rotate[JOIN]

Johan G. F. Belinfante
2011 May 6

```
In[1]:= SetDirectory["l:"]; << goedel.11may05a

        :Package Title: goedel.11may05a              2011 May 5 at 11:05 a.m.

        It is now:  2011 May 6 at 14:45

        Loading Simplification Rules

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

        weightlimit = 40
```

summary

The function **JOIN** takes two lists to their concatenation. Concatenation of lists satisfy left and right cancellation laws. A variable-free statement of these cancellation laws is that **rotate[JOIN]** and **rotate[JOIN ∘ SWAP]** are functions.

rewrite rules for plus[x]

Two rewrite rules involving the function **plus[x]** are derived that are needed in the next section.

Theorem.

```
In[2]:= composite[id[x], plus[x]] // FastReifNormality

Out[2]= composite[id[x], plus[x]] == 0

In[3]:= composite[id[x_], plus[x_]] := 0
```

Corollary.

```
In[4]:= Assoc[x, id[domain[x]], plus[domain[x]]] // Reverse

Out[4]= composite[x, plus[domain[x]]] == 0

In[5]:= composite[x_, plus[domain[x_]]] := 0
```

recovering lists from their join

In this section it is shown how one recover each of two lists from their concatenation.

Theorem. Recovering the first of two joined lists.

```
In[6]:= Map[inverse[inverse[#]] &, SubstTest[composite, union[u, v], id[domain[list[x]]],
          {u → list[x], v -> composite[list[y], inverse[plus[domain[list[x]]]]]}] // Reverse

Out[6]= composite[APPLY[JOIN, PAIR[list[x], list[y]]], id[domain[list[x]]]] == list[x]

In[7]:= composite[APPLY[JOIN, PAIR[list[x_], list[y_]]], id[domain[list[x_]]] := list[x]
```

Theorem. Recovering the second of two joined lists.

```
In[8]:= SubstTest[composite, union[u, v], plus[domain[list[x]]],
          {u → list[x], v -> composite[list[y], inverse[plus[domain[list[x]]]]]}] // Reverse

Out[8]= composite[APPLY[JOIN, PAIR[list[x], list[y]]], plus[domain[list[x]]]] == list[y]

In[9]:= composite[APPLY[JOIN, PAIR[list[x_], list[y_]]], plus[domain[list[x_]]] := list[y]
```

cancellation laws

Cancellation laws are derived for joins of lists.

Theorem. Left cancellation law.

```
In[10]:= SubstTest[implies, equal[u, v],
           equal[composite[u, w], composite[v, w]], {u -> APPLY[JOIN, PAIR[list[x], list[y]]],
            v -> APPLY[JOIN, PAIR[list[x], list[z]]], w -> plus[domain[list[x]]]}] // Reverse

Out[10]= or[equal[list[y], list[z]], not[equal[
             APPLY[JOIN, PAIR[list[x], list[y]]], APPLY[JOIN, PAIR[list[x], list[z]]]]]] == True

In[11]:= or[equal[list[y_], list[z_]], not[equal[APPLY[JOIN, PAIR[list[x_], list[y_]]],
             APPLY[JOIN, PAIR[list[x_], list[z_]]]]]] := True
```

The right cancellation law requires a little more work.

Lemma. Right cancellation law.

```
In[12]:= SubstTest[implies, and[equal[s, t], equal[u, v]],
           equal[composite[s, u], composite[t, v]],
           {s -> APPLY[JOIN, PAIR[list[x], list[z]]], t -> APPLY[JOIN, PAIR[list[y], list[z]]],
            u → id[domain[list[x]]], v → id[domain[list[y]]]}] // Reverse

Out[12]= or[equal[list[x], list[y]],
           not[equal[APPLY[JOIN, PAIR[list[x], list[z]]], APPLY[JOIN, PAIR[list[y], list[z]]]]],
           not[equal[domain[list[x]], domain[list[y]]]] == True

In[13]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma.

