λ x. ids[x]

Johan G. F. Belinfante
2011 April 22

```
In[1]:= SetDirectory["l:"]; << goedel.11apr19a

        :Package Title: goedel.11apr19a               2011 April 19 at 6:35 p.m.

        It is now:  2011 Apr 22 at 13:46

        Loading Simplification Rules

        TOOLS.M is now incorporated in the GOEDEL program as of 2010 September 3

        weightlimit = 40
```

summary

The class **ids[x]** holds all identity elements for an algebraic system **x** such as a category or binary operation. In this notebook, the function **IDS = λ x. ids[x]** is defined, and some of its properties are derived.

sethood rules

In this section, some sethood rules are derived for the constructor **ids[x]**. These rules follow from the following upper bound, which says that identity elements are (by definition) idempotent.

```
In[2]:= subclass[ids[x], fix[composite[x, DUP]]]

Out[2]= True
```

Lemma. If **x** is a set, then **ids[x]** is a set.

```
In[3]:= SubstTest[implies, and[subclass[u, v], member[v, V]], member[u, V],
          {u -> ids[setpart[x]], v -> fix[composite[setpart[x], DUP]]}] // Reverse

Out[3]= member[ids[setpart[x]], V] == True

In[4]:= member[ids[setpart[x_]], V] := True
```

Theorem.

```
In[5]:= Map[implies[member[x, y], #] &,
          SubstTest[implies, equal[x, setpart[t]], member[ids[x], V], t → x]] // Reverse

Out[5]= or[member[ids[x], V], not[member[x, y]]] == True
```

```
In[6]:= or[member[ids[x_], V], not[member[x_, y_]]] := True
```

Corollary. A simplification rule.

```
In[8]:= equiv[and[equal[y, ids[x]], member[x, z], member[y, V]],
          and[equal[y, ids[x]], member[x, z]]]
Out[8]= True

In[9]:= and[equal[y_, ids[x_]], member[x_, z_], member[y_, V]] :=
          and[equal[y, ids[x]], member[x, z]]
```

The following rule will also be needed later.

Theorem.

```
In[10]:= ids[union[x, complement[image[V, set[y]]]]] // Normality
Out[10]= ids[union[x, complement[image[V, set[y]]]]] == intersection[ids[x], image[V, set[y]]]

In[11]:= ids[union[x_, complement[image[V, set[y_]]]]] := intersection[ids[x], image[V, set[y]]]
```

definition

Definition. The class **IDS** is defined by the following membership rule.

```
In[12]:= member[x_, IDS] := and[member[first[x], V], equal[second[x], ids[first[x]]]]
```

Observation. The sethood rules derived in the preceding section simplify the membership rule for pairs as follows.

```
In[13]:= member[pair[x, y], IDS]
Out[13]= and[equal[y, ids[x]], member[x, V]]
```

IDS is a function

Theorem. The class **IDS** is a relation.

```
In[14]:= Map[equal[V, #] &, dif[IDS, cart[V, V]] // complement // Normality]
Out[14]= subclass[IDS, cart[V, V]] == True

In[15]:= subclass[IDS, cart[V, V]] := True
```

Corollary. Simplification rule.

```
In[16]:= equal[composite[Id, IDS], IDS]
Out[16]= True
```

```
In[17]:= composite[Id, IDS] := IDS
```

Theorem. A vertical section rule.

```
In[18]:= image[IDS, set[x]] // Normality

Out[18]= image[IDS, set[x]] == intersection[image[V, set[x]], set[ids[x]]]

In[19]:= image[IDS, set[x_]] := intersection[image[V, set[x]], set[ids[x]]]
```

Theorem. The relation **IDS** is thin.

```
In[20]:= Map[equal[V, #] &,
          SubstTest[class, x, member[setpart[x], t], t -> domain[VERTSECT[IDS]]]]

Out[20]= equal[V, domain[VERTSECT[IDS]]] == True

In[21]:= domain[VERTSECT[IDS]] := V
```

Theorem. The relation **IDS** is a function.

```
In[22]:= Map[empty, SubstTest[reify, x,
            dif[set[image[t, set[setpart[x]]]], union[set[0], range[SINGLETON]]], t → IDS]]

Out[22]= FUNCTION[IDS] == True

In[23]:= FUNCTION[IDS] := True
```

