HULL[invar[SWAP]]

Johan G. Belinfante and Tiffany D. Goble
2004 June 15

```
In[1]:= SetDirectory["p:"]; << goedel58.15a; << tools.m
```

```
:Package Title: goedel58.15a     2004 June 15 at 2:05 a.m.

It is now:  2004 Jun 15 at 3:52

Loading Simplification Rules

TOOLS.M                       Revised 2004 May 14

weightlimit = 40
```

summary

Any idempotent function that is contained in and subcommutes with the subset relation **S** is a **HULL** function. In this notebook the function **IMAGE[union[Id,SWAP]]** is shown to have these properties, which implies that this function is equal to **HULL[invar[SWAP]]**.

comments

The following formula is already known to the **GOEDEL** program:

```
In[2]:= composite[IMAGE[union[Id, SWAP]], id[P[cart[V, V]]]]
Out[2]= HULL[SYM]
```

The object of this notebook is to derive a similar result for the function **IMAGE[union[Id, SWAP]]** by itself without the factor **id[P[cart[V, V]]]**. The result involves **invar[SWAP]**, which is closely related to the class **SYM** of all symmetric relations. To see the difference, compare these formulas:

```
In[3]:= class[x, subclass[inverse[x], x]]
Out[3]= invar[SWAP]

In[4]:= class[x, equal[inverse[x], x]]
Out[4]= SYM
```

The following formulas show how the one can be obtained from the other, and vice versa.

```
In[5]:= intersection[invar[SWAP], P[cart[V, V]]]
Out[5]= SYM
In[6]:= image[inverse[IMAGE[id[cart[V, V]]]], SYM]
Out[6]= invar[SWAP]
```

The main problem that one encounters in the derivation of a formula for **IMAGE[union[Id, SWAP]]** is that there is a rewrite rule that rewrites it:

```
In[8]:= IMAGE[union[Id, SWAP]]
Out[8]= composite[CUP, id[IMAGE[SWAP]], inverse[FIRST]]
```

This problem can be circumvented by turning off the **cond** flag: This is not strictly necessary, but it simplifies some intermediate formulas.

```
In[12]:= cond = False;
In[13]:= IMAGE[union[Id, SWAP]]
Out[13]= IMAGE[union[Id, SWAP]]
```

idempotence

Definition of idempotent

```
In[7]:= idempotent[x_] := equal[x, composite[x, x]]
```

The function **IMAGE[union[Id,SWAP]]** is idempotent. The main obstacle to proving this is that a rewrite rule kicks in and rewrites

```
In[14]:= SubstTest[implies, idempotent[x], idempotent[IMAGE[x]], x → union[Id, SWAP]]
Out[14]= equal[composite[IMAGE[union[Id, SWAP]], IMAGE[union[Id, SWAP]]],
           IMAGE[union[Id, SWAP]]] == True
In[15]:= % /. Equal → SetDelayed
```

increasing

The function **IMAGE[union[Id,SWAP]]** is a subclass of the subset relation **S**. This can be viewed as a special case of a more general result. We begin with some lemmas:

```
In[16]:= SubstTest[implies, subclass[u, x], subclass[image[u, y], image[x, y]], u → Id]
Out[16]= or[not[equal[V, fix[x]]], subclass[y, image[x, y]]] == True
In[17]:= or[not[equal[V, fix[x_]]], subclass[y_, image[x_, y_]]] := True
```

Lemma.

```
In[18]:= SubstTest[equal, 0, composite[Id, x], x → dif[IMAGE[x], S]] // Reverse
Out[18]= subclass[P[domain[VERTSECT[x]]], subvar[x]] == subclass[IMAGE[x], S]
In[19]:= subclass[P[domain[VERTSECT[x_]]], subvar[x_]] := subclass[IMAGE[x], S]
```

