quotients for a generic thin equivalence relation

Johan G. F. Belinfante
2002 December 17

```
<< goedel52.q64; << tools.m

:Package Title: goedel52.q64          2002 December 14 at 10:20 p.m

It is now:  2002 Dec 17 at 7:35

Loading Simplification Rules

TOOLS.M                       Revised 2002 November 30

weightlimit = 40
```

■ summary

Specific equivalence relations, like **EQUIDIFF**, can be fairly complicated, which slows down derivations of rewrite rules whenever one needs to use the **Normality** test, or one of its variants. A more efficient approach is to develop a body of facts about equivalence relations in general and then to apply this theory to specific cases.

This notebook contains derivations of rewrite rules for a thin generic equivalence relation **q** that can serve as a template for specific equivalence relations. An alternative would be to develop a body of theorems about equivalence relations and use these to deduce formulas for specific cases. There are both advantages and disadvantages of a generic development over a theoretical development. Using a generic template requires no reasoning to be done, which can substantially shorten the time needed to do applications.

■ generic equivalence relation

A thin generic equivalence relation **q** will be introduced which satisfies the following conditions:

```
and[EQUIVALENCE[q], thin[q]]

and[equal[q, inverse[q]], equal[V, domain[VERTSECT[q]]], subclass[composite[q, q], q]]
```

These conditions will be added as separate rules:

```
inverse[q] := q

subclass[composite[q, q], q] := True

domain[VERTSECT[q]] := V
```

■ a corollary

```
SubstTest[inverse, inverse[x], x -> q] // Reverse
```

```
composite[Id, q] == q
```

```
composite[Id, q] := q
```

■ domain, range and fix

```
SubstTest[implies, subclass[u, v], subclass[fix[u], fix[v]],
    {u -> composite[inverse[x], x], v -> x}] /. x -> q
```

```
subclass[domain[q], fix[q]] == True
```

```
subclass[domain[q], fix[q]] := True
```

```
SubstTest[and, subclass[u, v], subclass[v, u], {u -> fix[q], v -> domain[q]}]
```

```
True == equal[domain[q], fix[q]]
```

```
domain[q] := fix[q]
```

```
SubstTest[range, inverse[x], x -> q]
```

```
range[q] == fix[q]
```

```
range[q] := fix[q]
```

■ sharpening the transitive property

```
SubstTest[implies, subclass[u, v], subclass[composite[w, u], composite[w, v]],
    {u -> id[fix[q]], v -> q, w -> q}]
```

```
subclass[q, composite[q, q]] == True
```

```
subclass[q, composite[q, q]] := True
```

```
equal[composite[q, q], q] // AssertTest
```

```
equal[q, composite[q, q]] == True
```

```
composite[q, q] := q
```

■ rules for complement[q]

Some rules for complements are needed in the derivation of the fact that equivalence classes are mutually disjoint.

```
SubstTest[implies, subclass[u, v], subclass[composite[w, u], composite[w, v]],
 {u → id[fix[q]], v → q, w → complement[q]}]
```

```
subclass[composite[complement[q], id[fix[q]]], composite[complement[q], q]] == True
```

```
subclass[composite[complement[q], id[fix[q]]], composite[complement[q], q]] := True
```

```
SubstTest[and, subclass[u, v],
  subclass[v, u], {u -> composite[complement[q], id[fix[q]]],
   v -> composite[complement[q], q]}] // Reverse
```

```
equal[composite[complement[q], q], composite[complement[q], id[fix[q]]]] == True
```

```
composite[complement[q], q] := composite[complement[q], id[fix[q]]]
```

```
composite[q, complement[q]] // DoubleInverse
```

```
composite[q, complement[q]] == composite[id[fix[q]], complement[q]]
```

```
composite[q, complement[q]] := composite[id[fix[q]], complement[q]]
```

■ quotient set

The **quotient class** of an equivalence relation is the class of all its equivalence classes. Any definition for the quotient class will also apply to relations in general, not just to equivalence relations. This raises the question whether one should use **domain[x]** or **fix[x]** in these general fomulas. For applications to equivalence relations, of course, it makes no difference what one does. The choice made here is tentative, and in fact is introduced only to provide a shorthand to help clarify what is going on.

```
quot[x_] := image[VERTSECT[x], fix[x]]
```

