invariant hull: range[iterate[x,y]]

Johan G. F. Belinfante
2003 January 8

```
<< goedel52.q93; << tools.m
```

```
:Package Title: goedel52.q93          2003 January 7 at 3:40 p.m.
```

```
It is now:  2003 Jan 8 at 22:2
```

```
Loading Simplification Rules
```

```
TOOLS.M                       Revised 2002 December 27
```

```
weightlimit = 40
```

■ definitions and summary

This notebook contains an application of iteration to the theory of invariant subsets. Recall that a class **y** is ***invariant*** under **x** if:

```
invariant[x, y]
```

```
subclass[image[x, y], y]
```

The class of all sets invariant under **x** is

```
class[y, invariant[x, y]]
```

```
invar[x]
```

The intersection of all sets belonging to **x** that contain a set **y** is the **x-*hull*** of **y**. The corresponding function is called **HULL[x]**.

```
lambda[y, A[intersection[x, image[S, singleton[y]]]]]
```

```
HULL[x]
```

A relation **x** is ***thin*** if all its vertical sections are sets:

```
thin[x]
```

```
equal[V, domain[VERTSECT[x]]]
```

```
assert[forall[y, member[image[x, singleton[y]], V]]]
```

```
equal[V, domain[VERTSECT[x]]]
```

A relation **x** is ***idempotent*** if the composite of **x** with itself is **x**.

```
idempotent[x_] := equal[composite[x, x], x]
```

In this notebook it is shown that when **x** is thin, the function **HULL[invar[x]]** is idempotent.

■ invariant substitution

Equality substitutions can be made in the predicate **invariant**. If **y** and **z** are equal, and if one is invariant under **x**, so is the other.

```
Map[not,
 SubstTest[and, implies[p1, p2], implies[and[p2, p3], p4], implies[and[p1, p4], p5],
        not[implies[and[p1, p3], p5]],
    {p1 -> equal[y, z], p2 -> equal[image[x, y], image[x, z]], p3 -> invariant[x, y],
                p4 -> subclass[image[x, z], y], p5 -> invariant[x, z]}]]
```

```
or[not[equal[y, z]], not[subclass[image[x, y], y]], subclass[image[x, z], z]] == True
```

```
or[not[equal[y_, z_]],
  not[subclass[image[x_, y_], y_]], subclass[image[x_, z_], z_]] := True
```

In particular:

```
SubstTest[implies, and[equal[r, z], invariant[x, r]], invariant[x, z],
    r -> range[iterate[x, y]]]
```

```
or[not[equal[z, range[iterate[x, y]]]], subclass[image[x, z], z]] == True
```

```
or[not[equal[z_, range[iterate[x_, y_]]]], subclass[image[x_, z_], z_]] := True
```

■ a fact about idempotents

If **x** is idempotent, then so is the function **IMAGE[x]**. This is true whether or not **x** is thin:

```
Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
        {p1 -> equal[x, y], p2 -> equal[IMAGE[x], IMAGE[y]],
   p3 -> equal[composite[IMAGE[x], id[z]], composite[IMAGE[y], id[z]]]}]]
```

```
or[equal[composite[IMAGE[x], id[z]], composite[IMAGE[y], id[z]]], not[equal[x, y]]] ==
 True
```

```
% /. {y -> composite[x, x], z -> P[domain[VERTSECT[x]]]}
```

```
or[equal[composite[IMAGE[x], IMAGE[x]], IMAGE[x]],
  not[equal[x, composite[x, x]]]] == True
```

```
or[equal[composite[IMAGE[x_], IMAGE[x_]], IMAGE[x_]],
  not[equal[x_, composite[x_, x_]]]] := True
```

Thus:

```
implies[idempotent[x], idempotent[IMAGE[x]]]
```

```
True
```

It is our intention to apply this result to the idempotent relation **union[Id,trv[x]]**. But first some lemmas need to be derived.

```
idempotent[union[Id, trv[x]]]
```

```
True
```

■ a formula for IMAGE[union[Id,trv[x]]].

