unary operation wrapper

Johan G. F. Belinfante
2009 February 4

A unary operation wrapper \texttt{unop[x]} is introduced via an \texttt{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 \texttt{unop[x]}.

\[
\text{In}[2]:= \text{image[V, intersection[unop[x_], set[y_]]]} := \\
\quad \text{intersection[image[V, intersection[UNOPS, set[x]]], image[V, intersection[x, set[y]]]]}
\]

Theorem. Normalization condition for \texttt{unop[x]}.

\[
\text{In}[3]:= \text{Map[fix, SubstTest[reify, y, image[V, intersection[t, set[y]]], t \rightarrow \text{unop[x]}}]\] // Reverse
\]

Lemma. (Simplification rule.)

\[
\text{In}[5]:= \text{equiv[or[equal[0, x], member[x, UNOPS]], member[x, UNOPS]]}
\]

\[
\text{Out}[5]= \text{True}
\]

\[
\text{In}[6]:= \text{or[equal[0, x_], member[x_, UNOPS]]} := \text{member[x, UNOPS]}
\]
Theorem. (Wrapper removal rule.)

\[\text{In[7]} := \text{SubstTest[equal, x,}
  \text{intersection[x, image[V, intersection[t, set[x]]]], t \rightarrow UNOPS] // Reverse}\]

\[\text{Out[7]} = \text{equal[x, unop[x]] = member[x, UNOPS]}\]

\[\text{In[8]} := \text{equal[x_, unop[x_]] := member[x, UNOPS]}\]

Theorem. Automatic wrapper removal rule.

\[\text{In[9]} := \text{implies[member[x, UNOPS], equal[unop[x], x]]}\]

\[\text{Out[9]} = \text{True}\]

\[\text{In[10]} := \text{unop[x_] := x /; member[x, UNOPS]}\]

Theorem. (Wrapper introduction rule.)

\[\text{In[11]} := \text{SubstTest[member,}
  \text{intersection[x, image[V, intersection[t, set[x]]]], t, t \rightarrow UNOPS] // Reverse}\]

\[\text{Out[11]} = \text{member[unop[x], UNOPS] = True}\]

\[\text{In[12]} := \text{member[unop[x_], UNOPS] := True}\]

---

**reify rule**

Theorem.

\[\text{In[13]} := \text{SubstTest[reify, x,}
  \text{intersection[f[x], image[V, intersection[t, set[f[x]]]], t \rightarrow UNOPS] // Reverse}\]

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

\[\text{In[14]} := \text{reify[x_, unop[y_]] :=}
  \text{composite[reify[x, y], id[image[inverse[VERTSECT[reify[x, y]]], UNOPS]]]}\]

---

**properties of the unop wrapper**

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

\[\text{In[15]} := \text{SubstTest[implies, member[t, UNOPS], member[t, V], t \rightarrow unop[x]] // Reverse}\]

\[\text{Out[15]} = \text{member[unop[x], V] = True}\]

\[\text{In[16]} := \text{member[unop[x_], V] := True}\]
Lemma. Any unary operation is a relation.

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

\[\text{Out[17]= or\{not\{member[x, \text{UNOPS}\], subclass[x, \text{cart\[V, V]\}\]\]} = True}\]

\[\text{In[18]:= or\{not\{member[x_, \text{UNOPS}\], subclass[x_, \text{cart\[V, V]\]}\]\} := True}\]

Theorem. The class \text{unop\[x\]} is a relation.

\[\text{In[19]:= SubstTest[implies, member[t, \text{UNOPS}], subclass[t, \text{cart\[V, V]\}], t \rightarrow \text{unop\[x\]}] // Reverse}\]

\[\text{Out[19]= subclass[unop\[x\], \text{cart\[V, V]\}] = True}\]

\[\text{In[20]= subclass[unop\[x\], \text{cart\[V, V]\}] :: True}\]

Corollary.

\[\text{In[21]:= equal[composite\[\text{Id}, \text{unop\[x\]}\], \text{unop\[x\]}]}\]

\[\text{Out[21]= True}\]

\[\text{In[22]= composite[\text{Id}, \text{unop\[x\]}] :: \text{unop\[x\]}}\]

Theorem. The class \text{unop\[x\]} is a function.

\[\text{In[23]:= SubstTest[implies, member[t, \text{UNOPS}], \text{FUNCTION\[t\]}, t \rightarrow \text{unop\[x\]}] // Reverse}\]

\[\text{Out[23]= \text{FUNCTION[unop\[x\]]} = True}\]

\[\text{In[24]= \text{FUNCTION[unop\[x\]]} :: True}\]

Theorem. The range of \text{unop\[x\]} is contained in its domain.

\[\text{In[25]:= SubstTest[implies, member[t, \text{UNOPS}], subclass[range[t], domain[t]], t \rightarrow \text{unop\[x\]}] // Reverse}\]

\[\text{Out[25]= subclass[range[unop\[x\]], domain[unop\[x\]]] = True}\]

\[\text{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], 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.)
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.

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

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

```plaintext
In[43]:= or[member[composite[unop[x]], binop[y]], BINOPS],
      not[contains[domain[unop[x]], range[binop[y]]]] /. 
     {x \rightarrow RC[succ[set[0]]], y \rightarrow 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.

```plaintext
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 \rightarrow UNOPS, v \rightarrow BINOPS}]
Out[44]= subclass[image[COMPOSE, composite[id[BINOPS],
      inverse[IMAGE[SECOND]], IMAGE[FIRST], id[UNOPS]]], BINOPS] = True
```

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

```plaintext
In[45]:= % /. Equal\rightarrow SetDelayed
```

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

```plaintext
In[46]:= SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]],
   {t \rightarrow COMPOSE, u \rightarrow set[PAIR[id[range[binop[x]]], binop[x]]], v \rightarrow 
     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 \rightarrow x_) /. Equal\rightarrow 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