APPLY rules for MONOIDS

Johan G. F. Belinfante
2009 January 1

In[1]:=
```
SetDirectory["l:"]; << goedel.09jan01a;<< tools.m
```

```
:Package Title: goedel.09jan01a        2009 January 1 at 5:00 p.m.

It is now:  2009 Jan 1 at 17:6

Loading Simplification Rules

TOOLS.M                                          Revised 2008 December 26

weightlimit = 40
```

summary

To facilitate the kind of reasoning about monoids that one finds in the literature, it is convenient to restate various properties of monoids using elements and **APPLY**. If **x** is a monoid operation, then **range[x]** is the underlying set on which this operation acts. If **u** and **v** are elements of **range[x]**, then **APPLY[x, PAIR[u, v]]** is their **product**, usually denoted in the literature by **u · v**. In this notebook, the usual formulation of the definition (axioms) for a monoid are derived.

monoids as mappings

Theorem.

In[2]:=
```
Map[not, SubstTest[and, implies[p1, p2],
   implies[p2, p3], implies[and[p1, p3], p4], not[implies[p1, p4]],
   {p1 → member[x, MONOIDS], p2 → equal[range[x], fix[domain[x]]], p3 → equal[
      map[cartsq[range[x]], range[x]], map[cartsq[fix[domain[x]]], fix[domain[x]]]],
    p4 -> member[x, map[cart[range[x], range[x]], range[x]]]}]] // Reverse
```

Out[2]=
```
or[member[x, map[cart[range[x], range[x]], range[x]]], not[member[x, MONOIDS]]] == True
```

In[3]:=
```
or[member[x_, map[cart[range[x_], range[x_]], range[x_]]],
  not[member[x_, MONOIDS]]] := True
```

Corollary.

In[4]:=
```
Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
   {p1 → member[x, MONOIDS], p2 -> member[x, map[cartsq[range[x]], range[x]]],
    p3 -> equal[cart[range[x], range[x]], domain[x]]}]] // Reverse
```

Out[4]=
```
or[equal[cart[range[x], range[x]], domain[x]], not[member[x, MONOIDS]]] == True
```

```
In[5]:= or[equal[cart[range[x_], range[x_]], domain[x_]], not[member[x_, MONOIDS]]] := True
```

closure

Lemma.

```
In[6]:= SubstTest[implies, and[member[w, map[t, z]], member[s, t]],
          member[APPLY[w, s], z], {s → PAIR[u, v], t → cart[x, y]}] // Reverse

Out[6]= or[member[APPLY[w, PAIR[u, v]], z], not[member[u, x]],
          not[member[v, y]], not[member[w, map[cart[x, y], z]]]] == True

In[7]:= or[member[APPLY[w_, PAIR[u_, v_]], z_], not[member[u_, x_]],
          not[member[v_, y_]], not[member[w_, map[cart[x_, y_], z_]]]] := True
```

Theorem. (Closure property.) For any monoid **x**, one has: **u ∈ range[x] & v ∈ range[x] ⟹ u · v ∈ range[x]**.

```
In[8]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3], not[implies[p1, p3]],
           {p1 -> and[member[x, MONOIDS], member[u, range[x]], member[v, range[x]]],
            p2 -> member[x, map[cart[range[x], range[x]], range[x]]],
            p3 -> member[APPLY[x, PAIR[u, v]], range[x]]}]] // Reverse

Out[8]= or[member[APPLY[x, PAIR[u, v]], range[x]], not[member[u, range[x]]],
          not[member[v, range[x]]], not[member[x, MONOIDS]]] == True

In[9]:= or[member[APPLY[x_, PAIR[u_, v_]], range[x_]], not[member[u_, range[x_]]],
          not[member[v_, range[x_]]], not[member[x_, MONOIDS]]] := True
```

associativity

Theorem. Associative law: **(u · v) · w = u · (v · w)** Note that this rewrite rule does not need explicit hypotheses that the three variables **u**, **v** and **w** are members of **range[x]** because if this is not the case the equation reduces to **V = V**.

```
In[10]:= Map[not, SubstTest[and, implies[p1, p2],
            implies[p2, p3], not[implies[p1, p3]], {p1 → member[x, MONOIDS],
             p2 → member[x, SEMIGPS], p3 -> equal[APPLY[x, PAIR[u, APPLY[x, PAIR[v, w]]]],
               APPLY[x, PAIR[APPLY[x, PAIR[u, v]], w]]]}]] // Reverse

Out[10]= or[equal[APPLY[x, PAIR[u, APPLY[x, PAIR[v, w]]]],
            APPLY[x, PAIR[APPLY[x, PAIR[u, v]], w]]], not[member[x, MONOIDS]]] == True

In[11]:= or[equal[APPLY[x_, PAIR[APPLY[x_, PAIR[u_, v_]], w_]],
            APPLY[x_, PAIR[u_, APPLY[x_, PAIR[v_, w_]]]]], not[member[x_, MONOIDS]]] := True
```