It will be shown in this section that the following formula holds not only for thin relations, but for any relation:

```
IMAGE[union[Id, trv[th]]]
```

```
composite[CUP, id[IMAGE[trv[th]]], inverse[FIRST]]
```

The general case can be derived using **VSNormality** with the **simplify** flag turned off.

```
simplify = False;
```

```
composite[CUP, id[IMAGE[trv[x]]], inverse[FIRST]] // VSNormality // Reverse
```

```
intersection[S, complement[composite[complement[S], IMAGE[SECOND],
    IMAGE[id[cart[complement[singleton[0]], V]]], id[intersection[
      P[cart[omega, V]], subvar[union[cross[SUCC, x]], id[cart[singleton[0], V]]]]]],
    intersection[complement[composite[inverse[GREATEST[complement[cross[SUCC, x]]]],
       inverse[SECOND], complement[inverse[E]]]], composite[inverse[
       IMAGE[id[complement[cart[omega, V]]]]], inverse[IMAGE[SECOND]], inverse[S]]]]],
  inverse[LB[union[composite[inverse[E], IMAGE[SECOND],
      IMAGE[id[cart[complement[singleton[0]], V]]], id[intersection[
        P[cart[omega, V]], subvar[union[cross[SUCC, x]], id[cart[singleton[0], V]]]]],
      intersection[complement[composite[inverse[GREATEST[complement[cross[SUCC, x]]]],
         inverse[SECOND], complement[inverse[E]]]],
       composite[inverse[IMAGE[id[complement[cart[omega, V]]]]],
        inverse[IMAGE[SECOND]], inverse[S]]]], inverse[E]]]] ==
 composite[CUP, id[IMAGE[trv[x]]], inverse[FIRST]]
```

```
intersection[S, complement[composite[complement[S], IMAGE[SECOND],
    IMAGE[id[cart[complement[singleton[0]], V]]], id[intersection[P[cart[omega, V]],
      subvar[union[cross[SUCC, x_], id[cart[singleton[0], V]]]]]],
    intersection[complement[composite[inverse[GREATEST[complement[cross[SUCC, x_]]]],
       inverse[SECOND], complement[inverse[E]]]], composite[inverse[
       IMAGE[id[complement[cart[omega, V]]]]], inverse[IMAGE[SECOND]], inverse[S]]]]],
  inverse[LB[union[composite[inverse[E], IMAGE[SECOND],
      IMAGE[id[cart[complement[singleton[0]], V]]], id[intersection[P[cart[omega, V]],
        subvar[union[cross[SUCC, x_], id[cart[singleton[0], V]]]]]], intersection[
       complement[composite[inverse[GREATEST[complement[cross[SUCC, x_]]]],
         inverse[SECOND], complement[inverse[E]]]],
       composite[inverse[IMAGE[id[complement[cart[omega, V]]]]],
        inverse[IMAGE[SECOND]], inverse[S]]]], inverse[E]]]] :=
 composite[CUP, id[IMAGE[trv[x]]], inverse[FIRST]]
```

```
IMAGE[union[Id, trv[x]]] // VSNormality
```

```
IMAGE[union[Id, trv[x]]] == composite[CUP, id[IMAGE[trv[x]]], inverse[FIRST]]
```

```
IMAGE[union[Id, trv[x_]]] := composite[CUP, id[IMAGE[trv[x]]], inverse[FIRST]]
```

■ a membership rule for a certain function

This function was encountered in the preceding section:

```
VERTSECT[reify[y, range[iterate[x, y]]]]

composite[CUP, id[IMAGE[trv[x]]], inverse[FIRST]]
```

To get a nice membership rule for this function, the following rewrite rule is needed:

```
SubstTest[image, w, union[u, v],
  {u -> singleton[0], v -> complement[singleton[0]], w -> iterate[x, y]}] // Reverse

union[y, image[iterate[x, y], complement[singleton[0]]]] == range[iterate[x, y]]

union[y_, image[iterate[x_, y_], complement[singleton[0]]]] := range[iterate[x, y]]
```

This is the membership rule that is required:

```
SubstTest[member, z, image[w, singleton[y]],
  w -> composite[CUP, id[IMAGE[trv[x]]], inverse[FIRST]]] // Reverse

member[pair[y, z], composite[CUP, id[IMAGE[trv[x]]], inverse[FIRST]]] ==
 and[equal[z, range[iterate[x, y]]], member[z, V]]

member[pair[y_, z_], composite[CUP, id[IMAGE[trv[x_]]], inverse[FIRST]]] :=
 and[equal[z, range[iterate[x, y]]], member[z, V]]
```

■ formula for the range

```
SubstTest[class, pair[y, z], implies[member[pair[y, z], w], member[z, invar[x]]],
    w -> composite[CUP, id[IMAGE[trv[x]]], inverse[FIRST]]]

cart[V, V] == union[cart[V, invar[x]], cart[complement[P[domain[VERTSECT[trv[x]]]]], V],
  composite[complement[CUP], id[IMAGE[trv[x]]], inverse[FIRST]]]

Map[equal[0, composite[Id, complement[#]]] &, %] // Reverse

subclass[image[CUP, IMAGE[trv[x]]], invar[x]] == True

subclass[image[CUP, IMAGE[trv[x_]]], invar[x_]] := True
```

