:= intersection[PROJ, UNOPS] := PROJ
```

In particular, identities are unary operations, as noted before:

```
In[43]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]],
          subclass[u, w], {u → P[Id], v → PROJ, w → UNOPS}]
```

```
Out[43]= subclass[P[Id], UNOPS] == True
```

```
In[44]:= subclass[P[Id], UNOPS] := True
```

involutions are unary operations

Involutions are functions that are their own inverses.

```
In[45]:= class[x, and[FUNCTION[x], equal[x, inverse[x]]]]
```

```
Out[45]= INVOL
```

Lemma: the domain and range of any involution are equal.

```
In[46]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]],
          subclass[u, w], {u → INVOL, v → SYM, w → image[inverse[DORA], Id]}]
```

```
Out[46]= subclass[image[DORA, INVOL], Id] == True
```

```
In[47]:= % /. Equal → SetDelayed
```

In the reverse direction:

```
In[48]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
           {u → P[Id], v → INVOL, w → DORA}]

Out[48]= equal[V, fix[image[DORA, INVOL]]] == True

In[49]:= % /. Equal → SetDelayed
```

Combining these opposite inclusions yields this equation:

```
In[50]:= SubstTest[and, subclass[u, v], subclass[v, u], {u -> image[DORA, INVOL], v -> Id}]

Out[50]= True == equal[Id, image[DORA, INVOL]]

In[51]:= image[DORA, INVOL] := Id
```

As a corollary, it follows that any involution is a unary operation:

```
In[52]:= SubstTest[subclass, w, intersection[x, y],
           {w → INVOL, x → FUNS, y → image[inverse[DORA], inverse[S]]}]

Out[52]= subclass[INVOL, UNOPS] == True

In[53]:= subclass[INVOL, UNOPS] := True
```

In particular, the functions **RC[x]** are involutions, and therefore unary operations, a fact noted in an earlier section.

```
In[54]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
           {u -> range[RCF], v -> INVOL, w → UNOPS}]

Out[54]= subclass[range[RCF], UNOPS] == True

In[55]:= subclass[range[RCF], UNOPS] := True
```