Lemma. A simplification rule.

```
In[24]:= union[complement[image[V, set[x]]], complement[image[V, set[ids[x]]]]] //
          DoubleComplement

Out[24]= union[complement[image[V, set[x]]], complement[image[V, set[ids[x]]]]] ==
          complement[image[V, set[x]]]

In[25]:= union[complement[image[V, set[x_]]], complement[image[V, set[ids[x_]]]]] :=
          complement[image[V, set[x]]]
```

Theorem. An **APPLY** rule for **IDS**.

```
In[26]:= SubstTest[A, image[t, set[x]], t → IDS]

Out[26]= APPLY[IDS, x] == union[complement[image[V, set[x]]], ids[x]]

In[27]:= APPLY[IDS, x_] := union[complement[image[V, set[x]]], ids[x]]
```

normalization

Theorem. Normalization rule.

```
In[28]:= composite[inverse[E], IDS] // FastReifNormality// Reverse

Out[28]= intersection[composite[inverse[E], IMAGE[inverse[DUP]], IMAGE[FIRST]],
           composite[complement[inverse[E]], IMAGE[FIRST], IMAGE[FIRST],
            IMAGE[id[composite[Di, SECOND]]]], composite[complement[inverse[E]], IMAGE[SECOND],
            IMAGE[FIRST], IMAGE[id[composite[Di, FIRST]]]]] == composite[inverse[E], IDS]

In[29]:= intersection[composite[inverse[E], IMAGE[inverse[DUP]], IMAGE[FIRST]],
           composite[complement[inverse[E]], IMAGE[FIRST], IMAGE[FIRST],
            IMAGE[id[composite[Di, SECOND]]]], composite[complement[inverse[E]], IMAGE[SECOND],
            IMAGE[FIRST], IMAGE[id[composite[Di, FIRST]]]]] := composite[inverse[E], IDS]
```

Corollary. The function **IDS** is total.

```
In[30]:= Map[equal[IDS, #] &, IDS // FastReifNormality] // Reverse

Out[30]= equal[V, domain[IDS]] == True

In[31]:= domain[IDS] := V
```

---

range[IDS]

Lemma.

```
In[60]:= SubstTest[APPLY, IMAGE[thinpart[t]], setpart[x],
           t -> composite[id[inverse[DUP]], inverse[SECOND]]] // Reverse

Out[60]= A[intersection[image[inverse[IMAGE[SECOND]], set[setpart[x]]], P[inverse[DUP]]]] ==
          composite[id[setpart[x]], inverse[DUP]]

In[61]:= A[intersection[image[inverse[IMAGE[SECOND]], set[setpart[x_]]], P[inverse[DUP]]]] :=
          composite[id[setpart[x]], inverse[DUP]]
```

Lemma.

```
In[69]:= SubstTest[image, IMAGE[thinpart[t]], set[setpart[x]],
           t -> composite[id[inverse[DUP]], inverse[SECOND]]] // Reverse

Out[69]= intersection[image[inverse[IMAGE[SECOND]], set[setpart[x]]], P[inverse[DUP]]] ==
          set[composite[id[setpart[x]], inverse[DUP]]]

In[70]:= intersection[image[inverse[IMAGE[SECOND]], set[setpart[x_]]], P[inverse[DUP]]] :=
          set[composite[id[setpart[x]], inverse[DUP]]]
```

Theorem.

```
In[81]:= composite[IDS, id[P[inverse[DUP]]], inverse[IMAGE[SECOND]]] // FastReifNormality

Out[81]= composite[IDS, id[P[inverse[DUP]]], inverse[IMAGE[SECOND]]] == Id

In[82]:= composite[IDS, id[P[inverse[DUP]]], inverse[IMAGE[SECOND]]] := Id
```

Corollary.

```
In[84]:= ImageComp[IDS, composite[id[P[inverse[DUP]]], inverse[IMAGE[SECOND]]], V] // Reverse

Out[84]= image[IDS, P[inverse[DUP]]] == V
```

```
In[85]:= image[IDS, P[inverse[DUP]]] := V
```

