IDEM and PROJ

Johan G. F. Belinfante and Shilpakala Vasudevan
2004 December 14

```
In[1]:= SetDirectory["i:"]; << goedel64.14a; << tools.m

        :Package Title: goedel64.14a      2004 December 14 at 9:15 a.m.

        It is now:  2004 Dec 14 at 20:38

        Loading Simplification Rules

        TOOLS.M                       Revised 2004 November 17

        weightlimit = 40
```

summary

The term **idempotent** in the **GOEDEL** program refers to a relation that satisfies **composite[x,x] = x**. It is natural therefore to define **IDEM** to be the class of such relations.

```
In[2]:= idempotent[x]

Out[2]= equal[x, composite[x, x]]

In[3]:= member[x_, IDEM] := and[member[x, V], equal[composite[x, x], x]]
```

Some properties of the class **IDEM** are derived in this notebook.

projections

Comment: Until now the name **IDEM** had been used for the more restrictive class of idempotent functions. To avoid confusion, that class has been renamed **PROJ**.

```
In[4]:= ?? PROJ

        PROJ is the class of all projections, that is, idempotent functions

In[5]:= member[x, PROJ]

Out[5]= and[equal[x, composite[x, x]], FUNCTION[x], member[x, V]]
```

normalization

The class **IDEM** is normalized by this rule:

```
In[6]:= IDEM // Normality // Reverse
Out[6]= fix[composite[COMPOSE, DUP]] == IDEM

In[7]:= fix[composite[COMPOSE, DUP]] := IDEM
```

This affects the normalization of **PROJ**.

```
In[8]:= PROJ // Normality // Reverse
Out[8]= intersection[FUNS, IDEM] == PROJ

In[9]:= intersection[FUNS, IDEM] := PROJ
```

Corollary.

```
In[10]:= subclass[PROJ, IDEM] // AssertTest
Out[10]= subclass[PROJ, IDEM] == True

In[11]:= subclass[PROJ, IDEM] := True
```

A special result: the only idempotent bijections are identity functions.

```
In[12]:= AssInt[BIJ, FUNS, IDEM] // Reverse
Out[12]= intersection[BIJ, IDEM] == P[Id]

In[13]:= intersection[BIJ, IDEM] := P[Id]
```

connections with EQV and TRV

The class **IDEM** is intermediate between **EQV** and **TRV**.

```
In[14]:= subclass[EQV, IDEM] // AssertTest
Out[14]= subclass[EQV, IDEM] == True

In[15]:= subclass[EQV, IDEM] := True

In[16]:= subclass[IDEM, TRV] // AssertTest
Out[16]= subclass[IDEM, TRV] == True
```

```
In[17]:= subclass[IDEM, TRV] := True
```

Corollaries:

```
In[18]:= equal[intersection[IDEM, TRV], IDEM]
Out[18]= True

In[19]:= intersection[IDEM, TRV] := IDEM

In[20]:= equal[intersection[EQV, IDEM], EQV]
Out[20]= True

In[21]:= intersection[EQV, IDEM] := EQV

In[22]:= AssInt[SYM, TRV, IDEM]
Out[22]= intersection[IDEM, SYM] == EQV

In[23]:= intersection[IDEM, SYM] := EQV

In[24]:= AssInt[IDEM, TRV, invar[SWAP]] // Reverse
Out[24]= intersection[IDEM, invar[SWAP]] == EQV

In[25]:= intersection[IDEM, invar[SWAP]] := EQV
```

---

other inclusions

The class of partial orderings is contained in **IDEM**.

```
In[26]:= subclass[PO, IDEM] // AssertTest
Out[26]= subclass[PO, IDEM] == True

In[27]:= subclass[PO, IDEM] := True
```

Further corollaries.

```
In[28]:= subclass[TO, IDEM] // AssertTest
Out[28]= subclass[TO, IDEM] == True

In[29]:= subclass[TO, IDEM] := True

In[30]:= SubstTest[implies, and[subclass[x, y], subclass[y, z]],
          subclass[x, z], {x -> WO, y → TO, z → IDEM}]
Out[30]= subclass[WO, IDEM] == True
```

```
In[31]:= subclass[WO, IDEM] := True

In[32]:= SubstTest[implies, and[subclass[x, y], subclass[y, z]],
          subclass[x, z], {x → P[Id], y → PROJ, z → IDEM}]

Out[32]= subclass[P[Id], IDEM] == True

In[33]:= subclass[P[Id], IDEM] := True
```

U[IDEM]

Inclusion in one direction:

```
In[34]:= Map[equal[0, #] &, dif[IDEM, P[cart[V, V]]] // Renormality]

Out[34]= subclass[U[IDEM], cart[V, V]] == True

In[35]:= % /. Equal → SetDelayed
```

The following lemma about idempotent cartesian products will be needed to obtain an inclusion in the opposite direction.

```
In[36]:= Map[image[CART, #] &, image[inverse[CART], IDEM] // RelnNormality]

Out[36]= intersection[IDEM, range[CART]] ==
          union[image[CART, composite[E, inverse[E]]], singleton[0]]

In[37]:= intersection[IDEM, range[CART]] :=
          union[image[CART, composite[E, inverse[E]]], singleton[0]]
```

The following is an application of this formula.

```
In[38]:= SubstTest[subclass, U[intersection[x, y]], U[x], {x → IDEM, y → range[CART]}]

Out[38]= subclass[cart[V, V], U[IDEM]] == True

In[39]:= % /. Equal → SetDelayed
```

