MONOIDS

Johan G. F. Belinfante
2008 December 29

```
In[1]:= SetDirectory["l:"]; << goedel.08dec28a;<< tools.m

        :Package Title: goedel.08dec28a         2008 December 28 at 10:10 a.m.

        It is now:  2008 Dec 29 at 5:52

        Loading Simplification Rules

        TOOLS.M                                      Revised 2008 December 26

        weightlimit = 40
```

summary

The class **MONOIDS** of monoids is introduced, and some examples of monoids are given.

definition

The definition of the class **MONOIDS** is:

```
In[2]:= image[V, intersection[MONOIDS, set[x_]]] := intersection[image[V, ids[x]],
          image[V, intersection[ASSOCIATIVE, set[x]]], image[V, intersection[BINOPS, set[x]]]]
```

immediate properties of monoids

Theorem.

```
In[3]:= Map[empty, dif[MONOIDS, SEMIGPS] // Renormality]

Out[3]= subclass[MONOIDS, SEMIGPS] == True

In[4]:= subclass[MONOIDS, SEMIGPS] := True
```

Corollary.

```
In[5]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]],
          subclass[u, w], {u → MONOIDS, v → SEMIGPS, w → BINOPS}] // Reverse

Out[5]= subclass[MONOIDS, BINOPS] == True
```

```
In[6]:= subclass[MONOIDS, BINOPS] := True
```

Corollary.

```
In[7]:= implies[member[x, MONOIDS], FUNCTION[x]] // AssertTest

Out[7]= or[FUNCTION[x], not[member[x, MONOIDS]]] == True

In[8]:= or[FUNCTION[x_], not[member[x_, MONOIDS]]] := True
```

Corollary.

```
In[9]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 → member[x, MONOIDS], p2 → member[x, SEMIGPS], p3 → associative[x]}]] // Reverse

Out[9]= or[associative[x], not[member[x, MONOIDS]]] == True

In[10]:= or[associative[x_], not[member[x_, MONOIDS]]] := True
```

Theorem.

```
In[11]:= or[not[equal[0, ids[x]]], not[member[x, MONOIDS]]] // AssertTest

Out[11]= or[not[equal[0, ids[x]]], not[member[x, MONOIDS]]] == True

In[12]:= or[not[equal[0, ids[x_]]], not[member[x_, MONOIDS]]] := True
```

Theorem.

```
In[13]:= Map[not, SubstTest[and, implies[p1, p2],
            implies[p2, p3], not[implies[p1, p3]], {p1 → member[x, MONOIDS],
             p2 -> member[x, BINOPS], p3 → equal[ids[x], set[e[x]]]}]] // Reverse

Out[13]= or[equal[ids[x], set[e[x]]], not[member[x, MONOIDS]]] == True

In[14]:= or[equal[ids[x_], set[e[x_]]], not[member[x_, MONOIDS]]] := True
```

Theorem.

```
In[15]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p2, p3], p4],
            not[implies[p1, p4]], {p1 → member[x, MONOIDS], p2 -> not[equal[0, ids[x]]],
             p3 → member[x, BINOPS], p4 → member[e[x], ids[x]]}]] // Reverse

Out[15]= or[member[e[x], ids[x]], not[member[x, MONOIDS]]] == True

In[16]:= or[member[e[x_], ids[x_]], not[member[x_, MONOIDS]]] := True
```

Theorem.

```
In[17]:= or[equal[0, ids[x]], member[x, MONOIDS], not[member[x, SEMIGPS]]] // AssertTest

Out[17]= or[equal[0, ids[x]], member[x, MONOIDS], not[member[x, SEMIGPS]]] == True

In[18]:= or[equal[0, ids[x_]], member[x_, MONOIDS], not[member[x_, SEMIGPS]]] := True
```

examples

Theorem. The natural numbers are a monoid under addition.

```
In[19]:= member[NATADD, MONOIDS] // AssertTest
Out[19]= member[NATADD, MONOIDS] == True
```

```
In[20]:= member[NATADD, MONOIDS] := True
```

Theorem. The natural numbers are a monoid under multiplication.

```
In[21]:= member[NATMUL, MONOIDS] // AssertTest
Out[21]= member[NATMUL, MONOIDS] == True
```

```
In[22]:= member[NATMUL, MONOIDS] := True
```

Theorem. The integers form a monoid under addition.

```
In[23]:= member[INTADD, MONOIDS] // AssertTest
Out[23]= member[INTADD, MONOIDS] == True
```

```
In[24]:= member[INTADD, MONOIDS] := True
```

Theorem. The integers form a monoid under multiplication.

```
In[25]:= member[INTMUL, MONOIDS] // AssertTest
Out[25]= member[INTMUL, MONOIDS] == True
```

```
In[26]:= member[INTMUL, MONOIDS] := True
```

