λ x. inv[x]

Johan G. F. Belinfante
2011 April 24

```
In[1]:= SetDirectory["l:"]; << goedel.11apr23a

        :Package Title: goedel.11apr23a               2011 April 23 at 3:35 p.m.

        It is now:  2011 Apr 24 at 14:14

        Loading Simplification Rules

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

        weightlimit = 40
```

summary

The function **INV = λ x. inv[x]** is introduced and some of its properties are derived.

definition

The class **INV** is defined by the following membership rule.

```
In[2]:= member[x_, INV] := and[member[first[x], V], equal[second[x], inv[first[x]]]]
```

Lemma. Simplification rule.

```
In[3]:= SubstTest[intersection, image[inverse[t], ids[t]],
          inverse[image[inverse[t], ids[t]]], t -> union[x, complement[image[V, set[y]]]]]

Out[3]= inv[union[x, complement[image[V, set[y]]]]] == composite[id[image[V, set[y]]], inv[x]]

In[4]:= inv[union[x_, complement[image[V, set[y_]]]]] := composite[id[image[V, set[y]]], inv[x]]
```

Theorem. Vertical section rule.

```
In[5]:= image[INV, set[x]] // Normality

Out[5]= image[INV, set[x]] == intersection[image[V, set[x]], set[inv[x]]]

In[6]:= image[INV, set[x_]] := intersection[image[V, set[x]], set[inv[x]]]
```

Lemma.

```
In[7]:= Map[equal[V, #] &, complement[dif[INV, cart[V, V]]] // Normality]

Out[7]= subclass[INV, cart[V, V]] == True

In[8]:= % /. Equal → SetDelayed
```

Lemma.

```
In[9]:= equal[composite[Id, INV], INV]

Out[9]= True

In[10]:= composite[Id, INV] := INV
```

Theorem.

```
In[11]:= Map[equal[V, #] &, SubstTest[class, x,
           member[image[u, set[x]], v], {u → INV, v → union[set[0], range[SINGLETON]]}]]

Out[11]= FUNCTION[INV] == True

In[12]:= FUNCTION[INV] := True
```

---

symmetry

Lemma.

```
In[13]:= symdif[INV,
           composite[CORE[SYM], IMG, intersection[composite[inverse[FIRST], IMAGE[SWAP]],
             composite[inverse[SECOND], IDS]]]] // VSNormality

Out[13]= union[intersection[INV, composite[complement[CORE[SYM]], IMG, intersection[
              composite[inverse[FIRST], IMAGE[SWAP]], composite[inverse[SECOND], IDS]]]],
            intersection[complement[INV], composite[CORE[SYM], IMG, intersection[
              composite[inverse[FIRST], IMAGE[SWAP]], composite[inverse[SECOND], IDS]]]]] == 0

In[14]:= % /. Equal → SetDelayed
```

Theorem. Normalization rule.

```
In[15]:= SubstTest[empty, symdif[u, v],
           {u → INV, v -> composite[CORE[SYM], IMG, intersection[composite[
                inverse[FIRST], IMAGE[SWAP]], composite[inverse[SECOND], IDS]]]}]

Out[15]= equal[INV, composite[CORE[SYM], IMG, intersection[
             composite[inverse[FIRST], IMAGE[SWAP]], composite[inverse[SECOND], IDS]]]] == True

In[16]:= composite[CORE[SYM], IMG, intersection[
             composite[inverse[FIRST], IMAGE[SWAP]], composite[inverse[SECOND], IDS]]] := INV
```

Corollary.

```
In[17]:= Assoc[CORE[SYM], CORE[SYM], composite[IMG, intersection[
           composite[inverse[FIRST], IMAGE[SWAP]], composite[inverse[SECOND], IDS]]]]
```

```
Out[17]= composite[CORE[SYM], INV] == INV
```

```
In[18]:= composite[CORE[SYM], INV] := INV
```

Corollary.

```
In[19]:= Assoc[IMAGE[SWAP], CORE[SYM], INV]
```

```
Out[19]= composite[IMAGE[SWAP], INV] == INV
```

```
In[20]:= composite[IMAGE[SWAP], INV] := INV
```

Corollary.

```
In[21]:= Assoc[IMAGE[SECOND], IMAGE[SWAP], INV]
```

```
Out[21]= composite[IMAGE[SECOND], INV] == composite[IMAGE[FIRST], INV]
```

```
In[22]:= composite[IMAGE[SECOND], INV] := composite[IMAGE[FIRST], INV]
```

