transforms of reflexive relations

Johan G. F. Belinfante
2006 May 2

```
In[1]:= SetDirectory["l:"]; << goedel81.01a; << tools.m

        :Package Title: goedel81.01a          2006 May 1 at 10:10 p.m.

        It is now:  2006 May 2 at 11:37

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 March 7

        weightlimit = 40
```

summary

Rewrite rules for transforms of reflexive relations are derived.

general fix formulas for transforms

Lemma.

```
In[2]:= fix[composite[x, inverse[y], inverse[x]]] // InvertFixTest
Out[2]= fix[composite[x, inverse[y], inverse[x]]] == fix[composite[x, y, inverse[x]]]

In[3]:= fix[composite[x_, inverse[y_], inverse[x_]]] := fix[composite[x, y, inverse[x]]]

In[4]:= fix[composite[inverse[x], inverse[y], x]] // InvertFixTest
Out[4]= fix[composite[inverse[x], inverse[y], x]] == fix[composite[inverse[x], y, x]]

In[5]:= fix[composite[inverse[x_], inverse[y_], x_]] := fix[composite[inverse[x], y, x]]
```

fix formula for reflexive relations

Observation.

```
In[6]:= abstract[y, fix[composite[x, y, inverse[x]]]]
Out[6]= intersection[composite[x, FIRST], composite[x, SECOND]]
```

Using this observation, one derives:

```
In[7]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
         {u → y, v → cart[fix[y], fix[y]],
          w -> intersection[composite[x, FIRST], composite[x, SECOND]]}]

Out[7]= or[not[REFLEXIVE[y]],
          subclass[fix[composite[x, y, inverse[x]]], image[x, fix[y]]]] == True

In[8]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The inclusion can be strengthened to an equation:

```
In[9]:= SubstTest[and, implies[p, subclass[u, v]], subclass[v, u], {p → REFLEXIVE[y],
           u -> fix[composite[x, y, inverse[x]]], v -> image[x, fix[y]]}] // Reverse

Out[9]= or[equal[fix[composite[x, y, inverse[x]]], image[x, fix[y]]], not[REFLEXIVE[y]]] == True

In[10]:= or[equal[fix[composite[x_, y_, inverse[x_]]], image[x_, fix[y_]]],
            not[REFLEXIVE[y_]]] := True
```

The **REFLEXIVE** literal can be eliminated by introducing the **rfx** wrapper. Note that the **rfx** wrapper was partly removed due to a rewrite rule for **fix[rfx[x]]**.

```
In[11]:= SubstTest[implies, REFLEXIVE[z],
          equal[fix[composite[x, z, inverse[x]]], image[x, fix[z]]], z → rfx[y]]

Out[11]= equal[fix[composite[x, rfx[y], inverse[x]]], image[x, fix[y]]] == True

In[12]:= fix[composite[x_, rfx[y_], inverse[x_]]] := image[x, fix[y]]
```

Corollary.

```
In[13]:= SubstTest[fix, composite[z, rfx[y], inverse[z]], z → inverse[x]]

Out[13]= fix[composite[inverse[x], rfx[y], x]] == image[inverse[x], fix[y]]

In[14]:= fix[composite[inverse[x_], rfx[y_], x_]] := image[inverse[x], fix[y]]
```

A curious variable-free version of the theorem can be derived:

```
In[15]:= Map[VERTSECT,
            SubstTest[reify, y, fix[composite[x, f[y], inverse[x]]], f → rfx]] // Reverse

Out[15]= composite[IMAGE[intersection[composite[x, FIRST], composite[x, SECOND]]], CORE[RFX]] ==
          composite[IMAGE[x], IMAGE[inverse[DUP]]]

In[16]:= composite[IMAGE[intersection[composite[x_, FIRST], composite[x_, SECOND]]],
            CORE[RFX]] := composite[IMAGE[x], IMAGE[inverse[DUP]]]
```

corollaries

```
In[17]:= SubstTest[fix, composite[x, rfx[z], inverse[x]], z → eqv[y]]
Out[17]= fix[composite[x, eqv[y], inverse[x]]] == image[x, fix[eqv[y]]]

In[18]:= fix[composite[x_, eqv[y_], inverse[x_]]] := image[x, fix[eqv[y]]]

In[19]:= SubstTest[fix, composite[x, rfx[z], inverse[x]], z → po[y]]
Out[19]= fix[composite[x, po[y], inverse[x]]] == image[x, fix[po[y]]]

In[20]:= fix[composite[x_, po[y_], inverse[x_]]] := image[x, fix[po[y]]]

In[21]:= SubstTest[fix, composite[x, rfx[z], inverse[x]], z → to[y]]
Out[21]= fix[composite[x, to[y], inverse[x]]] == image[x, fix[to[y]]]

In[22]:= fix[composite[x_, to[y_], inverse[x_]]] := image[x, fix[to[y]]]

In[23]:= SubstTest[fix, composite[x, rfx[z], inverse[x]], z → wo[y]]
Out[23]= fix[composite[x, wo[y], inverse[x]]] == image[x, fix[wo[y]]]

In[24]:= fix[composite[x_, wo[y_], inverse[x_]]] := image[x, fix[wo[y]]]
```

reflexivity is preserved by transforms

Transforms of reflexive relations are reflexive.