The following result is not limited to equivalence relations:

```
IminComp[GREATEST[x], S, V] // Reverse
```

```
image[inverse[S], domain[GREATEST[x]]] == image[inverse[UB[x]], fix[x]]
```

```
image[inverse[S], domain[GREATEST[x_]]] := image[inverse[UB[x]], fix[x]]
```

■ cliques

For non−thinequivalence relations such as the equipollence relation, the equivalence classes may be proper classes. In this case, one must replace **quot[q]** with the larger class **cliques[q]**. The connection between the two is needed in deriving the canonical factorization of a thin equvalence relation.

```
SubstTest[subclass, domain[UB[x]], cliques[composite[inverse[x], x]], x → q]
```

```
subclass[domain[UB[q]], cliques[q]] == True
```

```
subclass[domain[UB[q]], cliques[q]] := True
```

```
equal[domain[UB[q]], cliques[q]] // AssertTest
```

```
equal[cliques[q], domain[UB[q]]] == True
```

```
domain[UB[q]] := cliques[q]
```

```
Map[domain, SubstTest[LB, inverse[x], x -> q]]
```

```
image[inverse[S], range[VERTSECT[q]]] == cliques[q]
```

```
image[inverse[S], range[VERTSECT[q]]] := cliques[q]
```

■ UB[q]

The upper bound function symbol **UB** and the lower bound function symbol **LB** play key roles in the derivations. It is unclear a−priorihow best to orient some of the rewrite rules derived, and the results obtained appear to be rather sensitive to the choice of orientation.

```
SubstTest[LB, inverse[x], x -> q]
```

```
composite[inverse[VERTSECT[q]], S] == UB[q]
```

```
composite[inverse[VERTSECT[q]], S] := UB[q]
```

The following is only temporary; the right side will be rewritten in terms of **cliques[q]** later on.

```
IminComp[inverse[VERTSECT[q]], S, fix[q]] // Reverse
```

```
image[inverse[S], image[VERTSECT[q], fix[q]]] == image[inverse[UB[q]], fix[q]]
```

```
image[inverse[S], image[VERTSECT[q], fix[q]]] := image[inverse[UB[q]], fix[q]]
```

■ GREATEST[q]

```
SubstTest[implies, subclass[u, v], subclass[domain[u], domain[v]],
    {u -> GREATEST[x], v -> UB[x]}] /. x -> q
```

```
subclass[domain[GREATEST[q]], cliques[q]] == True
```

```
subclass[domain[GREATEST[q]], cliques[q]] := True
```

```
equal[composite[inverse[E], id[cliques[q]]], GREATEST[q]] // AssertTest
```

```
equal[composite[inverse[E], id[cliques[q]]], GREATEST[q]] == True
```

```
composite[inverse[E], id[cliques[q]]] := GREATEST[q]
```

The following factorization is weaker than the canonical factorization, and holds also for equivalence relations that are not thin, such as the equipollence relation. This factorization will be used below in the derivation of the canonical factorization.

```
q == composite[inverse[E], id[cliques[q]], E]
```

```
True
```

```
IminComp[inverse[E], id[cliques[q]], V]
```

```
domain[GREATEST[q]] == intersection[cliques[q], complement[singleton[0]]]
```

```
domain[GREATEST[q]] := intersection[cliques[q], complement[singleton[0]]]
```

■ rules for IMAGE[q] and VERTSECT[q]

The canonical projection (or natural map) is the function that takes each element of **fix[q]** to its equivalence class. This function is a restriction of the function **VERTSECT[q]** , which in turn can be related to the function **IMAGE[q]**. The latter is idempotent:

```
Map[VERTSECT, Assoc[q, q, inverse[E]]]
```

```
composite[IMAGE[q], IMAGE[q]] == IMAGE[q]
```

```
composite[IMAGE[q], IMAGE[q]] := IMAGE[q]
```

```
Map[fix, Assoc[IMAGE[q], IMAGE[q], inverse[IMAGE[q]]]] // Reverse
```

```
range[IMAGE[q]] == fix[IMAGE[q]]
```

```
range[IMAGE[q]] := fix[IMAGE[q]]
```

```
Assoc[IMAGE[q], IMAGE[q], SINGLETON]
```

```
composite[IMAGE[q], VERTSECT[q]] == VERTSECT[q]
```

```
composite[IMAGE[q], VERTSECT[q]] := VERTSECT[q]
```

■ a formula for range[VERTSECT[q]]

