cancellation laws for functions

Johan G. F. Belinfante
2008 August 22

```
In[1]:= SetDirectory["l:"]; << goedel.08aug20a;<< tools.m
```

```
:Package Title: goedel.08aug20a                2008 August 20 at 11:55 p.m.

It is now:  2008 Aug 22 at 10:52

Loading Simplification Rules

TOOLS.M                                    Revised 2008 July 5

weightlimit = 40
```

summary

In this notebook several cancellation laws for functions are derived. Each of these cancellation laws has the following form:

$$(\forall\, f, g \in \mathbf{FUNS}) \qquad r \circ f = r \circ g \;\Leftrightarrow\; f = g,$$

where the relation **r** can be **DISJOINT, S, inverse[S],** etc. For these first three cases, tools are available that exploit the cancellation law:

```
In[2]:= Begin["Goedel`Private`"];
```

```
In[3]:= ?? CancelDJ
```

```
CancelDJ[x] attempts to cancel a factor of DISJOINT

CancelDJ[x_] := Module[{w = composite[DISJOINT, funpart[x]]},
  VERTSECT[complement[composite[inverse[E], w]]] == funpart[x]]
```

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

```
CancelSR[x] attempts to cancel a factor of S

CancelSR[x_] := Module[{w = composite[S, funpart[x]]},
  VERTSECT[complement[composite[complement[inverse[E]], w]]] == funpart[x]]
```

```
In[5]:= ?? CancelSI
```

```
CancelSI[x] attempts to cancel a factor of inverse[S]

CancelSI[x_] := Module[{w = composite[inverse[S], funpart[x]]},
  composite[VERTSECT[composite[inverse[E], w]], id[domain[funpart[x]]]] == funpart[x]]
```

The proofs of the cancellation laws for each of these three cases are based on the **VERTSECT** functions that occur in the corresponding tool, but the tools themselves are not used here. In this notebook, the explicit hypotheses that **f** and **g** be

functions are avoided by using **funpart** wrappers. The main advantage of doing so is that the cancellation laws can then be expressed as simple rewrite rules.

Comment. There is also a cancellation law for **E** and it is not limited to functions.

```
In[7]:= equal[composite[E, x], composite[E, y]]

Out[7]= equal[composite[Id, x], composite[Id, y]]
```

temporary rewrite rule

Technical Lemma. (Temporary rewrite rule.)

```
In[8]:= SubstTest[equal, VERTSECT[complement[composite[inverse[E], u]]],
          VERTSECT[complement[composite[inverse[E], v]]],
          {u → composite[DISJOINT, funpart[x]], v → composite[DISJOINT, funpart[y]]}]

Out[8]= equal[composite[BIGCUP, IMAGE[funpart[x]], id[P[domain[funpart[x]]]]],
          composite[BIGCUP, IMAGE[funpart[y]], id[P[domain[funpart[y]]]]]] ==
         equal[funpart[x], funpart[y]]

In[9]:= equal[composite[BIGCUP, IMAGE[funpart[x_]], id[P[domain[funpart[x_]]]]],
          composite[BIGCUP, IMAGE[funpart[y_]], id[P[domain[funpart[y_]]]]]] :=
         equal[funpart[x], funpart[y]]
```

Comment. This rewrite rule suffices for each of the three cancellation laws for **DISJOINT**, **S** and **inverse[S]**.

cancelling DISJOINT

Lemma.

```
In[10]:= SubstTest[implies, equal[u, v], equal[VERTSECT[complement[composite[inverse[E], u]]],
           VERTSECT[complement[composite[inverse[E], v]]]],
          {u → composite[DISJOINT, funpart[x]], v → composite[DISJOINT, funpart[y]]}] // Reverse

Out[10]= or[equal[funpart[x], funpart[y]],
          not[equal[composite[DISJOINT, funpart[x]], composite[DISJOINT, funpart[y]]]]] == True

In[11]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. (Cancelling a factor of **DISJOINT**.)

```
In[12]:= equiv[equal[composite[DISJOINT, funpart[x]], composite[DISJOINT, funpart[y]]],
          equal[funpart[x], funpart[y]]]

Out[12]= True
```

```
In[13]:= equal[composite[DISJOINT, funpart[x_]], composite[DISJOINT, funpart[y_]]] :=
          equal[funpart[x], funpart[y]]
```

cancelling S

Lemma.