left unital law

Lemma.

```
In[15]:= Map[or[#, equal[z, APPLY[x, PAIR[y, z]]], not[member[z, range[x]]]] &,
           SubstTest[implies, equal[u, v], equal[APPLY[u, z], APPLY[v, z]],
            {u → composite[x, LEFT[y]], v → id[range[x]]}]] // Reverse

Out[15]= or[equal[z, APPLY[x, PAIR[y, z]]],
           not[equal[composite[x, LEFT[y]], id[range[x]]]], not[member[z, range[x]]]] == True

In[16]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Theorem.

```
In[20]:= Map[not, SubstTest[and, implies[p1, p3], implies[and[p2, p3], p4],
            not[implies[and[p1, p2], p4]], {p1 → member[x, MONOIDS],
             p2 → member[w, range[x]], p3 → equal[composite[x, LEFT[e[x]]], id[range[x]]],
             p4 → equal[w, APPLY[x, PAIR[e[x], w]]]}]] // Reverse

Out[20]= or[equal[w, APPLY[x, PAIR[e[x], w]]],
           not[member[w, range[x]]], not[member[x, MONOIDS]]] == True

In[21]:= or[equal[w_, APPLY[x_, PAIR[e[x_], w_]]],
           not[member[w_, range[x_]]], not[member[x_, MONOIDS]]] := True
```

right unital law

The right unital law will be derived using **flip**. This approach requires two lemmas.

Lemma.

```
In[23]:= SubstTest[implies, and[member[w, range[t]], member[t, MONOIDS]],
           equal[w, APPLY[t, PAIR[e[t], w]]], t → flip[x]] // Reverse

Out[23]= or[equal[w, APPLY[x, PAIR[w, e[x]]]], not[member[w, image[x, cart[V, V]]]],
           not[member[composite[x, SWAP], MONOIDS]]] == True

In[24]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

Lemma.

```
In[26]:= SubstTest[implies, equal[x, binop[t]],
           equal[image[x, cart[V, V]], range[x]], t → x] // Reverse

Out[26]= or[equal[image[x, cart[V, V]], range[x]], not[member[x, BINOPS]]] == True

In[27]:= or[equal[image[x_, cart[V, V]], range[x_]], not[member[x_, BINOPS]]] := True
```

Theorem.

```
In[29]:= Map[not, SubstTest[and, implies[p1, p3], implies[p1, p5],
            implies[p3, p4], implies[and[p2, p4, p5], p6], not[implies[and[p1, p2], p6]],
            {p1 → member[x, MONOIDS], p2 → member[w, range[x]], p3 → member[x, BINOPS], p4 ->
              equal[image[x, cart[V, V]], range[x]], p5 -> member[composite[x, SWAP], MONOIDS],
             p6 -> equal[w, APPLY[x, PAIR[w, e[x]]]]}]] // Reverse

Out[29]= or[equal[w, APPLY[x, PAIR[w, e[x]]]],
           not[member[w, range[x]]], not[member[x, MONOIDS]]] == True

In[31]:= or[equal[APPLY[x_, PAIR[w_, e[x_]]], w_],
           not[member[w_, range[x_]]], not[member[x_, MONOIDS]]] := True
```

a corollary

Theorem.

```
In[36]:= Map[not, SubstTest[and, implies[p1, p2],
            implies[and[p1, p2], p3], not[implies[p1, p3]], {p1 → member[x, MONOIDS],
             p2 → member[e[x], ids[x]], p3 → member[e[x], range[x]]}]] // Reverse

Out[36]= or[member[e[x], range[x]], not[member[x, MONOIDS]]] == True

In[37]:= or[member[e[x_], range[x_]], not[member[x_, MONOIDS]]] := True
```

Corollary. If **x** is a monoid, then **e[x] · e[x] = e[x]**.

```
In[39]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3],
            not[implies[p1, p3]], {p1 → member[x, MONOIDS], p2 → member[e[x], range[x]],
             p3 → equal[APPLY[x, PAIR[e[x], e[x]]], e[x]]}]] // Reverse

Out[39]= or[equal[APPLY[x, PAIR[e[x], e[x]]], e[x]], not[member[x, MONOIDS]]] == True

In[41]:= or[equal[APPLY[x_, PAIR[e[x_], e[x_]]], e[x_]], not[member[x_, MONOIDS]]] := True
```