compound wrappers for thin equivalence relations

Johan G. F. Belinfante
2004 September 30

```
In[1]:= SetDirectory["i:"]; << goedel61.29a; << tools.m

        :Package Title: goedel61.29a          2004 September 29 at 8:20 p.m.

        It is now:  2004 Sep 30 at 18:39

        Loading Simplification Rules

        TOOLS.M                      Revised 2004 September 25

        weightlimit = 40
```

summary

In this notebook it is shown that both **eqv[thinpart[x]]** and **thinpart[eqv[x]]** can serve as generic thin equivalence relations. These results are analogous to facts about thin transitive relations. In a few cases one can derive the **eqv** result from the corresponding one for **trv**, but additional considerations are generally required.

basic result

When the **thinpart** wrapper is on the inside, one needs to show that thinness holds:

```
In[2]:= SubstTest[implies, and[subclass[u, v], thin[v]],
         thin[u], {u → eqv[thinpart[x]], v → trv[thinpart[x]]}]

Out[2]= equal[V, domain[VERTSECT[eqv[thinpart[x]]]]] == True

In[3]:= domain[VERTSECT[eqv[thinpart[x_]]]] := V
```

Lemma.

```
In[4]:= SubstTest[implies, and[equal[x, y], EQUIVALENCE[x]],
         equal[x, eqv[y]], y → thinpart[x]]

Out[4]= or[equal[x, eqv[thinpart[x]]], not[equal[V, domain[VERTSECT[x]]]],
          not[EQUIVALENCE[x]], not[subclass[x, cart[V, V]]]] == True

In[5]:= (% /. x → x_) /. Equal → SetDelayed
```

This can be cleaned up:

```
In[6]:= Map[not, SubstTest[and, implies[and[p1, p2, p3], p4], implies[p1, p3],
          not[implies[and[p1, p2], p4]], {p1 → EQUIVALENCE[x], p2 → thin[x],
           p3 → subclass[x, cart[V, V]], p4 → equal[x, eqv[thinpart[x]]]}]]

Out[6]= or[equal[x, eqv[thinpart[x]]],
          not[equal[V, domain[VERTSECT[x]]]], not[EQUIVALENCE[x]]] == True

In[7]:= (% /. x → x_) /. Equal → SetDelayed
```

Corollary.

```
In[8]:= equiv[equal[x, eqv[thinpart[x]]], and[EQUIVALENCE[x], thin[x]]]

Out[8]= True
```

This justifies the rewrite rule:

```
In[9]:= equal[x_, eqv[thinpart[x_]]] :=
         and[equal[V, domain[VERTSECT[x]]], EQUIVALENCE[x]]
```

---

restrictions of equivalence relations

In this section it is shown that restrictions of equivalence relations to invariant subclasses are equivalence relations. The following basic computation needed for this takes ten seconds with the **simplify** flag off versus thirty seconds with that flag on.

```
In[10]:= simplify = False;

In[11]:= EQUIVALENCE[composite[eqv[x], id[y]]] // AssertTest

Out[11]= EQUIVALENCE[composite[eqv[x], id[y]]] == subclass[image[eqv[x], y], y]

In[12]:= EQUIVALENCE[composite[eqv[x_], id[y_]]] := subclass[image[eqv[x], y], y]
```

In particular:

```
In[13]:= SubstTest[EQUIVALENCE, composite[eqv[x], id[y]], y → domain[VERTSECT[eqv[x]]]]

Out[13]= EQUIVALENCE[thinpart[eqv[x]]] == True

In[14]:= EQUIVALENCE[thinpart[eqv[x_]]] := True
```

This result can also be expressed without wrappers:

```
In[15]:= SubstTest[implies, equal[y, eqv[x]], EQUIVALENCE[thinpart[y]], y → x]
Out[15]= or[EQUIVALENCE[thinpart[x]], not[EQUIVALENCE[x]]] == True

In[16]:= or[EQUIVALENCE[thinpart[x_]], not[EQUIVALENCE[x_]]] := True
```

