unary operation wrapper

Johan G. F. Belinfante
2009 February 4

```
In[1]:= << l:goedel.09feb03a;<< l:tools.m

        :Package Title: goedel.09feb03a          2009 February 3 at 6:00 p.m.

        It is now:  2009 Feb 4 at 11:25

        Loading Simplification Rules

        TOOLS.M                                  Revised 2009 January 29

        weightlimit = 40
```

summary

A unary operation wrapper **unop[x]** is introduced via an **image[V, -]** rule. As an application it is shown that if the domain of a unary operation is equal to the range of a binary operation, then their composite is a binary operation. Counterexamples are presented to show that one cannot weaken the equality hypothesis to an inclusion in this theorem. Finally, a variable-free equational formulation of the theorem is derived.

definition

The following rewrite rule serves to define the wrapper **unop[x]**.

```
In[2]:= image[V, intersection[unop[x_], set[y_]]] :=
          intersection[image[V, intersection[UNOPS, set[x]]], image[V, intersection[x, set[y]]]]
```

Theorem. Normalization condition for **unop[x]**.

```
In[3]:= Map[fix, SubstTest[reify, y, image[V, intersection[t, set[y]]], t → unop[x]]] // Reverse

Out[3]= intersection[x, image[V, intersection[UNOPS, set[x]]]] == unop[x]

In[4]:= intersection[x_, image[V, intersection[UNOPS, set[x_]]]] := unop[x]
```

Lemma. (Simplification rule.)

```
In[5]:= equiv[or[equal[0, x], member[x, UNOPS]], member[x, UNOPS]]

Out[5]= True

In[6]:= or[equal[0, x_], member[x_, UNOPS]] := member[x, UNOPS]
```

Theorem. (Wrapper removal rule.)

```
In[7]:= SubstTest[equal, x,
          intersection[x, image[V, intersection[t, set[x]]]], t → UNOPS] // Reverse

Out[7]= equal[x, unop[x]] == member[x, UNOPS]

In[8]:= equal[x_, unop[x_]] := member[x, UNOPS]
```

Theorem. Automatic wrapper removal rule.

```
In[9]:= implies[member[x, UNOPS], equal[unop[x], x]]

Out[9]= True

In[10]:= unop[x_] := x /; member[x, UNOPS]
```

Theorem. (Wrapper introduction rule.)

```
In[11]:= SubstTest[member,
           intersection[x, image[V, intersection[t, set[x]]]], t, t → UNOPS] // Reverse

Out[11]= member[unop[x], UNOPS] == True

In[12]:= member[unop[x_], UNOPS] := True
```

reify rule

Theorem.

```
In[13]:= SubstTest[reify, x,
           intersection[f[x], image[V, intersection[t, set[f[x]]]]], t → UNOPS] // Reverse

Out[13]= reify[x, unop[f[x]]] ==
          composite[reify[x, f[x]], id[image[inverse[VERTSECT[reify[x, f[x]]]], UNOPS]]]

In[14]:= reify[x_, unop[y_]] :=
          composite[reify[x, y], id[image[inverse[VERTSECT[reify[x, y]]], UNOPS]]]
```

properties of the unop wrapper

Theorem. The class **unop[x]** is a set.

```
In[15]:= SubstTest[implies, member[t, UNOPS], member[t, V], t → unop[x]] // Reverse

Out[15]= member[unop[x], V] == True

In[16]:= member[unop[x_], V] := True
```

Lemma. Any unary operation is a relation.

```
In[17]:= SubstTest[implies, and[member[x, y], subclass[y, z]],
           member[x, z], {y → UNOPS, z → P[cart[V, V]]}] // Reverse

Out[17]= or[not[member[x, UNOPS]], subclass[x, cart[V, V]]] == True

In[18]:= or[not[member[x_, UNOPS]], subclass[x_, cart[V, V]]] := True
```

Theorem. The class **unop[x]** is a relation.

```
In[19]:= SubstTest[implies, member[t, UNOPS], subclass[t, cart[V, V]], t → unop[x]] // Reverse

Out[19]= subclass[unop[x], cart[V, V]] == True

In[20]:= subclass[unop[x_], cart[V, V]] := True
```

Corollary.

```
In[21]:= equal[composite[Id, unop[x]], unop[x]]

Out[21]= True

In[22]:= composite[Id, unop[x_]] := unop[x]
```