```
In[25]:= REFLEXIVE[composite[x, rfx[y], inverse[x]]] // AssertTest
Out[25]= REFLEXIVE[composite[x, rfx[y], inverse[x]]] == True

In[26]:= REFLEXIVE[composite[x_, rfx[y_], inverse[x_]]] := True
```

Corollary.

```
In[27]:= SubstTest[REFLEXIVE, composite[z, rfx[y], inverse[z]], z → inverse[x]]
Out[27]= REFLEXIVE[composite[inverse[x], rfx[y], x]] == True

In[28]:= REFLEXIVE[composite[inverse[x_], rfx[y_], x_]] := True
```

The following rules are replacements for more special rules that will be removed.

```
In[29]:= SubstTest[REFLEXIVE, inverse[z], z -> composite[inverse[x], inverse[y], x]] // Reverse
Out[29]= REFLEXIVE[composite[inverse[x], inverse[y], x]] ==
          REFLEXIVE[composite[inverse[x], y, x]]
```

```
In[30]:= REFLEXIVE[composite[inverse[x_], inverse[y_], x_]] :=
          REFLEXIVE[composite[inverse[x], y, x]]

In[31]:= SubstTest[REFLEXIVE, inverse[z], z -> composite[x, inverse[y], inverse[x]]] // Reverse

Out[31]= REFLEXIVE[composite[x, inverse[y], inverse[x]]] ==
          REFLEXIVE[composite[x, y, inverse[x]]]

In[32]:= REFLEXIVE[composite[x_, inverse[y_], inverse[x_]]] :=
          REFLEXIVE[composite[x, y, inverse[x]]]
```

---

a variable-free equation

The **rfx** wrapper can be removed:

```
In[33]:= SubstTest[implies, equal[y, rfx[z]], REFLEXIVE[composite[x, y, inverse[x]]], z → y]

Out[33]= or[not[REFLEXIVE[y]], REFLEXIVE[composite[x, y, inverse[x]]]] == True

In[34]:= or[not[REFLEXIVE[y_]], REFLEXIVE[composite[x_, y_, inverse[x_]]]] := True
```

The following technical lemma prepares for eliminating the variables.

```
In[35]:= implies[and[member[x, V], member[y, RFX]],
           member[composite[x, y, inverse[x]], RFX]] // NotNotTest

Out[35]= or[and[member[composite[x, y, inverse[x]], V], REFLEXIVE[composite[x, y, inverse[x]]]],
           not[member[x, V]], not[member[y, V]], not[REFLEXIVE[y]]] == True

In[36]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The variable **y** is eliminated as follows.

```
In[37]:= (Map[equal[V, #] &, SubstTest[class, y, implies[and[member[w, V], member[y, z]],
               member[composite[w, y, inverse[w]], z]], z → RFX]] // Reverse) /. w → setpart[x]

Out[37]= subclass[image[IMAGE[cross[setpart[x], setpart[x]]], RFX], RFX] == True

In[38]:= subclass[image[IMAGE[cross[setpart[x_], setpart[x_]]], RFX], RFX] := True
```

One can eliminate the variable **x** using **reify**.

```
In[39]:= Map[equal[0, #] &, SubstTest[reify, x,
           dif[image[IMAGE[cross[setpart[x], setpart[x]]], z], z], z → RFX]] // Reverse

Out[39]= subclass[image[IMG, cart[image[CROSS, Id], RFX]], RFX] == True

In[40]:= % /. Equal → SetDelayed
```

The key to obtaining a reverse inclusion is the observation that every reflexive relation is a restriction of itself. Restriction is a special case of transformation. This fact is captured in the following rewrite rule.

```
In[41]:= image[CROSS, id[P[Id]]]

Out[41]= image[IMAGE[DUP], image[CART, Id]]
```

The following function is needed to exploit this observation.

```
In[42]:= abstract[x, image[IMG, cart[image[CROSS, x], RFX]]]

Out[42]= composite[IMG, id[cart[V, RFX]], inverse[FIRST], CROSS]
```

The reverse inclusion is obtained using this function as follows:

```
In[43]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
           {u → id[P[Id]], v → Id, w -> composite[IMG, id[cart[V, RFX]], inverse[FIRST], CROSS]}]

Out[43]= subclass[RFX, image[IMG, cart[image[CROSS, Id], RFX]]] == True

In[44]:= % /. Equal → SetDelayed
```

Combining the two inclusions yields an equation:

```
In[45]:= SubstTest[and, subclass[u, v], subclass[v, u],
            {u -> image[IMG, cart[image[CROSS, Id], RFX]], v -> RFX}] // Reverse

Out[45]= equal[RFX, image[IMG, cart[image[CROSS, Id], RFX]]] == True

In[46]:= image[IMG, cart[image[CROSS, Id], RFX]] := RFX
```

---

serendipity

The fact that restriction is a special case of transformation also yields this inclusion:

```
In[47]:= SubstTest[implies, subclass[u, v],
           subclass[image[w, u], image[w, v]], {u → id[P[Id]], v → Id, w → CROSS}]

Out[47]= subclass[image[IMAGE[DUP], image[CART, Id]], image[CROSS, Id]] == True

In[48]:= subclass[image[IMAGE[DUP], image[CART, Id]], image[CROSS, Id]] := True
```