Corollaries.

```
In[17]:= SubstTest[inverse, eqv[y], y -> thinpart[eqv[x]]]
Out[17]= inverse[thinpart[eqv[x]]] == thinpart[eqv[x]]

In[18]:= inverse[thinpart[eqv[x_]]] := thinpart[eqv[x]]

In[19]:= composite[id[domain[VERTSECT[eqv[x]]]], eqv[x]] // DoubleInverse
Out[19]= composite[id[domain[VERTSECT[eqv[x]]]], eqv[x]] == thinpart[eqv[x]]

In[20]:= composite[id[domain[VERTSECT[eqv[x_]]]], eqv[x_]] := thinpart[eqv[x]]
```

The following rule for equivalence relations is the special case $\mathbf{y = V}$.

```
In[21]:= SubstTest[implies, EQUIVALENCE[x], EQUIVALENCE[restrict[x, y, y]], y → V]
Out[21]= or[EQUIVALENCE[composite[Id, x]], not[EQUIVALENCE[x]]] == True

In[22]:= or[EQUIVALENCE[composite[Id, x_]], not[EQUIVALENCE[x_]]] := True
```

---

wrappers in the reverse order

In this section a generic thin equivalence relation is constructed with the wrappers in the reverse order of what was considered earlier.

```
In[23]:= SubstTest[implies, and[equal[y, eqv[x]], equal[z, thinpart[y]]],
          equal[z, thinpart[eqv[x]]], {y → x, z → x}]
Out[23]= or[equal[x, thinpart[eqv[x]]], not[equal[V, domain[VERTSECT[x]]]],
           not[EQUIVALENCE[x]], not[subclass[x, cart[V, V]]]] == True

In[24]:= (% /. x → x_) /. Equal → SetDelayed
```

This can be simplified:

```
In[25]:= Map[not, SubstTest[and, implies[and[p1, p2, p3], p4],
           implies[p1, p3], not[implies[and[p1, p2], p4]],
           {p1 → EQUIVALENCE[x], p2 → thin[x], p3 → subclass[x, cart[V, V]],
            p4 → equal[x, thinpart[eqv[x]]]}]]

Out[25]= or[equal[x, thinpart[eqv[x]]],
           not[equal[V, domain[VERTSECT[x]]]], not[EQUIVALENCE[x]]] == True

In[26]:= (% /. x → x_) /. Equal → SetDelayed
```

Corollary.

```
In[27]:= equiv[equal[x, thinpart[eqv[x]]], and[thin[x], EQUIVALENCE[x]]] // not // not

Out[27]= True
```

This justifies the following rewrite rule.

```
In[28]:= equal[x_, thinpart[eqv[x_]]] :=
          and[equal[V, domain[VERTSECT[x]]], EQUIVALENCE[x]]
```

---

a thinness result

If **x** is thin, then so is **eqv[x]**.

```
In[29]:= SubstTest[implies, and[subclass[u, v], thin[v]],
          thin[u], {u → eqv[x], v → trv[x]}]

Out[29]= or[equal[V, domain[VERTSECT[eqv[x]]]],
           not[equal[V, domain[VERTSECT[x]]]]] == True

In[30]:= or[equal[V, domain[VERTSECT[eqv[x_]]]],
           not[equal[V, domain[VERTSECT[x_]]]]] := True
```

---

variable-free formulas

The class of all **y** such that **composite[x, id[y]]** belongs to **z** is **image[inverse[IMAGE[composite[id[x], inverse[FIRST]]]], z]**.

```
In[31]:= member[y, image[inverse[IMAGE[composite[id[x], inverse[FIRST]]]], z]]

Out[31]= and[member[y, V], member[composite[x, id[y]], z]]
```

The result about restrictions of equivalence relations to invariant subclasses can therefore be recast as follows:

```
In[32]:= image[inverse[IMAGE[composite[id[eqv[x]], inverse[FIRST]]]], EQV] // Normality

Out[32]= image[inverse[IMAGE[composite[id[eqv[x]], inverse[FIRST]]]], EQV] ==
          invar[eqv[x]]

In[33]:= image[inverse[IMAGE[composite[id[eqv[x_]], inverse[FIRST]]]], EQV] :=
          invar[eqv[x]]
```