The two inclusions can be combined into an equation:

```
In[40]:= equal[U[IDEM], cart[V, V]] // assert

Out[40]= True

In[41]:= U[IDEM] := cart[V, V]
```

Corollary.

```
In[42]:= equal[intersection[IDEM, P[cart[V, V]]], IDEM]
Out[42]= True
In[43]:= intersection[IDEM, P[cart[V, V]]] := IDEM
In[44]:= ImageComp[IMAGE[id[cart[V, V]]], id[P[cart[V, V]]], IDEM] // Reverse
Out[44]= image[IMAGE[id[cart[V, V]]], IDEM] == IDEM
In[45]:= image[IMAGE[id[cart[V, V]]], IDEM] := IDEM
```

---

analogous results for PROJ

In this section some corollaries for the class of projections are derived. To derive the formula for U[PRPOJ] one needs to know that every ordered pair belongs to some idempotent function. Note that a constant function is idempotent if its value belongs to its domain:

```
In[46]:= member[cart[x, singleton[y]], PROJ]
Out[46]= or[and[member[x, V], member[y, x]], equal[0, x], not[member[y, V]]]
```

An example of such a function is **cart[pairset[x,y],singleton[y]]**. This observation yields:

```
In[47]:= SubstTest[implies, and[member[u, v], member[v, w]], member[u, U[w]],
          {u → pair[x, y], v -> cart[pairset[x, y], singleton[y]], w -> PROJ}]
Out[47]= or[member[pair[x, y], U[PROJ]], not[member[x, V]], not[member[y, V]]] == True
In[48]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
In[49]:= Map[equal[0, composite[Id, complement[#]]] &,
           SubstTest[class, pair[x, y], or[member[pair[x, y], z],
             not[member[x, V]], not[member[y, V]]], z -> U[PROJ]]] // Reverse
Out[49]= subclass[cart[V, V], U[PROJ]] == True
In[50]:= % /. Equal → SetDelayed
```

For the reverse inclusion, one can use the fact that **PROJ** is a subclass of **IDEM**.

```
In[51]:= SubstTest[implies, subclass[x, y], subclass[U[x], U[y]], {x → PROJ, y → IDEM}]
Out[51]= subclass[U[PROJ], cart[V, V]] == True
In[52]:= % /. Equal → SetDelayed
```

```
In[53]:= SubstTest[and, subclass[u, v], subclass[v, u], {u -> U[PROJ], v -> cart[V, V]}]
Out[53]= True == equal[cart[V, V], U[PROJ]]

In[54]:= U[PROJ] := cart[V, V]
```

Other corollaries:

```
In[55]:= AssInt[P[cart[V, V]], FUNS, IDEM]
Out[55]= intersection[PROJ, P[cart[V, V]]] == PROJ

In[56]:= intersection[PROJ, P[cart[V, V]]] := PROJ

In[57]:= ImageComp[IMAGE[id[cart[V, V]]], id[P[cart[V, V]]], PROJ] // Reverse
Out[57]= image[IMAGE[id[cart[V, V]]], PROJ] == PROJ

In[58]:= image[IMAGE[id[cart[V, V]]], PROJ] := PROJ
```

---

inverses of idempotent relations

Lemma.

```
In[59]:= SubstTest[implies, equal[x, y],
          equal[composite[Id, x], composite[Id, y]], y → composite[x, x]]
Out[59]= or[equal[composite[Id, x], composite[x, x]],
           not[equal[x, composite[x, x]]]] == True

In[60]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[61]:= Map[implies[idempotent[x], #] &,
          SubstTest[equal, inverse[x], inverse[y], y → composite[x, x]]]
Out[61]= or[equal[composite[inverse[x], inverse[x]], inverse[x]],
           not[equal[x, composite[x, x]]]] == True

In[62]:= (% /. x → x_) /. Equal → SetDelayed

In[63]:= implies[member[x, IDEM], member[inverse[x], IDEM]] // NotNotTest
Out[63]= or[and[equal[composite[inverse[x], inverse[x]], inverse[x]],
             member[domain[x], V], member[range[x], V]],
           not[equal[x, composite[x, x]]], not[member[x, V]]] == True

In[64]:= (% /. x → x_) /. Equal → SetDelayed
```

```
In[65]:= Map[equal[V, #] &, SubstTest[class, x,
           implies[member[x, y], member[inverse[x], y]], y → IDEM]] // Reverse
```

```
Out[65]= subclass[image[IMAGE[SWAP], IDEM], IDEM] == True
```

```
In[66]:= (% /. x → x_) /. Equal → SetDelayed
```

```
In[67]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u -> image[IMAGE[SWAP], IDEM], v -> IDEM, w → IMAGE[SWAP]}]
```

```
Out[67]= subclass[IDEM, image[IMAGE[SWAP], IDEM]] == True
```

```
In[68]:= (% /. x → x_) /. Equal → SetDelayed
```

```
In[69]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> image[IMAGE[SWAP], IDEM], v → IDEM}]
```

```
Out[69]= True == equal[IDEM, image[IMAGE[SWAP], IDEM]]
```

```
In[70]:= image[IMAGE[SWAP], IDEM] := IDEM
```

Corollary.

```
In[71]:= ImageComp[IMAGE[SWAP], id[P[cart[V, V]]], IDEM]
```

```
Out[71]= image[INVERSE, IDEM] == IDEM
```

```
In[72]:= image[INVERSE, IDEM] := IDEM
```

There is no corresponding result for **PROJ**. For example, the inverse of a constant function need not be a function.