■ an equation

The class **range[iterate[x,y]]** is the smallest class that contains **y** and is invariant under **x**. When **x** is thin, this fact can be expressed as a propery of the function **HULL[invar[x]]**. To derive this formula, one needs an equation that is derived in this section by proving inclusions in both directions. In one direction this inclusion holds:

```
SubstTest[implies, member[u, v], subclass[A[v], u],
    {u -> range[iterate[x, y]], v -> intersection[invar[x], image[S, singleton[y]]]}]
```

```
or[not[member[y, V]], not[member[range[iterate[x, y]], V]], subclass[
   A[intersection[image[S, singleton[y]], invar[x]]], range[iterate[x, y]]]] == True
```

```
or[not[member[y_, V]], not[member[range[iterate[x_, y_]], V]], subclass[
   A[intersection[image[S, singleton[y_]], invar[x_]]], range[iterate[x_, y_]]]] := True
```

Although we will not do so, we note that this result could be simplified using the following fact:

```
SubstTest[implies, and[subclass[y, z], member[z, V]], member[y, V],
    z -> range[iterate[x, y]]]
```

```
or[member[y, V], not[member[range[iterate[x, y]], V]]] == True
```

```
or[member[y_, V], not[member[range[iterate[x_, y_]], V]]] := True
```

In the opposite direction:

```
Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], implies[p4, p5],
implies[and[p3, p5], p6],
        not[implies[and[p1, p4], p6]],
        {p1 -> subclass[y, z], p2 -> subclass[iterate[x, y], iterate[x, z]],
            p3 -> subclass[range[iterate[x, y]], range[iterate[x, z]]],
            p4 -> invariant[x, z], p5 -> equal[range[iterate[x, z]], z],
            p6 -> subclass[range[iterate[x, y]], z]}]]
```

```
or[not[subclass[y, z]], not[subclass[image[x, z], z]],
  subclass[range[iterate[x, y]], z]] == True
```

```
or[not[subclass[y_, z_]], not[subclass[image[x_, z_], z_]],
  subclass[range[iterate[x_, y_]], z_]] := True
```

```
Map[equal[V, #] &, SubstTest[class, z,
   or[not[subclass[y, z]], not[subclass[image[x, z], z]], subclass[r, z]],
    r -> range[iterate[x, y]]]] // Reverse
```

```
subclass[range[iterate[x, y]], A[intersection[image[S, singleton[y]], invar[x]]]] == True
```

```
subclass[range[iterate[x_, y_]],
  A[intersection[image[S, singleton[y_]], invar[x_]]]] := True
```

These two inclusions can be combined into an equation:

```
SubstTest[and, subclass[u, v], implies[p, subclass[v, u]],
    {p -> and[member[y, V], member[range[iterate[x, y]], V]],
        u -> range[iterate[x, y]],
   v -> A[intersection[image[S, singleton[y]], invar[x]]]}] // Reverse
```

```
or[equal[A[intersection[image[S, singleton[y]], invar[x]]], range[iterate[x, y]]],
  not[member[y, V]], not[member[range[iterate[x, y]], V]]] == True
```

```
or[equal[A[intersection[image[S, singleton[y_]], invar[x_]]], range[iterate[x_, y_]]],
  not[member[y_, V]], not[member[range[iterate[x_, y_]], V]]] := True
```

■ rewriting the equation

The equation derived in the preceding section can be reformulated in terms of the **HULL** function:

```
implies[and[member[y, V], member[range[iterate[x, y]], V]],
    member[pair[y, range[iterate[x, y]]], HULL[invar[x]]]]

True
```

The following equivalent statement may be more convenient:

```
Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[and[p4, p5], p2],
  implies[and[p5, p3], p6], not[implies[and[p1, p4, p5], p6]],
    {p1 -> member[y, V], p2 -> member[range[iterate[x, y]], V],
        p3 ->
    equal[A[intersection[image[S, singleton[y]], invar[x]]], range[iterate[x, y]]],
        p4 -> member[z, V], p5 -> equal[z, range[iterate[x, y]]],
        p6 -> equal[z, A[intersection[image[S, singleton[y]], invar[x]]]]}]]

or[equal[z, A[intersection[image[S, singleton[y]], invar[x]]]],
  not[equal[z, range[iterate[x, y]]]], not[member[y, V]], not[member[z, V]]] == True

or[equal[z_, A[intersection[image[S, singleton[y_]], invar[x_]]]],
  not[equal[z_, range[iterate[x_, y_]]]], not[member[y_, V]], not[member[z_, V]]] := True
```

