IPD preserves inverses for bijections

Johan G. F. Belinfante
2009 September 6

```
In[1]:= SetDirectory["l:"]; << goedel.09sep05a; << tools.m

        :Package Title: goedel.09sep05a              2009 September 5 at 7:30 p.m.

        It is now:  2009 Sep 6 at 14:7

        Loading Simplification Rules

        TOOLS.M                                    Revised 2009 September 3

        weightlimit = 40
```

summary

The function **IPD** that maps any set **x** to the restriction of **IMAGE[x]** to the power set of the domain of **x** sends relations to functions, inverse functions to bijections, involutions to involutions, and identity functions to identity functions. Although **IPD** does not preserve inverses in general, it does preserve inverses for bijections.

temporary definition

The following temporary abbreviation will be used in this notebook.

```
In[2]:= f[x_] := composite[IMAGE[x], id[P[domain[x]]]]
```

If **x** is a set, the proper class function **IMAGE[x]** factors as the product of the set function **f[x]** and the proper class function **IMAGE[id[domain[x]]]**.

Observation. Applying **IPD** to a set **x** yields **f[x]**.

```
In[3]:= APPLY[IPD, setpart[x]] == f[setpart[x]]

Out[3]= True
```

preservation of inverses for bijections

Counterexample. In general, the constructor **f** does not preserve inverses.

```
In[4]:= (equal[f[inverse[x]], inverse[f[x]]] /. x → cartsq[succ[set[0]]]) // AssertTest

Out[4]= equal[union[cart[intersection[complement[set[0]], P[succ[set[0]]]], set[succ[set[0]]]],
          cart[set[0], set[0]]], union[cart[set[0], set[0]],
          cart[set[succ[set[0]]], intersection[complement[set[0]], P[succ[set[0]]]]]]] == False

In[5]:= % /. Equal→ SetDelayed
```

Theorem. The functions **IPD** and **INVERSE** do not commute.

```
In[6]:= Map[not, SubstTest[implies, equal[u, v],
          equal[image[u, w], image[v, w]], {u → composite[INVERSE, IPD],
           v → composite[IPD, INVERSE], w → set[cartsq[succ[set[0]]]]}]] // Reverse

Out[6]= equal[composite[INVERSE, IPD], composite[IPD, INVERSE]] == False

In[7]:= equal[composite[INVERSE, IPD], composite[IPD, INVERSE]] := False
```

Theorem. Rule for inverses of bijections.

```
In[8]:= Assoc[IMAGE[inverse[oopart[x]]], composite[IMAGE[oopart[x]], id[P[domain[oopart[x]]]]],
          inverse[IMAGE[oopart[x]]]] // Reverse

Out[8]= composite[id[P[domain[oopart[x]]]], inverse[IMAGE[oopart[x]]]] ==
         composite[IMAGE[inverse[oopart[x]]], id[P[range[oopart[x]]]]]

In[9]:= composite[id[P[domain[oopart[x_]]]], inverse[IMAGE[oopart[x_]]]] :=
         composite[IMAGE[inverse[oopart[x]]], id[P[range[oopart[x]]]]]
```

Restatement.

```
In[10]:= f[inverse[oopart[x]]] == inverse[f[oopart[x]]]

Out[10]= True
```

Since the inverse of a bijection is also a bijection, the following corollary is obtained.

Corollary.

```
In[11]:= SubstTest[composite, id[P[domain[oopart[t]]]],
           inverse[IMAGE[oopart[t]]], t → inverse[oopart[x]]] // Reverse

Out[11]= composite[id[P[range[oopart[x]]]], inverse[IMAGE[inverse[oopart[x]]]]] ==
          composite[IMAGE[oopart[x]], id[P[domain[oopart[x]]]]]

In[12]:= composite[id[P[range[oopart[x_]]]], inverse[IMAGE[inverse[oopart[x_]]]]] :=
          composite[IMAGE[oopart[x]], id[P[domain[oopart[x]]]]]
```

The variable **x** can be eliminated.

Lemma. Simplification rule.

```
In[13]:= Map[VERTSECT, SubstTest[reify, x, inverse[f[inverse[h[x]]]], h → oopart]]

Out[13]= composite[
           VERTSECT[intersection[composite[inverse[rotate[composite[IMG, SWAP]]], INVERSE],
             composite[inverse[SECOND], inverse[S], IMAGE[SECOND]]]],
           OOPART] == composite[IPD, OOPART]

In[14]:= % /. Equal → SetDelayed
```

The orientation of the following rewrite rule is tentative.

