fixed point rules for JOIN

Johan G. F. Belinfante
2008 October 24

```
In[1]:= SetDirectory["l:"]; << goedel.08oct23a;<< tools.m

        :Package Title: goedel.08oct23a          2008 October 23 at 5:15 p.m.

        It is now:  2008 Oct 24 at 13:22

        Loading Simplification Rules

        TOOLS.M                                    Revised 2008 October 21

        weightlimit = 40
```

summary

If appending or prepending a particular list never makes a difference, then that list is empty. Variable-free statements to this effect are derived in this notebook.

image[inverse[JOIN], set[0]]

Lemma. If the join of two lists is empty, then both lists are empty.

```
In[2]:= SubstTest[implies, equal[u, v], equal[domain[u], domain[v]],
          {u → 0, v :> APPLY[JOIN, PAIR[list[x], list[y]]]}] // Reverse

Out[2]= or[and[equal[0, list[x]], equal[0, list[y]]],
          not[equal[0, APPLY[JOIN, PAIR[list[x], list[y]]]]]] == True

In[3]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. Combining the lemma with its converse yields a logical equivalence that can be made into a rewrite rule.

```
In[4]:= equiv[equal[0, APPLY[JOIN, PAIR[list[x], list[y]]]],
         and[equal[0, list[x]], equal[0, list[y]]]]

Out[4]= True

In[5]:= equal[0, APPLY[JOIN, PAIR[list[x_], list[y_]]]] :=
         and[equal[0, list[x]], equal[0, list[y]]]
```

Corollary. A reformulation that makes it easier to apply **reify**.

```
In[6]:= image[V, APPLY[JOIN, PAIR[list[x], list[y]]]] // Normality

Out[6]= image[V, APPLY[JOIN, PAIR[list[x], list[y]]]] ==
         union[image[V, list[x]], image[V, list[y]]]

In[7]:= image[V, APPLY[JOIN, PAIR[list[x_], list[y_]]]] :=
         union[image[V, list[x]], image[V, list[y]]]
```

Lemma. A rewrite rule needed to simplify the **reify** result about to be derived.

```
In[8]:= (ub[composite[inverse[funpart[t]], complement[E]], V] // Normality) /. t → JOIN

Out[8]= ub[composite[inverse[JOIN], complement[E]], V] == image[inverse[JOIN], set[0]]

In[9]:= % /. Equal→ SetDelayed
```

Lemma. Eliminating both variables by two applications of **reify**. This result is simplified further in the theorem that follows.

```
In[10]:= Map[composite[id[LISTS], inverse[reify[y, complement[#]]], id[LISTS]] &,
          Map[domain[composite[#, id[LISTS]]] &,
           SubstTest[reify, x, image[V, APPLY[t, PAIR[list[x], list[y]]]], t → JOIN]]]

Out[10]= composite[id[LISTS], image[inverse[JOIN], set[0]], id[LISTS]] == cart[set[0], set[0]]

In[11]:= % /. Equal→ SetDelayed
```

Theorem. If the join of two lists is empty, then both lists are empty.

```
In[12]:= IminComp[JOIN, id[cartsq[LISTS]], set[0]]

Out[12]= image[inverse[JOIN], set[0]] == cart[set[0], set[0]]

In[13]:= image[inverse[JOIN], set[0]] := cart[set[0], set[0]]
```

---

a fixed point theorem

Lemma. A temporary rewrite rule.

```
In[14]:= SubstTest[member, list[x], image[intersection[u, v], set[w]],
           {u → JOIN, v → FIRST, w → pair[list[x], list[y]]}] // Reverse

Out[14]= equal[APPLY[JOIN, PAIR[list[x], list[y]]], list[x]] ==
          member[pair[pair[list[x], list[y]], list[x]], JOIN]

In[15]:= equal[APPLY[JOIN, PAIR[list[x_], list[y_]]], list[x_]] :=
          member[pair[pair[list[x], list[y]], list[x]], JOIN]
```

Theorem.

```
In[16]:= SubstTest[implies, equal[u, v], equal[domain[u], domain[v]],
           {u -> APPLY[JOIN, PAIR[list[x], list[y]]], v → list[x]}] // Reverse

Out[16]= or[equal[0, list[y]], not[member[pair[pair[list[x], list[y]], list[x]], JOIN]]] == True

In[17]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Observation used to motivate the **reify** result derived below. (Here **JOIN** been replaced by **z** to prevent application of the preceding theorem.)

```
In[18]:= class[t, or[equal[0, list[y]], not[member[pair[pair[list[x], list[y]], list[x]], z]]]]

Out[18]= union[complement[image[V,
            intersection[image[image[inverse[z], set[list[x]]], set[list[x]]], set[list[y]]]]],
          complement[image[V, list[y]]]]
```

Lemma. Simplification rule.

```
In[19]:= Map[range, SubstTest[fix, composite[x, JOIN, id[y]], y → cart[V, LISTS]]]

Out[19]= intersection[LISTS, range[fix[composite[x, JOIN]]]] == range[fix[composite[x, JOIN]]]

In[20]:= intersection[LISTS, range[fix[composite[x_, JOIN]]]] := range[fix[composite[x, JOIN]]]
```