Theorem.

```
In[94]:= Map[implies[#, equal[V, range[x]]] &, SubstTest[and,
            equal[V, u], subclass[u, v], {u -> image[x, y], v → range[x]}]] // Reverse

Out[94]= or[equal[V, range[x]], not[equal[V, image[x, y]]]] == True
```

```
In[95]:= or[equal[V, range[x_]], not[equal[V, image[x_, y_]]]] := True
```

Corollary.

```
In[97]:= SubstTest[implies, equal[image[x, y], V],
           equal[V, range[x]], {x → IDS, y → P[inverse[DUP]]}] // Reverse

Out[97]= equal[V, range[IDS]] == True
```

```
In[99]:= range[IDS] := V
```

---

composite rules

Theorem.

```
In[101]:=
        composite[IDS, IMAGE[id[cart[V, V]]]] // FastReifNormality

Out[101]=
        composite[IDS, IMAGE[id[cart[V, V]]]] == IDS
```

```
In[102]:=
        composite[IDS, IMAGE[id[cart[V, V]]]] := IDS
```

Theorem.

```
In[103]:=
        composite[IDS, IMAGE[id[cart[cart[V, V], V]]]] // FastReifNormality

Out[103]=
        composite[IDS, IMAGE[id[cart[cart[V, V], V]]]] == IDS
```

```
In[104]:=
        composite[IDS, IMAGE[id[cart[cart[V, V], V]]]] := IDS
```

Theorem. Flip rule for identities.

In[107]:=
```
composite[IDS, IMAGE[cross[SWAP, Id]]] // FastReifNormality
```

Out[107]=
```
composite[IDS, IMAGE[cross[SWAP, Id]]] == IDS
```

In[108]:=
```
composite[IDS, IMAGE[cross[SWAP, Id]]] := IDS
```

Theorem. Identities for a direct product.

In[110]:=
```
composite[IDS, IMAGE[cross[TWIST, Id]], CROSS] // FastReifTriNormality
```

Out[110]=
```
composite[IDS, IMAGE[cross[TWIST, Id]], CROSS] == composite[CART, cross[IDS, IDS]]
```

In[111]:=
```
composite[IDS, IMAGE[cross[TWIST, Id]], CROSS] := composite[CART, cross[IDS, IDS]]
```

---

ids[x] ⊂ fix[x ∘ DUP]

In this section a variable-free expression is derived for the fact that identity elements are idempotent.

Lemma. A simplification rule.

In[112]:=
```
composite[IMAGE[SECOND], IMAGE[id[inverse[DUP]]]] // FastReifNormality// Reverse
```

Out[112]=
```
composite[IMAGE[inverse[DUP]], IMAGE[cross[inverse[DUP], Id]]] ==
 composite[IMAGE[SECOND], IMAGE[id[inverse[DUP]]]]
```

In[113]:=
```
composite[IMAGE[inverse[DUP]], IMAGE[cross[inverse[DUP], Id]]] :=
 composite[IMAGE[SECOND], IMAGE[id[inverse[DUP]]]]
```

Observation.

In[114]:=
```
VERTSECT[reify[x, fix[composite[x, DUP]]]]
```

Out[114]=
```
composite[IMAGE[SECOND], IMAGE[id[inverse[DUP]]]]
```

Lemma.

In[115]:=
```
Map[empty[composite[Id, complement[#]]] &,
 dif[composite[inverse[E], IDS], composite[inverse[E], IMAGE[SECOND],
     IMAGE[id[inverse[DUP]]]]] // complement // FastReifNormality]
```

Out[115]=
```
subclass[composite[IMAGE[SECOND], IMAGE[id[inverse[DUP]]], inverse[IDS]], S] == True
```

In[116]:=
```
subclass[composite[IMAGE[SECOND], IMAGE[id[inverse[DUP]]], inverse[IDS]], S] := True
```

Corollary.