Theorem. The function **IPD** preserves inverses for bijections.

```
In[15]:= composite[IPD, INVERSE, OOPART] // FastReifNormality

Out[15]= composite[IPD, INVERSE, OOPART] == composite[INVERSE, IPD, OOPART]

In[16]:= composite[IPD, INVERSE, OOPART] := composite[INVERSE, IPD, OOPART]
```

Corollary.

```
In[17]:= Assoc[composite[IPD, INVERSE], OOPART, inverse[OOPART]]

Out[17]= composite[IPD, INVERSE, id[BIJ]] == composite[INVERSE, IPD, id[BIJ]]

In[18]:= composite[IPD, INVERSE, id[BIJ]] := composite[INVERSE, IPD, id[BIJ]]
```

Restatement. The function **INVERSE** commutes with the restriction of **IPD** to bijections.

```
In[19]:= commute[INVERSE, composite[IPD, id[BIJ]]]

Out[19]= True
```

inverse functions

The function **f[x]** is one-to-one when **x** is an inverse function.

```
In[20]:= ONEONE[f[inverse[funpart[x]]]]

Out[20]= True
```

A variable-free statement is possible.

Theorem.

```
In[21]:= Map[empty, composite[id[complement[BIJ]], IPD, INVERSE, FUNPART] // FastReifNormality]

Out[21]= subclass[image[IPD, image[INVERSE, FUNS]], BIJ] == True

In[22]:= subclass[image[IPD, image[INVERSE, FUNS]], BIJ] := True
```

Corollary. The function **IPD** maps bijections to bijections.

```
In[23]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
           {u → BIJ, v → image[INVERSE, FUNS], w → image[inverse[IPD], BIJ]}] // Reverse

Out[23]= subclass[image[IPD, BIJ], BIJ] == True

In[24]:= subclass[image[IPD, BIJ], BIJ] := True
```

An **involution** is a function that is its own inverse.

Corollary. The function **IPD** sends involutions to involutions.

```
In[25]:= SubstTest[implies, subcommute[funpart[x], y], invariant[funpart[x], fix[y]],
           {x → composite[IPD, id[BIJ]], y → composite[INVERSE, id[BIJ]]}] // Reverse

Out[25]= subclass[image[IPD, INVOL], INVOL] == True

In[26]:= subclass[image[IPD, INVOL], INVOL] := True
```

relative complement functions

The function **RC[x]** that sends each subset of **x** to its relative complement in **x** is an involution. The class **range[RCF]** of all such functions is a subclass of the class **INVOL** of involutions.

Theorem.

```
In[27]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
           {u → range[RCF], v → INVOL, w → image[inverse[IPD], INVOL]}] // Reverse

Out[27]= subclass[image[IPD, range[RCF]], INVOL] == True

In[28]:= subclass[image[IPD, range[RCF]], INVOL] := True
```

Theorem. Since **RC[x]** is its own inverse, the same holds for **f[RC[x]]**.

```
In[29]:= Assoc[composite[IPD, INVERSE], OOPART, RCF] // Reverse

Out[29]= composite[INVERSE, IPD, RCF] == composite[IPD, RCF]

In[30]:= composite[INVERSE, IPD, RCF] := composite[IPD, RCF]
```

Theorem. An explicit formula for **inverse[f[RC[setpart[x]]]]**.

```
In[31]:= ApComp[composite[INVERSE, IPD], RCF, setpart[x]]

Out[31]= composite[id[P[P[setpart[x]]]], inverse[IMAGE[RC[setpart[x]]]]] ==
          composite[IMAGE[RC[setpart[x]]], id[P[P[setpart[x]]]]]

In[32]:= composite[id[P[P[setpart[x_]]]], inverse[IMAGE[RC[setpart[x_]]]]] :=
          composite[IMAGE[RC[setpart[x]]], id[P[P[setpart[x]]]]]
```

Comment. If **x** is a proper class, then **RC[x] = 0** is an involution, and so is **f[0] = 0**, but nevertheless the above explicit formula does not hold without the **setpart** wrapper.

f[inverse[funpart[x]]]

In this section some additional results for inverse functions are derived.

