the class image[VS,EQV]

Johan G. F. Belinfante
2006 October 19

```
In[1]:= SetDirectory["l:"]; << goedel86.18a; << tools.m

:Package Title: goedel86.18a          2006 October 18 at 1:45 a.m.

It is now:  2006 Oct 19 at 9:14

Loading Simplification Rules

TOOLS.M                      Revised 2006 October 12

weightlimit = 40
```

summary

In this notebook it is shown that a (small) function is a canonical projection of an equivalence relation if and only if it can be written in the form **composite[id[x], E]** for some set **x**.

canonical projections of equivalences

If **q** is an equivalence relation, the function **VERTSECT[q]** maps any set **x** to its vertical section **image[q, set[x]]**. This vertical section is the equivalence class of **x**, provided **x** belongs to the domain of **q**.

```
In[2]:= class[pair[x, y], equal[y, image[q, set[x]]]]

Out[2]= VERTSECT[q]
```

When **q** is a set, the restriction of the function **VERTSECT[q]** to the domain of **q** is also a set. The function **VS** takes **q** to this restriction. Comment: The combination **VERTSECT[reify[q, ...]]** is equivalent to **lambda[q, ...]**, but works more efficiently.

```
In[3]:= VERTSECT[reify[q, composite[VERTSECT[q], id[domain[q]]]]]

Out[3]= VS
```

Thus the canonical projection for a (small) equivalence relation **eqv[setpart[x]]** can be obtained by applying the function **VS**.

```
In[4]:= APPLY[VS, eqv[setpart[x]]]

Out[4]= composite[VERTSECT[eqv[setpart[x]]], id[fix[eqv[setpart[x]]]]]
```

The canonical projection for a (small) equivalence relation **q** is the function **f = APPLY[VS, q]**. The class **image[VS, EQV]** is therefore the class of all canonical projections of (small) equalence relations. One can recover an equivalence relation **q** from its canonical projection **f** by the formula **q = composite[inverse[f], f]**. A variable-free version of this fact is:

```
In[5]:= Map[range, SubstTest[reify, x,
          image[composite[COMPOSE, intersection[composite[inverse[FIRST], INVERSE],
            composite[inverse[SECOND], IMAGE[id[cart[V, V]]]]]],
           VS, w], set[setpart[x]]], w → EQUIV]] // Reverse

Out[5]= image[COMPOSE, composite[id[image[VS, EQV]], INVERSE]] == EQV

In[6]:= image[COMPOSE, composite[id[image[VS, EQV]], INVERSE]] := EQV
```

co-restrictions

The class of restrictions of any class **x** is:

```
In[7]:= class[z, exists[y, equal[z, composite[x, id[y]]]]]

Out[7]= RS[x]
```

The class of co-restrictions of **x** is the class of inverse of restrictions of the inverse of **x**.

```
In[8]:= class[z, assert[exists[y, equal[z, composite[id[y], x]]]]]

Out[8]= intersection[invar[composite[id[x], inverse[SECOND], SECOND]],
         P[composite[id[domain[VERTSECT[inverse[x]]]], x]]]
```

Another formula for this class will be derived in this section. It is analogous to the following result for restrictions:

```
In[9]:= range[IMAGE[composite[id[x], inverse[FIRST]]]]

Out[9]= RS[x]
```

Lemma. A temporary rewrite rule.

```
In[10]:= (image[INVERSE, RS[inverse[x]]] // Normality // Reverse)

Out[10]= intersection[invar[composite[id[x], inverse[SECOND], SECOND]],
           P[composite[id[domain[VERTSECT[inverse[x]]]], x]]] == image[INVERSE, RS[inverse[x]]]

In[11]:= intersection[invar[composite[id[x_], inverse[SECOND], SECOND]],
           P[composite[id[domain[VERTSECT[inverse[x_]]]], x_]]] :=
          image[INVERSE, RS[inverse[x]]]
```

Theorem. A formula for the class of co-restrictions of **x**.

```
In[12]:= (range[IMAGE[composite[id[x], inverse[SECOND]]]] // Renormality)

Out[12]= range[IMAGE[composite[id[x], inverse[SECOND]]]] == image[INVERSE, RS[inverse[x]]]
```

```
In[13]:= range[IMAGE[composite[id[x_], inverse[SECOND]]]] := image[INVERSE, RS[inverse[x]]]
```

canonical projections of equivalences

Canonical projections are functions, but not every function is the canonical projection of an equivalence relation. Canonical projections of thin equivalence relations are of the form **composite[id[x], E]**, where **x** is the set of equivalence classes. To derive a variable-free formulation of this result, the **thinpart** wrapper is first replaced with a **setpart** wrapper.

```
In[14]:= SubstTest[composite, inverse[E],
          id[range[composite[VERTSECT[eqv[thinpart[y]]], id[fix[eqv[thinpart[y]]]]]]],
          y → setpart[x]]

Out[14]= composite[inverse[E], id[image[VERTSECT[eqv[setpart[x]]], fix[eqv[setpart[x]]]]]] ==
          composite[id[fix[eqv[setpart[x]]]], inverse[VERTSECT[eqv[setpart[x]]]]]

In[15]:= composite[inverse[E], id[image[VERTSECT[eqv[setpart[x_]]], fix[eqv[setpart[x_]]]]]] :=
          composite[id[fix[eqv[setpart[x]]]], inverse[VERTSECT[eqv[setpart[x]]]]]
```