```
In[14]:= SubstTest[implies, equal[s, t],
           equal[domain[s], domain[t]], {s -> APPLY[JOIN, PAIR[list[x], list[z]]],
            t -> APPLY[JOIN, PAIR[list[y], list[z]]]}] // Reverse

Out[14]= or[equal[domain[list[x]], domain[list[y]]], not[equal[
             APPLY[JOIN, PAIR[list[x], list[z]]], APPLY[JOIN, PAIR[list[y], list[z]]]]] == True

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

Theorem. Right cancellation law.

```
In[16]:= Map[not,
           SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3], not[implies[p1, p3]], {p1 ->
              equal[APPLY[JOIN, PAIR[list[x], list[z]]], APPLY[JOIN, PAIR[list[y], list[z]]]],
             p2 -> equal[domain[list[x]], domain[list[y]]],
             p3 -> equal[list[x], list[y]]}]] // Reverse

Out[16]= or[equal[list[x], list[y]], not[equal[
             APPLY[JOIN, PAIR[list[x], list[z]]], APPLY[JOIN, PAIR[list[y], list[z]]]]] == True

In[17]:= or[equal[list[x_], list[y_]], not[equal[APPLY[JOIN, PAIR[list[x_], list[z_]]],
             APPLY[JOIN, PAIR[list[y_], list[z_]]]]]] := True
```

JOIN ∘ LEFT[list[x]] is one-to-one

Lemma.

```
In[18]:= SubstTest[implies, and[equal[y, list[u]], equal[z, list[v]]],
           or[equal[y, z], not[equal[APPLY[JOIN, PAIR[list[x], y]],
              APPLY[JOIN, PAIR[list[x], z]]]]], {u → y, v → z}] // Reverse

Out[18]= or[equal[y, z],
           not[equal[APPLY[JOIN, PAIR[list[x], y]], APPLY[JOIN, PAIR[list[x], z]]]],
           not[FUNCTION[y]], not[FUNCTION[z]], not[member[domain[y], omega]],
           not[member[domain[z], omega]]] == True

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

Lemma.

```
In[20]:= Map[empty[composite[Id, complement[#]]] &, SubstTest[class, pair[y, z],
           implies[and[member[y, u], member[z, u], equal[APPLY[v, y], APPLY[v, z]]],
            equal[y, z]], {u → LISTS, v → composite[JOIN, LEFT[list[x]]]}]]

Out[20]= subclass[composite[inverse[LEFT[list[x]]], inverse[JOIN], JOIN, LEFT[list[x]]], Id] ==
          True

In[21]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem.

```
In[22]:= SubstTest[subclass, composite[t, inverse[t]],
          Id, t → composite[inverse[LEFT[list[x]]], inverse[JOIN]]]

Out[22]= FUNCTION[composite[inverse[LEFT[list[x]]], inverse[JOIN]]] == True

In[23]:= FUNCTION[composite[inverse[LEFT[list[x_]]], inverse[JOIN]]] := True
```

JOIN ∘ RIGHT[list[x]] is one-to-one

Lemma.

```
In[24]:= SubstTest[implies, and[equal[y, list[u]], equal[z, list[v]]],
           or[equal[y, z], not[equal[APPLY[JOIN, PAIR[y, list[x]]],
              APPLY[JOIN, PAIR[z, list[x]]]]]], {u → y, v → z}] // Reverse

Out[24]= or[equal[y, z],
           not[equal[APPLY[JOIN, PAIR[y, list[x]]], APPLY[JOIN, PAIR[z, list[x]]]]],
           not[FUNCTION[y]], not[FUNCTION[z]], not[member[domain[y], omega]],
           not[member[domain[z], omega]]] == True

In[25]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma.