Lemma. If **x** is a set function, then **f[x] ∘ f[inverse[x]] ⊂ Id**.

```
In[33]:= composite[f[funpart[setpart[x]]], f[inverse[funpart[setpart[x]]]]] //
          FastReifNormality

Out[33]= composite[IMAGE[funpart[setpart[x]]], id[P[domain[funpart[setpart[x]]]]],
           IMAGE[inverse[funpart[setpart[x]]]], id[P[range[funpart[setpart[x]]]]]] ==
          id[P[range[funpart[setpart[x]]]]]

In[34]:= composite[IMAGE[funpart[setpart[x_]]],
           id[P[domain[funpart[setpart[x_]]]]], IMAGE[inverse[funpart[setpart[x_]]]],
           id[P[range[funpart[setpart[x_]]]]]] := id[P[range[funpart[setpart[x]]]]]
```

Theorem. An explicit formula for **f[inverse[funpart[x]]]** for the case that **x** is a set.

```
In[35]:= Assoc[f[funpart[setpart[x]]], f[inverse[funpart[setpart[x]]]],
           inverse[f[inverse[funpart[setpart[x]]]]]] // Reverse

Out[35]= composite[id[P[range[funpart[setpart[x]]]]],
           inverse[IMAGE[inverse[funpart[setpart[x]]]]] == composite[
           IMAGE[funpart[setpart[x]]], id[range[IMAGE[inverse[funpart[setpart[x]]]]]]

In[36]:= composite[id[P[range[funpart[setpart[x_]]]]],
           inverse[IMAGE[inverse[funpart[setpart[x_]]]]]] := composite[
           IMAGE[funpart[setpart[x]]], id[range[IMAGE[inverse[funpart[setpart[x]]]]]]
```

Observation. If **x** is a set function, then **f[inverse[x]] ⊂ inverse[f[x]]**.

```
In[37]:= subclass[f[inverse[funpart[setpart[x]]]], inverse[f[funpart[setpart[x]]]]]

Out[37]= True
```

The variable **x** can be eliminated.

Lemma.

```
In[39]:= equal[composite[IMAGE[funpart[setpart[x]]],
           id[intersection[complement[P[domain[funpart[setpart[x]]]]],
             range[IMAGE[inverse[funpart[setpart[x]]]]]]]], 0]

Out[39]= True
```

```
In[41]:= composite[IMAGE[funpart[setpart[x_]]],
           id[intersection[complement[P[domain[funpart[setpart[x_]]]]],
             range[IMAGE[inverse[funpart[setpart[x_]]]]]]]] := 0
```

Theorem. A variable-free reformulation.

```
In[45]:= Map[empty[composite[inverse[S], #, inverse[FUNPART]]] &,
          SubstTest[reify, x, inverse[dif[APPLY[funpart[t], inverse[funpart[setpart[x]]]],
            inverse[APPLY[funpart[t], funpart[setpart[x]]]]], t → IPD]]

Out[45]= subclass[composite[IPD, id[FUNS], INVERSE, inverse[IPD], INVERSE], S] == True

In[46]:= subclass[composite[IPD, id[FUNS], INVERSE, inverse[IPD], INVERSE], S] := True
```

Lemma. Simpification rule.

```
In[52]:= composite[id[FUNS], inverse[IMAGE[id[cart[V, V]]]]] // DoubleInverse

Out[52]= composite[id[FUNS], inverse[IMAGE[id[cart[V, V]]]]] == id[FUNS]

In[53]:= composite[id[FUNS], inverse[IMAGE[id[cart[V, V]]]]] := id[FUNS]
```

Corollary.

```
In[57]:= SubstTest[implies, subclass[u, v], subclass[composite[t, u, w], composite[t, v, w]],
           {t → INVERSE, u→ inverse[composite[IPD, id[FUNS], INVERSE, inverse[IPD], INVERSE]],
            v → inverse[S], w -> composite[IPD, id[P[cart[V, V]]]]}] // Reverse

Out[57]= subclass[composite[IPD, INVERSE, id[FUNS]], composite[inverse[S], INVERSE, IPD]] == True

In[58]:= subclass[composite[IPD, INVERSE, id[FUNS]], composite[inverse[S], INVERSE, IPD]] := True
```