Next the **eqv** and **setpart** wrappers are removed:

```
In[16]:= SubstTest[implies, equal[x, eqv[setpart[y]]],
          member[x, image[inverse[VS], image[inverse[IMAGE[SWAP]], RS[inverse[E]]]]], y → x]

Out[16]= or[and[equal[composite[id[domain[x]], inverse[VERTSECT[x]]],
             composite[inverse[E], id[image[VERTSECT[x], domain[x]]]]],
            member[domain[thinpart[x]], V], member[image[VERTSECT[x], domain[x]], V]],
           not[EQUIVALENCE[x]], not[member[x, V]]] == True

In[17]:= (% /. x → x_) /. Equal → SetDelayed
```

The variable **x** can now be eliminated as follows:

```
In[18]:= Map[equal[V, #] &, SubstTest[class, x, implies[member[x, u], member[x, v]], {u → EQV,
             v → image[inverse[VS], image[inverse[IMAGE[SWAP]], RS[inverse[E]]]]}]] // Reverse

Out[18]= subclass[image[IMAGE[SWAP], image[VS, EQV]], RS[inverse[E]]] == True

In[19]:= % /. Equal → SetDelayed
```

Lemma.

```
In[20]:= ImageComp[INVERSE, IMAGE[SWAP], x] // Reverse

Out[20]= image[INVERSE, image[IMAGE[SWAP], x]] == image[IMAGE[id[cart[V, V]]], x]

In[21]:= image[INVERSE, image[IMAGE[SWAP], x_]] := image[IMAGE[id[cart[V, V]]], x]
```

Lemma.

```
In[22]:= ImageComp[IMAGE[id[cart[V, V]]], VS, x] // Reverse

Out[22]= image[IMAGE[id[cart[V, V]]], image[VS, x]] == image[VS, x]

In[23]:= image[IMAGE[id[cart[V, V]]], image[VS, x_]] := image[VS, x]
```

One can now reformulate this property of canonical projections as follows:

```
In[24]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u -> image[IMAGE[SWAP], image[VS, EQV]], v -> RS[inverse[E]], w → INVERSE}]

Out[24]= subclass[image[VS, EQV], image[INVERSE, RS[inverse[E]]]] == True

In[25]:= subclass[image[VS, EQV], image[INVERSE, RS[inverse[E]]]] := True
```

It will be shown in this notebook that the reverse inclusion also holds. In other words, canonical projections of (small) equivalences are characterized by the property of being co-restrictions of **E**.

an observation

The class of all functions of the form **APPLY[VS, x]** is the class of functions whose range does not hold the empty set.

```
In[29]:= range[VS]

Out[29]= intersection[FUNS, P[cart[V, complement[set[0]]]]]
```

This observation can be made more explicit: If the range of a function **y** does not hold the empty set, then **y** can be written as **composite[VERTSECT[z], id[domain[z]]]** for the relation **z** = **composite[inverse[E], y]**.

```
In[26]:= implies[and[equal[y, funpart[w]],
           not[member[0, range[y]]], equal[z, composite[inverse[E], y]]],
          equal[y, composite[VERTSECT[z], id[domain[z]]]]]

Out[26]= True
```

The removal of the **funpart** wrapper yields:

```
In[27]:= SubstTest[implies, and[equal[y, funpart[w]],
           not[member[0, range[y]]], equal[z, composite[inverse[E], y]]],
          equal[y, composite[VERTSECT[z], id[domain[z]]]], w → y]

Out[27]= or[equal[y, composite[VERTSECT[z], id[domain[z]]]], member[0, range[y]],
           not[equal[z, composite[inverse[E], y]]], not[FUNCTION[y]]] == True

In[28]:= or[equal[y_, composite[VERTSECT[z_], id[domain[z_]]]], member[0, range[y_]],
           not[equal[z_, composite[inverse[E], y_]]], not[FUNCTION[y_]]] := True
```

For functions of the form **composite[id[x], E]**, the class **x** must be pairwise disjoint.

```
In[30]:= FUNCTION[composite[id[x], E]]

Out[30]= subclass[cart[x, x], union[DISJOINT, Id]]
```