This observation makes it easy to eliminate the variables **y** and **z**.

```
Map[composite[Id, complement[#]] &, SubstTest[class, pair[y, z],
   implies[and[member[y, V], member[z, V], member[pair[y, z], u]],
    member[pair[y, z], v]],
    {u -> composite[CUP, id[IMAGE[trv[x]]], inverse[FIRST]],
        v -> HULL[invar[x]]}]] // Reverse

composite[intersection[complement[HULL[invar[x]]],
   composite[CUP, id[IMAGE[trv[x]]], inverse[FIRST]]],
  id[P[domain[VERTSECT[trv[x]]]]]] == 0

composite[intersection[complement[HULL[invar[x_]]],
   composite[CUP, id[IMAGE[trv[x_]]], inverse[FIRST]]],
  id[P[domain[VERTSECT[trv[x_]]]]]] := 0
```

This can be written as an inclusion:

```
SubstTest[equal, 0, dif[u, v],
    {u -> composite[CUP, id[IMAGE[trv[x]]],
     inverse[FIRST], id[P[domain[VERTSECT[trv[x]]]]]],
    v -> composite[HULL[invar[x]], id[P[domain[VERTSECT[trv[x]]]]]]}] // Reverse

subclass[composite[CUP, id[IMAGE[trv[x]]], inverse[FIRST]], HULL[invar[x]]] == True

subclass[composite[CUP, id[IMAGE[trv[x_]]], inverse[FIRST]], HULL[invar[x_]]] := True
```

The inclusion can be rewritten as an equation:

```
SubstTest[implies, and[FUNCTION[v], subclass[u, v]],
 equal[u, composite[v, id[domain[u]]]],
    {u -> composite[CUP, id[IMAGE[trv[x]]], inverse[FIRST]], v -> HULL[invar[x]]}]

equal[composite[HULL[invar[x]], id[P[domain[VERTSECT[trv[x]]]]]],
  composite[CUP, id[IMAGE[trv[x]]], inverse[FIRST]]] == True
```

As a guide to orienting this equation, it is useful to consider the following special case:

```
composite[CUP, id[IMAGE[inverse[S]]], inverse[FIRST]] // VSNormality

composite[CUP, id[IMAGE[inverse[S]]], inverse[FIRST]] == IMAGE[inverse[S]]
```

```
composite[CUP, id[IMAGE[inverse[S]]], inverse[FIRST]] := IMAGE[inverse[S]]
```

This suggests the following orientation in the general case:

```
composite[CUP, id[IMAGE[trv[x_]]], inverse[FIRST]] :=
 composite[HULL[invar[x]], id[P[domain[VERTSECT[trv[x]]]]]]
```

For a thin relation this simplifies:

```
composite[CUP, id[IMAGE[trv[th]]], inverse[FIRST]]
```

```
HULL[invar[th]]
```

As a corollary, we find that **HULL[invar[x]]** is idempotent in this case:

```
SubstTest[implies, equal[composite[z, z], z],
 equal[composite[IMAGE[z], IMAGE[z]], IMAGE[z]],
    z -> union[Id, trv[th]]]
```

```
equal[composite[HULL[invar[th]], HULL[invar[th]]], HULL[invar[th]]] == True
```

Thus:

```
idempotent[HULL[invar[th]]]
```

```
True
```

```
composite[HULL[invar[x_]], HULL[invar[x_]]] := HULL[invar[x]] /; cond && thin[x]
```

■ a formula for the range of HULL[invar[x]]

```
Map[subclass[#, invar[th]] &,
 ImageComp[composite[CUP, id[IMAGE[trv[th]]]], inverse[FIRST], V]]
```

```
subclass[range[HULL[invar[th]]], invar[th]] == True
```

This rule is only temporary, because it will be improved momentarily.

```
subclass[range[HULL[invar[x_]]], invar[x_]] := True /; cond && thin[x]
```

This inclusion can be strengthened to an equality:

```
SubstTest[and, subclass[u, v], subclass[v, u],
    {u -> range[HULL[invar[th]]], v -> invar[th]}]
```

```
True == equal[invar[th], range[HULL[invar[th]]]]
```

```
range[HULL[invar[x_]]] := invar[x] /; cond && thin[x]
```

A similar formula hold for the fixed point class:

```
SubstTest[implies,
 and[FUNCTION[z], equal[composite[z, z], z]], equal[range[z], fix[z]],
    z -> HULL[invar[th]]]
```

```
equal[fix[HULL[invar[th]]], invar[th]] == True
```

```
fix[HULL[invar[x_]]] := invar[x] /; cond && thin[x]
```