Theorem. The class **unop[x]** is a function.

```
In[23]:= SubstTest[implies, member[t, UNOPS], FUNCTION[t], t → unop[x]] // Reverse

Out[23]= FUNCTION[unop[x]] == True

In[24]:= FUNCTION[unop[x_]] := True
```

Theorem. The range of **unop[x]** is contained in its domain.

```
In[25]:= SubstTest[implies, member[t, UNOPS],
           subclass[range[t], domain[t]], t → unop[x]] // Reverse

Out[25]= subclass[range[unop[x]], domain[unop[x]]] == True

In[26]:= subclass[range[unop[x_]], domain[unop[x_]]] := True
```

variable-free formulations

In this section variable-free reformulations of two theorems from the preceding section are derived.

Theorem. Unary operations are relations.

```
In[27]:= Map[composite[VERTSECT[#], id[UNOPS]] &,
          SubstTest[reify, x, composite[Id, f[x]], f → unop]]

Out[27]= composite[IMAGE[id[cart[V, V]]], id[UNOPS]] == id[UNOPS]

In[28]:= composite[IMAGE[id[cart[V, V]]], id[UNOPS]] := id[UNOPS]
```

Theorem. Unary operations are functions.

```
In[29]:= Assoc[FUNPART, id[FUNS], id[UNOPS]]

Out[29]= composite[FUNPART, id[UNOPS]] == id[UNOPS]

In[30]:= composite[FUNPART, id[UNOPS]] := id[UNOPS]
```

unop and binop

In this section it is shown that if the domain of a unary operation is equal to the range of a binary operation, then their composite is a binary operation.

Lemma. The composite is a set.

```
In[31]:= SubstTest[member, composite[setpart[u], setpart[v]],
           V, {u → unop[x], v → binop[y]}] // Reverse

Out[31]= member[composite[unop[x], binop[y]], V] == True

In[32]:= member[composite[unop[x_], binop[y_]], V] := True
```

Theorem. If the domain of a unary operation is equal to the range of a binary operation, then their composite is a binary operation. Comment. Execution time is dramatically shortened (to under ten seconds), by omitting the following proof step from this derivation: **implies[and[p2, p3, p5, p6], p7]**.

```
In[33]:= Map[not, SubstTest[and, implies[p0, p2], implies[p0, p3],
              implies[and[p0, p1], p4], implies[and[p0, p1], p5], implies[and[p0, p1, p4], p6],
              not[implies[and[p0, p1], p7]], {p0 → equal[t, composite[unop[x], binop[y]]],
               p1 → equal[domain[unop[x]], range[binop[y]]], p2 → FUNCTION[t],
               p3 → member[t, V], p4 → subclass[range[t], range[binop[y]]],
               p5 → equal[domain[t], cartsq[fix[domain[t]]]],
               p6 → subclass[range[t], fix[domain[t]]], p7 → member[t, BINOPS]}] /.
            t -> composite[unop[x], binop[y]]] // Reverse

Out[33]= or[member[composite[unop[x], binop[y]], BINOPS],
           not[equal[domain[unop[x]], range[binop[y]]]]] == True

In[34]:= or[member[composite[unop[x_], binop[y_]], BINOPS],
           not[equal[domain[unop[x_]], range[binop[y_]]]]] := True
```

Corollary. (Wrapper-free restatement of the theorem.)

```
In[35]:= SubstTest[implies,
           and[equal[x, unop[u]], equal[y, binop[v]], equal[domain[x], range[y]]],
           member[composite[x, y], BINOPS], {u → x, v → y}] // Reverse

Out[35]= or[member[composite[x, y], BINOPS], not[equal[domain[x], range[y]]],
           not[member[x, UNOPS]], not[member[y, BINOPS]]] == True

In[36]:= or[member[composite[x_, y_], BINOPS], not[equal[domain[x_], range[y_]]],
           not[member[x_, UNOPS]], not[member[y_, BINOPS]]] := True
```

counterexamples

In this section some counterexamples are presented that show that various plausible weakenings of the hypotheses of the theorem in the preceding section are not possible.