A general result:

```
In[20]:= Map[assert, SubstTest[implies, and[subclass[u, v], subclass[v, w]],
           subclass[u, w], {u → P[domain[VERTSECT[x]]], v → V, w → subvar[x]}]]
Out[20]= or[not[equal[V, fix[x]]], subclass[IMAGE[x], S]] == True
In[21]:= or[not[equal[V, fix[x_]]], subclass[IMAGE[x_], S]] := True
```

In particular:

```
In[22]:= SubstTest[implies, subclass[Id, x], subclass[IMAGE[x], S], x → union[Id, SWAP]]
Out[22]= subclass[IMAGE[union[Id, SWAP]], S] == True
In[23]:= % /. Equal → SetDelayed
```

---

application of the characterization of HULL functions

```
In[25]:= SubstTest[implies,
          and[FUNCTION[x], idempotent[x], subclass[x, S], subcommute[x, S]],
          equal[x, HULL[fix[x]]], x → IMAGE[union[Id, SWAP]]]
Out[25]= equal[HULL[invar[SWAP]], IMAGE[union[Id, SWAP]]] == True
```

This result is rewritten when the **cond** flag is turned back on. We do so now.

```
In[27]:= cond = True;
```

A more general result can be derived. We begin by repeating the above characterization, replacing **union[Id, SWAP]** with a general class **x**.

```
In[31]:= SubstTest[implies,
          and[FUNCTION[y], idempotent[y], subclass[y, S], subcommute[y, S]],
          equal[y, HULL[fix[y]]], y → IMAGE[x]]

Out[31]= or[equal[HULL[fix[IMAGE[x]]], IMAGE[x]],
           not[equal[composite[IMAGE[x], IMAGE[x]], IMAGE[x]]],
           not[subclass[IMAGE[x], S]]] == True

In[32]:= (% /. x → x_) /. Equal → SetDelayed
```

A general theorem:

```
In[34]:= Map[not, SubstTest[and, implies[p1, p3], implies[p2, p4],
           implies[and[p3, p4], p5], not[implies[and[p1, p2], p5]],
           {p1 → subclass[Id, x], p2 → idempotent[x], p3 → subclass[IMAGE[x], S],
            p4 → idempotent[IMAGE[x]], p5 -> equal[HULL[fix[IMAGE[x]]], IMAGE[x]]}]]

Out[34]= or[equal[HULL[fix[IMAGE[x]]], IMAGE[x]],
           not[equal[V, fix[x]]], not[equal[x, composite[x, x]]]] == True

In[35]:= or[equal[HULL[fix[IMAGE[x_]]], IMAGE[x_]],
           not[equal[V, fix[x_]]], not[equal[x_, composite[x_, x_]]]] := True
```

The special formula can be derived as a corollary of this more general result.

```
In[36]:= SubstTest[implies, and[subclass[Id, x], idempotent[x]],
          equal[HULL[fix[IMAGE[x]]], IMAGE[x]], x → union[Id, SWAP]]

Out[36]= equal[composite[CUP, id[IMAGE[SWAP]], inverse[FIRST]], HULL[invar[SWAP]]] ==
          True

In[37]:= Equal[composite[CUP, id[IMAGE[SWAP]], inverse[FIRST]], HULL[invar[SWAP]]]

Out[37]= composite[CUP, id[IMAGE[SWAP]], inverse[FIRST]] == HULL[invar[SWAP]]

In[38]:= composite[CUP, id[IMAGE[SWAP]], inverse[FIRST]] := HULL[invar[SWAP]]
```

---

corollary

```
In[40]:= Assoc[CUP, composite[id[IMAGE[SWAP]], inverse[FIRST]],
           id[P[cart[V, V]]]] // Reverse

Out[40]= composite[HULL[invar[SWAP]], id[P[cart[V, V]]]] == HULL[SYM]

In[41]:= composite[HULL[invar[SWAP]], id[P[cart[V, V]]]] := HULL[SYM]
```