It is shown in this dection that the quotient class **quot[q]** differs from **range[VERTSECT[q]]** by at most one element, namely the empty set. The empty set never belongs to **quot[q]**. Whether it belongs to **range[VERTSECT[q]]** depends on whether **q** is empty or not.

```
member[0, quot[q]]
```

```
False
```

The following result holds in general:

```
ImageComp[VERTSECT[x], id[complement[domain[x]]], V] // Reverse
```

```
image[VERTSECT[x], complement[domain[x]]] ==
 intersection[image[V, complement[domain[x]]], singleton[0]]
```

```
image[VERTSECT[x], complement[domain[x_]]] :=
 intersection[image[V, complement[domain[x]]], singleton[0]]
```

This is the main result:

```
(SubstTest[image, VERTSECT[x], union[y, z],
    {y -> domain[x], z -> complement[domain[x]]}] /. x -> q)

range[VERTSECT[q]] == union[image[VERTSECT[q], fix[q]],
  intersection[image[V, complement[fix[q]]], singleton[0]]]
```

The following corollaries follow from the main result.

```
Map[subclass[#, union[image[VERTSECT[q], fix[q]], singleton[0]]] &, %]

subclass[range[VERTSECT[q]], union[image[VERTSECT[q], fix[q]], singleton[0]]] == True

subclass[range[VERTSECT[q]], union[image[VERTSECT[q], fix[q]], singleton[0]]] := True

equal[intersection[complement[singleton[0]], image[VERTSECT[q], fix[q]]],
image[VERTSECT[q], fix[q]]]

True

intersection[complement[singleton[0]], image[VERTSECT[q], fix[q]]] :=
 image[VERTSECT[q], fix[q]]

equal[intersection[complement[singleton[0]], range[VERTSECT[q]]],
  image[VERTSECT[q], fix[q]]] // AssertTest

equal[image[VERTSECT[q], fix[q]],
  intersection[complement[singleton[0]], range[VERTSECT[q]]]] == True

intersection[complement[singleton[0]], range[VERTSECT[q]]] := image[VERTSECT[q], fix[q]]
```

■ Uclosure formula

In this section it is shown that **fix[IMAGE[q]]** equals **Uclosure[quot[q]]**.

```
SubstTest[implies, subclass[u, v], subclass[Uclosure[u], Uclosure[v]],
    {u -> range[VERTSECT[q]], v -> union[image[VERTSECT[q], fix[q]], singleton[0]]}]

subclass[Uclosure[range[VERTSECT[q]]], Uclosure[image[VERTSECT[q], fix[q]]]] == True

subclass[Uclosure[range[VERTSECT[q]]], Uclosure[image[VERTSECT[q], fix[q]]]] := True

SubstTest[implies, subclass[u, v], subclass[Uclosure[u], Uclosure[v]],
    {u -> image[VERTSECT[q], fix[q]], v -> range[VERTSECT[q]]}]

subclass[Uclosure[image[VERTSECT[q], fix[q]]], Uclosure[range[VERTSECT[q]]]] == True

subclass[Uclosure[image[VERTSECT[q], fix[q]]], Uclosure[range[VERTSECT[q]]]] := True

SubstTest[and, subclass[u, v], subclass[v, u],
    {u -> Uclosure[image[VERTSECT[q], fix[q]]], v -> Uclosure[range[VERTSECT[q]]]}]

True == equal[Uclosure[image[VERTSECT[q], fix[q]]], Uclosure[range[VERTSECT[q]]]]
```

```
Uclosure[range[VERTSECT[q]]] := Uclosure[image[VERTSECT[q], fix[q]]]

ImageComp[BIGCUP, IMAGE[VERTSECT[q]], V] // Reverse

image[BIGCUP, range[IMAGE[VERTSECT[q]]]] == fix[IMAGE[q]]

image[BIGCUP, range[IMAGE[VERTSECT[q]]]] := fix[IMAGE[q]]

SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
    {u -> range[IMAGE[VERTSECT[q]]], v -> P[range[VERTSECT[q]]], w -> BIGCUP}]

subclass[fix[IMAGE[q]], Uclosure[image[VERTSECT[q], fix[q]]]] == True

subclass[fix[IMAGE[q]], Uclosure[image[VERTSECT[q], fix[q]]]] := True

SubstTest[implies, subclass[u, v], subclass[Uclosure[u], Uclosure[v]],
    {u -> range[VERTSECT[x]], v -> range[IMAGE[x]]}] /. x -> q

subclass[Uclosure[image[VERTSECT[q], fix[q]]], fix[IMAGE[q]]] == True

subclass[Uclosure[image[VERTSECT[q], fix[q]]], fix[IMAGE[q]]] := True

SubstTest[and, subclass[u, v], subclass[v, u],
    {u -> Uclosure[image[VERTSECT[q], fix[q]]], v -> fix[IMAGE[q]]}]

True == equal[fix[IMAGE[q]], Uclosure[image[VERTSECT[q], fix[q]]]]
```