Lemma.

```
In[37]:= member[cart[x, y], UNOPS] // AssertTest

Out[37]= member[cart[x, y], UNOPS] ==
          or[and[member[x, V], member[y, range[SINGLETON]], subclass[y, x]],
           equal[0, x], equal[0, y]]

In[38]:= member[cart[x_, y_], UNOPS] :=
          or[and[member[x, V], member[y, range[SINGLETON]], subclass[y, x]],
           equal[0, x], equal[0, y]]
```

Lemma.

```
In[39]:= SubstTest[member, t, map[cartsq[fix[domain[t]]], fix[domain[t]]],
          t -> cart[cart[set[0], succ[set[0]]], set[0]]]

Out[39]= member[cart[cart[set[0], succ[set[0]]], set[0]], BINOPS] == False

In[40]:= % /. Equal → SetDelayed
```

Counterexample. One cannot simply leave out the **domain** condition in the theorem of the preceding section.

```
In[41]:= member[composite[unop[x], binop[y]], BINOPS] /.
          {x → id[set[0]], y → composite[FIRST, id[cartsq[succ[set[0]]]]]}

Out[41]= False
```

Counterexample. One cannot replace **equal** by **subclass** in the **domain** condition in the theorem of the preceding section.

```
In[42]:= or[member[composite[unop[x], binop[y]], BINOPS],
           not[subclass[domain[unop[x]], range[binop[y]]]]] /.
          {x → id[set[0]], y → composite[FIRST, id[cartsq[succ[set[0]]]]]}

Out[42]= False
```

Counterexample. One cannot replace **equal** by **contains** in the **domain** condition in the theorem of the preceding section.

```
In[43]:= or[member[composite[unop[x], binop[y]], BINOPS],
           not[contains[domain[unop[x]], range[binop[y]]]]] /.
          {x → RC[succ[set[0]]], y → composite[FIRST, id[cartsq[set[0]]]]}

Out[43]= False
```

variable-free restatement

In this final section, a variable-free restatement of the theorem about composites of unary and binary operations is derived.

Lemma. Removing the variables yields an inclusion. In the remainder of this section, this inclusion is strengthened to an equation.

```
In[44]:= Map[empty[composite[Id, complement[#]]] &, SubstTest[class,
           pair[x, y], or[member[composite[x, y], v], not[equal[domain[x], range[y]]],
            not[member[x, u]], not[member[y, v]]], {u → UNOPS, v → BINOPS}]]

Out[44]= subclass[image[COMPOSE, composite[id[BINOPS],
             inverse[IMAGE[SECOND]], IMAGE[FIRST], id[UNOPS]]], BINOPS] == True

In[45]:= % /. Equal → SetDelayed
```

Lemma. Any binary operation can be written as the composite of the identity on its range with itself.

```
In[46]:= SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]],
           {t → COMPOSE, u -> set[PAIR[id[range[binop[x]]], binop[x]]], v ->
             composite[id[BINOPS], inverse[IMAGE[SECOND]], IMAGE[FIRST], id[UNOPS]]}] // Reverse

Out[46]= member[binop[x], image[COMPOSE,
            composite[id[BINOPS], inverse[IMAGE[SECOND]], IMAGE[FIRST], id[UNOPS]]]] == True

In[47]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma. One can use **reify** to remove the **binop** wrapper. This yields an inclusion in the reverse direction.

```
In[48]:= Map[empty[composite[#, id[BINOPS]]] &,
          SubstTest[reify, x, dif[set[binop[x]], t], t -> image[COMPOSE,
             composite[id[BINOPS], inverse[IMAGE[SECOND]], IMAGE[FIRST], id[UNOPS]]]]]

Out[48]= subclass[BINOPS, image[COMPOSE,
            composite[id[BINOPS], inverse[IMAGE[SECOND]], IMAGE[FIRST], id[UNOPS]]]] == True

In[49]:= % /. Equal → SetDelayed
```

Theorem. An equational formulation of the theorem is made into a rewrite rule.

```
In[50]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> image[COMPOSE, composite[id[BINOPS],
              inverse[IMAGE[SECOND]], IMAGE[FIRST], id[UNOPS]]], v -> BINOPS}]

Out[50]= equal[BINOPS, image[COMPOSE,
            composite[id[BINOPS], inverse[IMAGE[SECOND]], IMAGE[FIRST], id[UNOPS]]]] == True

In[51]:= image[COMPOSE,
           composite[id[BINOPS], inverse[IMAGE[SECOND]], IMAGE[FIRST], id[UNOPS]]] := BINOPS
```