Theorem. Elimination of both variables using **reify**.

```
In[21]:= Map[assert[empty[reify[y, #]]] &,
          Map[composite[Id, complement[#]] &, SubstTest[reify, x,
            union[complement[image[V, intersection[image[image[inverse[t], set[list[x]]],
                 set[list[x]]], set[list[y]]]]], complement[image[V, list[y]]]], t → JOIN]]]

Out[21]= subclass[range[fix[composite[inverse[FIRST], JOIN]]], set[0]] == True

In[22]:= % /. Equal → SetDelayed
```

Lemma. Reverse includion.

```
In[23]:= member[0, range[fix[composite[inverse[FIRST], JOIN]]]] // AssertTest

Out[23]= member[0, range[fix[composite[inverse[FIRST], JOIN]]]] == True

In[24]:= % /. Equal → SetDelayed
```

The above two inclusions can be combined into a temporary equation:

```
In[25]:= equal[range[fix[composite[inverse[FIRST], JOIN]]], set[0]]

Out[25]= True

In[26]:= range[fix[composite[inverse[FIRST], JOIN]]] := set[0]
```

Theorem. A fixed point rule for **JOIN**.

```
In[27]:= (implies[equal[range[t], set[0]],
            equal[composite[Id, t], cart[image[inverse[t], set[0]], set[0]]]] // AssertTest) /.
          t -> fix[composite[inverse[FIRST], JOIN]]
```

```
Out[27]= equal[cart[LISTS, set[0]], fix[composite[inverse[FIRST], JOIN]]] == True
```

```
In[28]:= fix[composite[inverse[FIRST], JOIN]] := cart[LISTS, set[0]]
```

---

a similar result

A temporary rewrite rule.

```
In[29]:= SubstTest[member, list[y], image[intersection[u, v], set[w]],
           {u → JOIN, v → SECOND, w → pair[list[x], list[y]]}] // Reverse
```

```
Out[29]= equal[APPLY[JOIN, PAIR[list[x], list[y]]], list[y]] ==
          member[pair[pair[list[x], list[y]], list[y]], JOIN]
```

```
In[30]:= equal[APPLY[JOIN, PAIR[list[x_], list[y_]]], list[y_]] :=
          member[pair[pair[list[x], list[y]], list[y]], JOIN]
```

Lemma.

```
In[31]:= SubstTest[implies, equal[u, v], equal[domain[u], domain[v]],
           {u -> APPLY[JOIN, PAIR[list[x], list[y]]], v → list[y]}] // Reverse
```

```
Out[31]= or[equal[0, list[x]], not[member[pair[pair[list[x], list[y]], list[y]], JOIN]]] == True
```

```
In[32]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. A simplification rule.

```
In[33]:= Map[domain, SubstTest[fix, composite[x, JOIN, id[y]], y → cart[LISTS, V]]]
```

```
Out[33]= intersection[LISTS, domain[fix[composite[x, JOIN]]]] == domain[fix[composite[x, JOIN]]]
```

```
In[34]:= intersection[LISTS, domain[fix[composite[x_, JOIN]]]] :=
          domain[fix[composite[x, JOIN]]]
```

Theorem. Removal of two variables using **reify** twice.

```
In[35]:= Map[assert[empty[composite[Id, complement[reify[y, #]]]]] &,
          Map[domain, SubstTest[reify, x,
            union[complement[image[V, intersection[image[image[inverse[t], set[list[y]]],
                 set[list[x]]], set[list[y]]]]], complement[image[V, list[x]]]], t → JOIN]]]
```

```
Out[35]= subclass[domain[fix[composite[inverse[SECOND], JOIN]]], set[0]] == True
```

```
In[36]:= % /. Equal → SetDelayed
```

Lemma. The reverse inclusion.

```
In[37]:= member[0, domain[fix[composite[inverse[SECOND], JOIN]]]] // AssertTest

Out[37]= member[0, domain[fix[composite[inverse[SECOND], JOIN]]]] == True

In[38]:= % /. Equal→ SetDelayed
```

Theorem. Temporary equation obtained by combining the above two inclusions.

```
In[39]:= equal[domain[fix[composite[inverse[SECOND], JOIN]]], set[0]]

Out[39]= True

In[40]:= domain[fix[composite[inverse[SECOND], JOIN]]] := set[0]
```

Corollary. A better rewrite rule.

```
In[41]:= (implies[equal[domain[t], set[0]],
            equal[composite[Id, t], cart[set[0], image[t, set[0]]]]] // AssertTest) /.
          t -> fix[composite[inverse[SECOND], JOIN]]

Out[41]= equal[cart[set[0], LISTS], fix[composite[inverse[SECOND], JOIN]]] == True

In[42]:= fix[composite[inverse[SECOND], JOIN]] := cart[set[0], LISTS]
```