---

an upper bound

Observation.

```
In[23]:= subclass[inv[x], domain[x]]
```

```
Out[23]= True
```

Lemma.

```
In[24]:= SubstTest[implies, and[subclass[u, v], member[v, V]],
           member[u, V], {u → inv[setpart[x]], v → domain[setpart[x]]}] // Reverse
```

```
Out[24]= member[inv[setpart[x]], V] == True
```

```
In[25]:= member[inv[setpart[x_]], V] := True
```

Theorem.

```
In[26]:= SubstTest[class, x, member[image[u, set[setpart[x]]], v],
           {u → INV, v → range[SINGLETON]}]
```

```
Out[26]= domain[INV] == V
```

```
In[27]:= domain[INV] := V
```

Lemma.

```
In[28]:= SubstTest[class, x, subclass[image[u, set[setpart[x]]], image[v, set[setpart[x]]]],
          {u -> composite[inverse[E], INV], v -> composite[inverse[E], IMAGE[FIRST]]}]

Out[28]= fix[composite[inverse[IMAGE[FIRST]], S, INV]] == V

In[29]:= fix[composite[inverse[IMAGE[FIRST]], S, INV]] := V
```

Theorem. A variable-free formulation of the inclusion **inv[x] ⊂ domain[x]**.

```
In[30]:= SubstTest[implies, subclass[u, v],
           subclass[image[t, u], image[t, v]], {t → composite[SWAP, cross[INV, Id]],
            u → Id, v -> composite[inverse[IMAGE[FIRST]], S, INV]}] // Reverse

Out[30]= subclass[composite[inverse[E], INV], composite[inverse[E], IMAGE[FIRST]]] == True

In[31]:= subclass[composite[inverse[E], INV], composite[inverse[E], IMAGE[FIRST]]] := True
```

flip rule

A variable-free formulation of the following will now be derived:

```
In[32]:= inv[flip[x]]

Out[32]= inv[x]
```

Lemma.

```
In[33]:= symdif[composite[INV, IMAGE[cross[SWAP, Id]]], INV] // VSNormality

Out[33]= union[intersection[INV, composite[complement[INV], IMAGE[cross[SWAP, Id]]]],
           intersection[complement[INV], composite[INV, IMAGE[cross[SWAP, Id]]]]] == 0

In[34]:= % /. Equal→ SetDelayed
```

Theorem.

```
In[35]:= SubstTest[empty, symdif[u, v], {u -> composite[INV, IMAGE[cross[SWAP, Id]]], v -> INV}]

Out[35]= equal[INV, composite[INV, IMAGE[cross[SWAP, Id]]]] == True

In[36]:= composite[INV, IMAGE[cross[SWAP, Id]]] := INV
```

direct products

Lemma.

```
In[41]:= symdif[composite[INV, IMAGE[cross[TWIST, Id]], CROSS],
           composite[CROSS, cross[INV, INV]]] // VSTriNormality

Out[41]= union[composite[intersection[composite[CROSS, cross[INV, INV]],
             composite[complement[INV], IMAGE[cross[TWIST, Id]], CROSS]], id[cart[V, V]]],
           composite[intersection[composite[complement[CROSS], cross[INV, INV]],
             composite[INV, IMAGE[cross[TWIST, Id]], CROSS]], id[cart[V, V]]]] == 0

In[42]:= % /. Equal→ SetDelayed
```

Theorem. Direct product rule for **INV**.

```
In[43]:= SubstTest[empty, composite[symdif[u, v], id[cart[V, V]]],
          {u -> composite[INV, IMAGE[cross[TWIST, Id]], CROSS],
           v -> composite[CROSS, cross[INV, INV]]}]

Out[43]= equal[composite[CROSS, cross[INV, INV]],
           composite[INV, IMAGE[cross[TWIST, Id]], CROSS]] == True

In[45]:= composite[INV, IMAGE[cross[TWIST, Id]], CROSS] := composite[CROSS, cross[INV, INV]]
```

---

an example

In this section a variable-free formulation of the following special rule is derived.

```
In[46]:= inv[composite[id[x], inverse[DUP]]]

Out[46]= id[x]
```