```
In[14]:= SubstTest[implies, equal[u, v],
           equal[VERTSECT[complement[composite[complement[inverse[E]], u]]],
            VERTSECT[complement[composite[complement[inverse[E]], v]]]],
           {u → composite[S, funpart[x]], v → composite[S, funpart[y]]}] // Reverse

Out[14]= or[equal[funpart[x], funpart[y]],
           not[equal[composite[S, funpart[x]], composite[S, funpart[y]]]]] == True

In[15]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. (Cancelling a factor of **S**.)

```
In[16]:= equiv[equal[composite[S, funpart[x]], composite[S, funpart[y]]],
          equal[funpart[x], funpart[y]]]

Out[16]= True

In[17]:= equal[composite[S, funpart[x_]], composite[S, funpart[y_]]] :=
          equal[funpart[x], funpart[y]]
```

cancelling inverse[S]

Lemma.

```
In[18]:= SubstTest[implies, equal[u, v],
           equal[composite[VERTSECT[composite[inverse[E], u]], id[domain[u]]],
            composite[VERTSECT[composite[inverse[E], v]], id[domain[v]]]],
           {u → composite[inverse[S], funpart[x]], v → composite[inverse[S], funpart[y]]}] //
          Reverse

Out[18]= or[equal[funpart[x], funpart[y]], not[
            equal[composite[inverse[S], funpart[x]], composite[inverse[S], funpart[y]]]]] == True

In[19]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. (Cancelling a factor of **inverse[S]**.)

```
In[20]:= equiv[equal[composite[inverse[S], funpart[x]], composite[inverse[S], funpart[y]]],
          equal[funpart[x], funpart[y]]]

Out[20]= True
```

```
In[21]:= equal[composite[inverse[S], funpart[x_]], composite[inverse[S], funpart[y_]]] :=
          equal[funpart[x], funpart[y]]
```

a useful rewrite rule

A more primitive way to derive cancellation laws uses **AssertTest**. For all of the applications considered in the next section, one needs the following rewrite rule:

Theorem.

```
In[22]:= Map[assert, SubstTest[equal, complement[u], complement[v],
            {u → composite[inverse[S], funpart[x]], v → composite[inverse[S], funpart[y]]}]] //
          Reverse

Out[22]= and[equal[domain[funpart[x]], domain[funpart[y]]],
           subclass[composite[funpart[x], inverse[funpart[y]]], S],
           subclass[composite[funpart[y], inverse[funpart[x]]], S]] ==
          equal[funpart[x], funpart[y]]

In[23]:= and[equal[domain[funpart[x_]], domain[funpart[y_]]],
           subclass[composite[funpart[x_], inverse[funpart[y_]]], S],
           subclass[composite[funpart[y_], inverse[funpart[x_]]], S]] :=
          equal[funpart[x], funpart[y]]
```

three more cancellation laws

Theorem. (Cancelling **complement[inverse[S]]**.)

```
In[24]:= equal[composite[complement[inverse[S]], funpart[x]],
           composite[complement[inverse[S]], funpart[y]]] // AssertTest

Out[24]= equal[composite[complement[inverse[S]], funpart[x]],
           composite[complement[inverse[S]], funpart[y]]] == equal[funpart[x], funpart[y]]
```

Theorem. (Cancelling **complement[inverse[E]]**.)

```
In[25]:= equal[composite[complement[inverse[S]], funpart[x_]],
           composite[complement[inverse[S]], funpart[y_]]] := equal[funpart[x], funpart[y]]

In[26]:= equal[composite[complement[inverse[E]], funpart[x]],
           composite[complement[inverse[E]], funpart[y]]] // AssertTest

Out[26]= equal[composite[complement[inverse[E]], funpart[x]],
           composite[complement[inverse[E]], funpart[y]]] == equal[funpart[x], funpart[y]]

In[27]:= equal[composite[complement[inverse[E]], funpart[x_]],
           composite[complement[inverse[E]], funpart[y_]]] := equal[funpart[x], funpart[y]]
```

Theorem. (Cancelling **Di**.)

```
In[41]:= equal[composite[Di, funpart[x]], composite[Di, funpart[y]]] // AssertTest

Out[41]= equal[composite[Di, funpart[x]], composite[Di, funpart[y]]] ==
          equal[funpart[x], funpart[y]]

In[42]:= equal[composite[Di, funpart[x_]], composite[Di, funpart[y_]]] :=
          equal[funpart[x], funpart[y]]
```

cancelling complement[E]

The final case also uses **AssertTest**, but a different lemma is needed.

Lemma.

```
In[54]:= Map[assert, SubstTest[equal, complement[u], complement[v],
            {u → composite[E, funpart[x]], v → composite[E, funpart[y]]}]] // Reverse

Out[54]= and[equal[domain[funpart[x]], domain[funpart[y]]],
           subclass[composite[funpart[x], inverse[funpart[y]]], Id],
           subclass[composite[funpart[y], inverse[funpart[x]]], Id]] ==
          equal[funpart[x], funpart[y]]

In[55]:= and[equal[domain[funpart[x_]], domain[funpart[y_]]],
           subclass[composite[funpart[x_], inverse[funpart[y_]]], Id],
           subclass[composite[funpart[y_], inverse[funpart[x_]]], Id]] :=
          equal[funpart[x], funpart[y]]
```

Theorem. (Cancelling **complement[E]**.)

```
In[56]:= equal[composite[complement[E], funpart[x]],
           composite[complement[E], funpart[y]]] // AssertTest

Out[56]= equal[composite[complement[E], funpart[x]], composite[complement[E], funpart[y]]] ==
          equal[funpart[x], funpart[y]]

In[57]:= equal[composite[complement[E], funpart[x_]], composite[complement[E], funpart[y_]]] :=
          equal[funpart[x], funpart[y]]
```