It is not clear how to orient this equation. The following is only tentative:

```
fix[IMAGE[q]] := Uclosure[image[VERTSECT[q], fix[q]]]
```

■ a canonical factorization

```
composite[inverse[E], id[union[x, singleton[0]]], E] // VSNormality

composite[inverse[E], id[union[x, singleton[0]]], E] == composite[inverse[E], id[x], E]

composite[inverse[E], id[union[x_, singleton[0]]], E] := composite[inverse[E], id[x], E]

SubstTest[composite, inverse[E],
  id[image[inverse[S], x]], E, x -> range[VERTSECT[q]]] // Reverse

composite[inverse[E], id[range[VERTSECT[q]]], E] == q

composite[inverse[E], id[range[VERTSECT[q]]], E] := q
```

This is the canonical factorization:

```
SubstTest[composite, inverse[E],
 id[intersection[complement[singleton[0]], x]], E, x -> range[VERTSECT[q]]]

composite[inverse[E], id[image[VERTSECT[q], fix[q]]], E] == q

composite[inverse[E], id[image[VERTSECT[q], fix[q]]], E] := q
```

This is the canonical factorization of a thin equivalence relation:

```
composite[inverse[E], id[quot[q]], E]

q
```

■ another canonical factorization of q

```
composite[inverse[VERTSECT[q]], VERTSECT[q]] // RelnNormality

composite[inverse[VERTSECT[q]], VERTSECT[q]] ==
 union[q, cart[complement[fix[q]], complement[fix[q]]]]

composite[inverse[VERTSECT[q]], VERTSECT[q]] :=
 union[q, cart[complement[fix[q]], complement[fix[q]]]]
```

Observation: any thin equivalence relation can be written as the reversed composite of a function with its inverse; the function being the canonical projection:

```
composite[id[fix[q]], inverse[VERTSECT[q]], VERTSECT[q], id[fix[q]]]

q

composite[inverse[VERTSECT[q]], Di, VERTSECT[q]] // DoubleComplement

composite[inverse[VERTSECT[q]], Di, VERTSECT[q]] ==
 union[composite[complement[q], id[fix[q]]], composite[id[fix[q]], complement[q]]]

composite[inverse[VERTSECT[q]], Di, VERTSECT[q]] :=
 union[composite[complement[q], id[fix[q]]], composite[id[fix[q]], complement[q]]]
```

■ DISJOINT

Replacing the diversity relation **Di** with the relation **DISJOINT** yields a similar formula:

```
composite[inverse[VERTSECT[q]], DISJOINT, VERTSECT[q]] // DoubleComplement

composite[inverse[VERTSECT[q]], DISJOINT, VERTSECT[q]] == composite[Id, complement[q]]

composite[inverse[VERTSECT[q]], DISJOINT, VERTSECT[q]] := composite[Id, complement[q]]

ImageComp[id[complement[singleton[0]]], VERTSECT[q], x] // Reverse

intersection[complement[singleton[0]], image[VERTSECT[q], x]] ==
 image[VERTSECT[q], intersection[x, fix[q]]]

intersection[complement[singleton[0]], image[VERTSECT[q], x_]] :=
 image[VERTSECT[q], intersection[x, fix[q]]]

Map[intersection[#, cart[complement[singleton[0]], complement[singleton[0]]]] &,
 ImageComp[cross[VERTSECT[q], VERTSECT[q]],
  cross[inverse[VERTSECT[q]], inverse[VERTSECT[q]]], union[Id, DISJOINT]]]

union[composite[id[image[VERTSECT[q], fix[q]]], DISJOINT,
   id[image[VERTSECT[q], fix[q]]]], id[image[VERTSECT[q], fix[q]]]] ==
 cart[image[VERTSECT[q], fix[q]], image[VERTSECT[q], fix[q]]]
```

```
Map[assert[subclass[cart[quot[q], quot[q]], #]] &, %]
```

```
equal[0, intersection[fix[composite[Di, id[image[VERTSECT[q], fix[q]]], E, inverse[E]]],
   image[VERTSECT[q], fix[q]]]] == True
```

```
intersection[fix[composite[Di, id[image[VERTSECT[q], fix[q]]], E, inverse[E]]],
  image[VERTSECT[q], fix[q]]] := 0
```

```
subclass[cart[quot[q], quot[q]], union[DISJOINT, Id]] // AssertTest
```

```
subclass[cart[image[VERTSECT[q], fix[q]], image[VERTSECT[q], fix[q]]],
  union[DISJOINT, Id]] == True
```

```
subclass[cart[image[VERTSECT[q], fix[q]], image[VERTSECT[q], fix[q]]],
  union[DISJOINT, Id]] := True
```