The remaining variable can be removed by reification:

```
In[34]:= SubstTest[reify, x,
           image[inverse[f[composite[id[eqv[x]], inverse[FIRST]]]], EQV],
           f → IMAGE] // Reverse

Out[34]= composite[image[inverse[IMG], EQV],
           IMAGE[composite[id[inverse[FIRST]], inverse[SECOND]]],
           EQUIV] == composite[INVAR, EQUIV]

In[35]:= composite[image[inverse[IMG], EQV],
           IMAGE[composite[id[inverse[FIRST]], inverse[SECOND]]],
           EQUIV] := composite[INVAR, EQUIV]
```

---

restrictions of transitive relations

For transitive relations, the invariance requirement is not needed. All restrictions of transitive relations are transitive.

```
In[36]:= TRANSITIVE[composite[trv[x], id[y]]] // AssertTest

Out[36]= TRANSITIVE[composite[trv[x], id[y]]] == True

In[37]:= TRANSITIVE[composite[trv[x_], id[y_]]] := True
```

Similarly:

```
In[38]:= SubstTest[implies, and[TRANSITIVE[u], TRANSITIVE[v]],
          TRANSITIVE[intersection[u, v]], {u → trv[y], v → cart[V, x]}]

Out[38]= TRANSITIVE[composite[id[x], trv[y]]] == True

In[39]:= TRANSITIVE[composite[id[x_], trv[y_]]] := True
```

a variable-free formula

One can derive a variable-free formula for transitive relations, analogous to the one for equivalence relations.

```
In[40]:= image[inverse[IMAGE[composite[id[trv[x]], inverse[FIRST]]]], TRV] // Normality

Out[40]= image[inverse[IMAGE[composite[id[trv[x]], inverse[FIRST]]]], TRV] ==
          P[domain[VERTSECT[trv[x]]]]

In[41]:= image[inverse[IMAGE[composite[id[trv[x_]], inverse[FIRST]]]], TRV] :=
          P[domain[VERTSECT[trv[x]]]]
```

Lemma.

```
In[42]:= LB[domain[VERTSECT[rotate[composite[S, inverse[HULL[TRV]], E]]]]] //
          RelnNormality

Out[42]= LB[domain[VERTSECT[rotate[composite[S, inverse[HULL[TRV]], E]]]]] == cart[V, V]

In[43]:= LB[domain[VERTSECT[rotate[composite[S, inverse[HULL[TRV]], E]]]]] :=
          cart[V, V]
```

A variable-free formula for **TRV** analogous to the one derived for **EQV**.

```
In[44]:= SubstTest[reify, x,
           image[inverse[f[composite[id[trv[x]], inverse[FIRST]]]], TRV],
           f → IMAGE] // Reverse

Out[44]= composite[image[inverse[IMG], TRV],
           IMAGE[composite[id[inverse[FIRST]], inverse[SECOND]]],
           HULL[TRV], IMAGE[id[cart[V, V]]]] == cart[V, V]

In[45]:= composite[image[inverse[IMG], TRV],
           IMAGE[composite[id[inverse[FIRST]], inverse[SECOND]]],
           HULL[TRV], IMAGE[id[cart[V, V]]]] := cart[V, V]
```

another variable-free formulation

Another way to formulate the results about restrictions of transitive relations is in terms of the constructor **RS[x]**.

```
In[46]:= subclass[RS[trv[x]], TRV] // AssertTest

Out[46]= subclass[RS[trv[x]], TRV] == True

In[47]:= subclass[RS[trv[x_]], TRV] := True
```

Removing the wrappers yields:

```
In[48]:= SubstTest[implies, equal[y, trv[x]], subclass[RS[y], TRV], y → x]

Out[48]= or[not[TRANSITIVE[x]], subclass[RS[x], TRV]] == True

In[49]:= or[not[TRANSITIVE[x_]], subclass[RS[x_], TRV]] := True
```

A variable-free result can be derived as follows:

```
In[50]:= Map[equal[V, #] &, SubstTest[class, x,
            implies[member[x, z], subclass[RS[x], z]], z → TRV]] // Reverse

Out[50]= subclass[image[COMPOSE, cart[TRV, P[Id]]], TRV] == True

In[51]:= subclass[image[COMPOSE, cart[TRV, P[Id]]], TRV] := True
```