Lemma. Simplification rule.

```
In[47]:= SubstTest[image, IMAGE[composite[id[t], inverse[SECOND]]],
           set[x], t → inverse[DUP]] // Reverse

Out[47]= intersection[image[inverse[IMAGE[SECOND]], set[x]], P[inverse[DUP]]] ==
          set[composite[id[x], inverse[DUP]]]

In[48]:= intersection[image[inverse[IMAGE[SECOND]], set[x_]], P[inverse[DUP]]] :=
          set[composite[id[x], inverse[DUP]]]
```

Theorem.

```
In[49]:= composite[INV, id[P[inverse[DUP]]], inverse[IMAGE[SECOND]]] // VSNormality

Out[49]= composite[INV, id[P[inverse[DUP]]], inverse[IMAGE[SECOND]]] == IMAGE[DUP]

In[50]:= composite[INV, id[P[inverse[DUP]]], inverse[IMAGE[SECOND]]] := IMAGE[DUP]
```

Corollary.

```
In[51]:= ImageComp[INV, composite[id[P[inverse[DUP]]], inverse[IMAGE[SECOND]]], V] // Reverse

Out[51]= image[INV, P[inverse[DUP]]] == P[Id]

In[52]:= image[INV, P[inverse[DUP]]] := P[Id]
```

APPLY and inverse image rules

Lemma.

```
In[62]:= SubstTest[member, composite[Id, t], V, t → inv[x]]

Out[62]= member[domain[inv[x]], V] == member[inv[x], V]

In[63]:= member[domain[inv[x_]], V] := member[inv[x], V]
```

Theorem.

```
In[68]:= Map[implies[member[x, y], #] &,
           SubstTest[implies, equal[x, setpart[t]], member[inv[x], V], t → x]] // Reverse

Out[68]= or[member[inv[x], V], not[member[x, y]]] == True

In[69]:= or[member[inv[x_], V], not[member[x_, y_]]] := True
```

Theorem. An **APPLY** rule for **INV**.

```
In[73]:= Map[complement[complement[#]] &, SubstTest[A, image[t, set[x]], t → INV]]

Out[73]= APPLY[INV, x] == union[complement[image[V, set[x]]], inv[x]]

In[74]:= APPLY[INV, x_] := union[complement[image[V, set[x]]], inv[x]]
```

Theorem. An inverse image rule for **INV**.

```
In[75]:= (member[x, image[inverse[funpart[t]], y]] // AssertTest) /. t → INV

Out[75]= member[x, image[inverse[INV], y]] == and[member[x, V], member[inv[x], y]]

In[76]:= member[x_, image[inverse[INV], y_]] := and[member[x, V], member[inv[x], y]]
```

image[INV, CATS] and image[INV, GROUPS]

Lemma.

```
In[78]:= implies[member[x, CATS], member[x, image[inverse[INV], BIJ]]] // NotNotTest

Out[78]= or[and[FUNCTION[inv[x]], member[inv[x], V]], not[category[x]], not[member[x, V]]] == True

In[79]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem.

```
In[81]:= Map[equal[V, #] &, SubstTest[class, x,
           implies[member[x, u], member[x, v]], {u → CATS, v -> image[inverse[INV], BIJ]}]]
```

```
Out[81]= subclass[image[INV, CATS], BIJ] == True
```

```
In[82]:= subclass[image[INV, CATS], BIJ] := True
```

Corollary.

```
In[84]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
           {u → GROUPS, v → CATS, w → image[inverse[INV], BIJ]}] // Reverse
```

```
Out[84]= subclass[image[INV, GROUPS], BIJ] == True
```

```
In[85]:= subclass[image[INV, GROUPS], BIJ] := True
```

Theorem.

```
In[87]:= Map[subclass[#, SYM] &, ImageComp[CORE[SYM], INV, V]]
```

```
Out[87]= subclass[range[INV], SYM] == True
```

```
In[88]:= subclass[range[INV], SYM] := True
```

Corollary.

```
In[91]:= SubstTest[subclass, t, intersection[u, v],
           {t -> image[INV, CATS], u -> BIJ, v → SYM}] // Reverse
```

```
Out[91]= subclass[image[INV, CATS], INVOL] == True
```

```
In[92]:= subclass[image[INV, CATS], INVOL] := True
```

Corollary.

```
In[93]:= SubstTest[subclass, t, intersection[u, v],
           {t -> image[INV, GROUPS], u -> BIJ, v → SYM}] // Reverse
```

```
Out[93]= subclass[image[INV, GROUPS], INVOL] == True
```

```
In[94]:= subclass[image[INV, GROUPS], INVOL] := True
```