Specializing the characterization of **VERTSECT** functions to those of the form **composite[id[x], E]** yields this corollary:

```
In[31]:= SubstTest[or, equal[y, composite[VERTSECT[z], id[domain[z]]]], member[0, range[y]],
          not[equal[z, composite[inverse[E], y]]], not[FUNCTION[y]], y → composite[id[x], E]]

Out[31]= or[equal[composite[id[x], E], composite[VERTSECT[z], id[domain[z]]]],
           not[equal[z, composite[inverse[E], id[x], E]]],
           not[subclass[cart[x, x], union[DISJOINT, Id]]]] == True

In[32]:= (% /. {x → x_, z → z_}) /. Equal → SetDelayed
```

A variable-free formulation of the following statement will now be derived:

```
In[33]:= implies[and[FUNCTION[z], equal[z, composite[id[x], E]],
           equal[y, composite[inverse[z], z]]], equal[z, composite[VERTSECT[y], id[domain[y]]]]]

Out[33]= True
```

There are three variables to be eliminated. One can eliminate the variable **x** by simply rewriting the statement as follows:

```
In[34]:= implies[and[FUNCTION[z], member[z, image[INVERSE, RS[inverse[E]]]],
           equal[y, composite[inverse[z], z]]], equal[z, composite[VERTSECT[y], id[domain[y]]]]]

Out[34]= True
```

Lemma.

```
In[35]:= (member[pair[x, y], composite[w, id[INVERSE], inverse[SECOND]]] // AssertTest) /.
          w → COMPOSE

Out[35]= member[pair[x, y], composite[COMPOSE, id[INVERSE], inverse[SECOND]]] ==
          and[equal[y, composite[inverse[x], x]],
           member[x, V], member[y, V], subclass[x, cart[V, V]]]

In[36]:= member[pair[x_, y_], composite[COMPOSE, id[INVERSE], inverse[SECOND]]] :=
          and[equal[y, composite[inverse[x], x]],
           member[x, V], member[y, V], subclass[x, cart[V, V]]]
```

The remaining variables can now be eliminated as follows.

```
In[37]:= Map[empty[composite[Id, complement[#]]] &, SubstTest[class, pair[x, y], implies[
             and[member[x, u], member[pair[x, setpart[y]], v]], member[pair[setpart[y], x], w]],
            {u -> intersection[FUNS, image[INVERSE, RS[inverse[E]]]],
             v -> composite[COMPOSE, id[INVERSE], inverse[SECOND]], w → VS}]] // Reverse

Out[37]= subclass[intersection[FUNS, image[INVERSE, RS[inverse[E]]]],
           fix[composite[VS, COMPOSE, id[INVERSE], inverse[SECOND]]]] == True

In[38]:= % /. Equal → SetDelayed
```

Lemma.

```
In[39]:= SubstTest[fix, composite[id[u], v], {u → FUNS, v → composite[VS, x]}] // Reverse

Out[39]= intersection[FUNS, fix[composite[VS, x]]] == fix[composite[VS, x]]

In[40]:= intersection[FUNS, fix[composite[VS, x_]]] := fix[composite[VS, x]]
```

Lemma.

```
In[41]:= SubstTest[subclass, fix[z], range[z],
           z → composite[VS, COMPOSE, id[INVERSE], inverse[s], id[FUNS]]] /. s → SECOND

Out[41]= subclass[fix[composite[VS, COMPOSE, id[INVERSE], inverse[SECOND]]], image[VS, EQV]] ==
          True

In[42]:= % /. Equal → SetDelayed
```

Theorem. Every (small) function which is a co-restriction of **E** is the canonical projection of some equivalence relation.

```
In[43]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
          {u -> intersection[FUNS, image[INVERSE, RS[inverse[E]]]],
           v -> fix[composite[VS, COMPOSE, id[INVERSE], inverse[SECOND]]], w -> image[VS, EQV]}]

Out[43]= subclass[intersection[FUNS, image[INVERSE, RS[inverse[E]]]], image[VS, EQV]] == True

In[44]:= % /. Equal → SetDelayed
```

Combining this result with the reverse inclusion yields an equation.

```
In[45]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> intersection[FUNS, image[INVERSE, RS[inverse[E]]]], v -> image[VS, EQV]}]

Out[45]= True == equal[image[VS, EQV], intersection[FUNS, image[INVERSE, RS[inverse[E]]]]]

In[46]:= intersection[FUNS, image[INVERSE, RS[inverse[E]]]] := image[VS, EQV]
```

comment

The class of co-restrictions of the membership relation **E** that appears in the formula for **image[VS,EQV]** can also be written as a certain subclass of the power class **P[E]** as follows:

```
In[47]:= image[INVERSE, RS[inverse[E]]] // Normality // Reverse

Out[47]= intersection[invar[composite[id[E], inverse[SECOND], SECOND]], P[E]] ==
          image[INVERSE, RS[inverse[E]]]

In[48]:= intersection[invar[composite[id[E], inverse[SECOND], SECOND]], P[E]] :=
          image[INVERSE, RS[inverse[E]]]
```