In[117]:=
```
(subclass[composite[inverse[E], funpart[u]], composite[inverse[E], funpart[v]]] //
   AssertTest) /. {u → IDS, v -> composite[IMAGE[SECOND], IMAGE[id[inverse[DUP]]]]}
```

Out[117]=
```
subclass[composite[inverse[E], IDS],
  composite[inverse[E], IMAGE[SECOND], IMAGE[id[inverse[DUP]]]]] == True
```

In[118]:=
```
subclass[composite[inverse[E], IDS],
  composite[inverse[E], IMAGE[SECOND], IMAGE[id[inverse[DUP]]]]] := True
```

Lemma.

In[119]:=
```
SubstTest[implies, subclass[u, v], subclass[composite[t, u], composite[t, v]],
  {t → IDS, u → Id, v -> composite[inverse[IMAGE[id[inverse[DUP]]]],
     inverse[IMAGE[SECOND]], IMAGE[SECOND], IMAGE[id[inverse[DUP]]]]}] // Reverse
```

Out[119]=
```
subclass[IDS, composite[IDS, inverse[IMAGE[id[inverse[DUP]]]],
   inverse[IMAGE[SECOND]], IMAGE[SECOND], IMAGE[id[inverse[DUP]]]]] == True
```

In[120]:=
```
% /. Equal→ SetDelayed
```

Theorem. A variable-free statement of the inclusion **ids[x] ⊂ fix[x ∘ DUP]**.

In[121]:=
```
SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
  {u → IDS, v -> composite[IDS, inverse[IMAGE[id[inverse[DUP]]]],
     inverse[IMAGE[SECOND]], IMAGE[SECOND], IMAGE[id[inverse[DUP]]]],
   w -> composite[inverse[S], IMAGE[SECOND], IMAGE[id[inverse[DUP]]]]}] // Reverse
```

Out[121]=
```
subclass[IDS, composite[inverse[S], IMAGE[SECOND], IMAGE[id[inverse[DUP]]]]] == True
```

In[122]:=
```
subclass[IDS, composite[inverse[S], IMAGE[SECOND], IMAGE[id[inverse[DUP]]]]] := True
```

the inclusion ids[x] ⊂ range[x]

Since **fix[x ∘ DUP] ⊂ range[x]**, the weaker inclusion **ids[x] ⊂ range[x]** also holds, and one can derive variable-free statements of that as well.

Lemma.

```
In[128]:=
Map[empty[composite[Id, complement[#]]] &,
 dif[composite[inverse[E], IMAGE[SECOND], IMAGE[id[inverse[DUP]]]],
    composite[inverse[E], IMAGE[SECOND]]] // complement // VSNormality]

Out[128]=
subclass[composite[IMAGE[SECOND],
   inverse[IMAGE[id[inverse[DUP]]]], inverse[IMAGE[SECOND]]], S] == True

In[129]:=
% /. Equal → SetDelayed
```

Theorem.

```
In[130]:=
subclass[composite[inverse[E], IMAGE[SECOND], IMAGE[id[inverse[DUP]]]],
  composite[inverse[E], IMAGE[SECOND]]] // AssertTest

Out[130]=
subclass[composite[inverse[E], IMAGE[SECOND], IMAGE[id[inverse[DUP]]]],
  composite[inverse[E], IMAGE[SECOND]]] == True

In[131]:=
subclass[composite[inverse[E], IMAGE[SECOND], IMAGE[id[inverse[DUP]]]],
  composite[inverse[E], IMAGE[SECOND]]] := True
```

Corollary

```
In[133]:=
SubstTest[implies, and[subclass[u, v], subclass[v, w]],
  subclass[u, w], {u → composite[inverse[E], IDS],
   v -> composite[inverse[E], IMAGE[SECOND], IMAGE[id[inverse[DUP]]]],
   w -> composite[inverse[E], IMAGE[SECOND]]}] // Reverse

Out[133]=
subclass[composite[inverse[E], IDS], composite[inverse[E], IMAGE[SECOND]]] == True

In[134]:=
subclass[composite[inverse[E], IDS], composite[inverse[E], IMAGE[SECOND]]] := True
```

The following additional corollary is now recognized automatically via conditional rewrite rules, and need not be added as a separate rewrite rule.

In[136]:=

```
subclass[IDS, composite[inverse[S], IMAGE[SECOND]]]
```

Out[136]=

```
True
```