This is the familiar fact that distinct equivalence classes are disjoint.

■ another formula for the natural map

```
subclass[cart[x, y], union[DISJOINT, Id]] // AssertTest // Reverse
```

```
equal[0, intersection[y, fix[composite[Di, id[x], E, inverse[E]]]]] ==
 subclass[cart[x, y], union[DISJOINT, Id]]
```

```
equal[0, intersection[y_, fix[composite[Di, id[x_], E, inverse[E]]]]] :=
 subclass[cart[x, y], union[DISJOINT, Id]]
```

The following result is general:

```
FUNCTION[composite[id[x], E]] // AssertTest // InvertFix
```

```
FUNCTION[composite[id[x], E]] == subclass[cart[x, x], union[DISJOINT, Id]]
```

```
FUNCTION[composite[id[x_], E]] := subclass[cart[x, x], union[DISJOINT, Id]]
```

In particular:

```
FUNCTION[composite[id[quot[q]], E]]
```

```
True
```

What is this function? Nothing other than the natural map, as will be shown below. The method employed uses the symmetric difference.

```
symdif[composite[id[image[VERTSECT[q], fix[q]]], E],
   composite[VERTSECT[q], id[fix[q]]]] // domain // Normality
```

```
union[intersection[
   complement[fix[composite[inverse[VERTSECT[q]], id[image[VERTSECT[q], fix[q]]], E]],
   fix[q]], intersection[
   complement[fix[composite[inverse[E], id[image[VERTSECT[q], fix[q]]], VERTSECT[q]]]],
   fix[q], image[inverse[VERTSECT[q]], image[VERTSECT[q], fix[q]]]]] == 0
```

```
union[intersection[
   complement[fix[composite[inverse[VERTSECT[q]], id[image[VERTSECT[q], fix[q]]], E]],
   fix[q]], intersection[
   complement[fix[composite[inverse[E], id[image[VERTSECT[q], fix[q]]], VERTSECT[q]]]],
   fix[q], image[inverse[VERTSECT[q]], image[VERTSECT[q], fix[q]]]]] := 0

SubstTest[composite, x, id[domain[x]],
    x -> symdif[composite[id[image[VERTSECT[q], fix[q]]], E],
    composite[VERTSECT[q], id[fix[q]]]]] // Reverse

union[composite[id[image[VERTSECT[q], fix[q]]],
   intersection[E, complement[VERTSECT[q]]], id[fix[q]]],
  composite[id[image[VERTSECT[q], fix[q]]], intersection[complement[E], VERTSECT[q]],
   id[fix[q]]], composite[id[image[VERTSECT[q], fix[q]]],
   inverse[IMAGE[id[complement[fix[q]]]], E]] == 0

union[composite[id[image[VERTSECT[q], fix[q]]],
   intersection[E, complement[VERTSECT[q]]], id[fix[q]]],
  composite[id[image[VERTSECT[q], fix[q]]], intersection[complement[E], VERTSECT[q]],
   id[fix[q]]], composite[id[image[VERTSECT[q], fix[q]]],
   inverse[IMAGE[id[complement[fix[q]]]], E]] := 0

SubstTest[equal, 0, symdif[u, v],
    {u -> composite[id[image[VERTSECT[q], fix[q]]], E],
  v -> composite[VERTSECT[q], id[fix[q]]]}]

True ==
 equal[composite[id[image[VERTSECT[q], fix[q]]], E], composite[VERTSECT[q], id[fix[q]]]]
```

The net result is a simple formula for the natural map.

```
composite[id[image[VERTSECT[q], fix[q]]], E] := composite[VERTSECT[q], id[fix[q]]]
```