Theorem. Monoids are never empty.

```
In[27]:= member[0, MONOIDS] // AssertTest
Out[27]= member[0, MONOIDS] == False
```

```
In[28]:= member[0, MONOIDS] := False
```

one-element monoids

Lemma.

```
In[29]:= ids[cart[cart[set[x], set[x]], set[x]]] // Normality
Out[29]= ids[cart[cart[set[x], set[x]], set[x]]] == set[x]
```

```
In[30]:= ids[cart[cart[set[x_], set[x_]], set[x_]]] := set[x]
```

Theorem. Any singleton can be made into a monoid.

```
In[31]:= member[cart[cart[set[x], set[x]], set[x]], MONOIDS] // AssertTest

Out[31]= member[cart[cart[set[x], set[x]], set[x]], MONOIDS] == member[x, V]

In[32]:= member[cart[cart[set[x_], set[x_]], set[x_]], MONOIDS] := member[x, V]
```

direct products of binary operations

Theorem. The neutral element (if any) of the direct product of binary operations is the ordered pair of their neutral elements.

```
In[33]:= Map[equal[#, e[composite[cross[binop[x], binop[y]], TWIST]]] &,
           Map[A, SubstTest[ids, binop[t], t -> direct[binop[x], binop[y]]]]] // Reverse

Out[33]= equal[e[composite[cross[binop[x], binop[y]], TWIST]],
           PAIR[e[binop[x]], e[binop[y]]]] == True

In[34]:= e[composite[cross[binop[x_], binop[y_]], TWIST]] := PAIR[e[binop[x]], e[binop[y]]]
```

Comment. This equation is valid whether or not the binary operations have neutral elements. If a binary operation **x** fails to have a neutral element, then **e[x]=V**, and then **PAIR[e[x], e[y]] = V**, which means that in that case the direct product of **x** with any binary operation **y** also fails to have a neutral element.

Corollary.

```
In[35]:= SubstTest[implies, and[equal[x, binop[u]], equal[y, binop[v]]],
           equal[e[direct[x, y]], PAIR[e[x], e[y]]], {u → x, v → y}] // Reverse

Out[35]= or[equal[e[composite[cross[x, y], TWIST]], PAIR[e[x], e[y]]],
           not[member[x, BINOPS]], not[member[y, BINOPS]]] == True

In[36]:= or[equal[e[composite[cross[x_, y_], TWIST]], PAIR[e[x_], e[y_]]],
           not[member[x_, BINOPS]], not[member[y_, BINOPS]]] := True
```

direct products of monoids

Lemma.

```
In[37]:= (Map[not, SubstTest[and, implies[p0, p1],
              implies[p0, p2], implies[and[p0, p1, p2], p3], not[implies[p0, p3]],
              {p0 → and[member[x, MONOIDS], member[y, MONOIDS], equal[z, direct[x, y]]],
               p1 → member[x, SEMIGPS], p2 → member[y, SEMIGPS], p3 → member[z, SEMIGPS],
               p4 → not[empty[ids[x]]], p5 → not[empty[ids[y]]],
               p6 → equal[ids[z], cart[ids[x], ids[y]]], p7 → not[empty[ids[z]]],
               p8 → member[z, MONOIDS]}]] /. z → direct[x, y]) // Reverse

Out[37]= or[member[composite[cross[x, y], TWIST], SEMIGPS],
           not[member[x, MONOIDS]], not[member[y, MONOIDS]]] == True

In[38]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. The direct product of monoids is a monoid. (Comment. The following step of the proof has been deliberately omitted to speed up this derivation: **implies[and[p2, p3, p4], p5]**. The rewrite rules of the **GOEDEL** program suffice to supply this step automatically.)

```
In[39]:= (Map[not, SubstTest[and, implies[p0, p1], implies[p0, p2], implies[p0, p3],
              implies[p0, p4], implies[and[p1, p5], p6], not[implies[p0, p6]],
              {p0 → and[member[x, MONOIDS], member[y, MONOIDS], equal[z, direct[x, y]]],
               p1 → member[z, SEMIGPS], p2 → not[empty[ids[x]]], p3 → not[empty[ids[y]]],
               p4 → equal[ids[z], cart[ids[x], ids[y]]], p5 → not[empty[ids[z]]],
               p6 → member[z, MONOIDS]}]] /. z → direct[x, y]) // Reverse

Out[39]= or[member[composite[cross[x, y], TWIST], MONOIDS],
           not[member[x, MONOIDS]], not[member[y, MONOIDS]]] == True

In[40]:= or[member[composite[cross[x_, y_], TWIST], MONOIDS],
           not[member[x_, MONOIDS]], not[member[y_, MONOIDS]]] := True
```

Corollary. Variable-free restatement of the theorem.

```
In[41]:= Map[empty[composite[Id, complement[#]]] &,
          dif[cartsq[MONOIDS], image[inverse[composite[IMAGE[cross[TWIST, Id]], CROSS]],
              MONOIDS]] // complement // RelnNormality]

Out[41]= subclass[
           image[IMAGE[cross[TWIST, Id]], image[CROSS, cart[MONOIDS, MONOIDS]]], MONOIDS] == True

In[42]:= subclass[
           image[IMAGE[cross[TWIST, Id]], image[CROSS, cart[MONOIDS, MONOIDS]]], MONOIDS] := True
```