```
In[26]:= Map[empty[composite[Id, complement[#]]] &, SubstTest[class, pair[y, z],
           implies[and[member[y, u], member[z, u], equal[APPLY[v, y], APPLY[v, z]]],
            equal[y, z]], {u → LISTS, v → composite[JOIN, RIGHT[list[x]]]}]]

Out[26]= subclass[composite[inverse[RIGHT[list[x]]], inverse[JOIN], JOIN, RIGHT[list[x]]], Id] ==
          True

In[27]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem.

```
In[28]:= SubstTest[subclass, composite[t, inverse[t]],
          Id, t → composite[inverse[RIGHT[list[x]]], inverse[JOIN]]]

Out[28]= FUNCTION[composite[inverse[RIGHT[list[x]]], inverse[JOIN]]] == True

In[29]:= FUNCTION[composite[inverse[RIGHT[list[x_]]], inverse[JOIN]]] := True
```

variable-free results

Lemma. A simplification rule.

```
In[30]:= composite[rotate[JOIN], id[cart[V, LISTS]]] // TripleRotate

Out[30]= composite[rotate[JOIN], id[cart[V, LISTS]]] == rotate[JOIN]

In[31]:= composite[rotate[JOIN], id[cart[V, LISTS]]] := rotate[JOIN]
```

Corollary.

```
In[32]:= SubstTest[funpart, composite[u, id[v]], {u → rotate[JOIN], v → cart[V, LISTS]}]

Out[32]= composite[funpart[rotate[JOIN]], id[cart[V, LISTS]]] == funpart[rotate[JOIN]]

In[33]:= % /. Equal → SetDelayed
```

Theorem. A variable-free statement of the left cancellation law.

```
In[34]:= Map[equal[rotate[JOIN], flip[rotate[inverse[#]]]] &,
          Map[composite[#, id[LISTS]] &, SubstTest[reify, x,
            funpart[composite[inverse[LEFT[list[x]]], t]], t -> inverse[JOIN]]]]

Out[34]= FUNCTION[rotate[JOIN]] == True

In[35]:= FUNCTION[rotate[JOIN]] := True
```

Lemma.

```
In[36]:= composite[FIRST, id[cart[V, LISTS]], intersection[composite[inverse[JOIN], FIRST],
            composite[inverse[SECOND], SECOND]]] // TripleRotate

Out[36]= composite[FIRST, id[cart[V, LISTS]],
           intersection[composite[inverse[JOIN], FIRST], composite[inverse[SECOND], SECOND]]] ==
          rotate[composite[JOIN, SWAP]]

In[37]:= % /. Equal → SetDelayed
```

Lemma. A simplification rule.

```
In[38]:= composite[rotate[composite[JOIN, SWAP]], id[cart[V, LISTS]]] // TripleRotate

Out[38]= composite[rotate[composite[JOIN, SWAP]], id[cart[V, LISTS]]] ==
          rotate[composite[JOIN, SWAP]]
```

```
In[39]:= composite[rotate[composite[JOIN, SWAP]], id[cart[V, LISTS]]] :=
          rotate[composite[JOIN, SWAP]]
```

Corollary.

```
In[40]:= SubstTest[funpart, composite[u, id[v]], {u → rotate[flip[JOIN]], v → cart[V, LISTS]}]

Out[40]= composite[funpart[rotate[composite[JOIN, SWAP]]], id[cart[V, LISTS]]] ==
          funpart[rotate[composite[JOIN, SWAP]]]

In[41]:= % /. Equal→ SetDelayed
```

Theorem. A variable-free statement of the right cancellation law.

```
In[42]:= Map[equal[rotate[composite[JOIN, SWAP]], flip[rotate[inverse[#]]]] &,
          Map[composite[#, id[LISTS]] &, SubstTest[reify, x,
            funpart[composite[inverse[LEFT[list[x]]], t]], t -> inverse[flip[JOIN]]]]]

Out[42]= FUNCTION[rotate[composite[JOIN, SWAP]]] == True

In[43]:= FUNCTION[rotate[composite[JOIN, SWAP]]] := True
```

serendipity

The following simplification rule was also discovered in the course of this work.

Theorem.

```
In[44]:= composite[IMAGE[id[cart[V, V]]], JOIN] // FastReifNormality

Out[44]= composite[IMAGE[id[cart[V, V]]], JOIN] == JOIN

In[45]:= composite[IMAGE[id[cart[V, V]]], JOIN] := JOIN
```