SEMIGPS

Johan G. F. Belinfante
2006 July 8

```
In[1]:= SetDirectory["l:"]; << goedel83.06a; << tools.m

        :Package Title: goedel83.06a         2006 July 6 at 5:00 p.m.

        It is now:  2006 Jul 8 at 17:23

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 June 27

        weightlimit = 40
```

summary

The class **SEMIGPS** of all associative binary operations is introduced in this notebook using an **image[V, -]** rewrite rule, and some basic properties of this class are deduced.

ersatz membership rule

The definition of **SEMIGPS** is:

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

The simplest example is the empty set. This case serves to verify that the definition can indeed act as an ersatz membership rule.

```
In[3]:= member[0, SEMIGPS] // AssertTest

Out[3]= member[0, SEMIGPS] == True

In[4]:= member[0, SEMIGPS] := True
```

To prevent expanding the membership rule, it is turned around:

```
In[5]:= SubstTest[equal, V, image[V, intersection[w, set[x]]], w → SEMIGPS]

Out[5]= and[associative[x], member[x, BINOPS]] == member[x, SEMIGPS]

In[6]:= and[associative[x_], member[x_, BINOPS]] := member[x, SEMIGPS]
```

an equation for SEMIGPS

An equation for **SEMIGPS** is readily derived:

```
In[7]:= SubstTest[class, x, member[x, intersection[y, z]],
          {y → ASSOCIATIVE, z → BINOPS}] // Reverse
Out[7]= intersection[ASSOCIATIVE, BINOPS] == SEMIGPS

In[8]:= intersection[ASSOCIATIVE, BINOPS] := SEMIGPS
```

Corollaries:

```
In[9]:= SubstTest[subclass, intersection[u, v], u, {u → ASSOCIATIVE, v → BINOPS}]
Out[9]= subclass[SEMIGPS, ASSOCIATIVE] == True

In[10]:= subclass[SEMIGPS, ASSOCIATIVE] := True

In[11]:= SubstTest[subclass, intersection[u, v], v, {u → ASSOCIATIVE, v → BINOPS}]
Out[11]= subclass[SEMIGPS, BINOPS] == True

In[12]:= subclass[SEMIGPS, BINOPS] := True
```

examples in arithmetic

Arithmetic of natural numbers:

```
In[13]:= member[NATADD, SEMIGPS] // AssertTest
Out[13]= member[NATADD, SEMIGPS] == True

In[14]:= member[NATADD, SEMIGPS] := True

In[15]:= member[NATMUL, SEMIGPS] // AssertTest
Out[15]= member[NATMUL, SEMIGPS] == True

In[16]:= member[NATMUL, SEMIGPS] := True
```

Integer addition:

```
In[17]:= member[INTADD, SEMIGPS] // AssertTest
Out[17]= member[INTADD, SEMIGPS] == True

In[18]:= member[INTADD, SEMIGPS] := True
```

examples in modular arithmetic

Modular addition:

```
In[19]:= member[composite[modulo[x], NATADD], BINOPS] // AssertTest
Out[19]= member[composite[modulo[x], NATADD], BINOPS] == True

In[20]:= member[composite[modulo[x_], NATADD], BINOPS] := True

In[21]:= member[composite[modulo[x], NATADD], V] // AssertTest
Out[21]= member[composite[modulo[x], NATADD], V] == True

In[22]:= member[composite[modulo[x_], NATADD], V] := True

In[23]:= member[composite[modulo[x], NATADD], SEMIGPS] // AssertTest
Out[23]= member[composite[modulo[x], NATADD], SEMIGPS] == True

In[24]:= member[composite[modulo[x_], NATADD], SEMIGPS] := True
```

Modular multiplication:

```
In[25]:= member[composite[modulo[x], NATMUL], V] // AssertTest
Out[25]= member[composite[modulo[x], NATMUL], V] == True

In[26]:= member[composite[modulo[x_], NATMUL], V] := True

In[27]:= member[composite[modulo[x], NATMUL], BINOPS] // AssertTest
Out[27]= member[composite[modulo[x], NATMUL], BINOPS] == True

In[28]:= member[composite[modulo[x_], NATMUL], BINOPS] := True

In[29]:= member[composite[modulo[x], NATMUL], SEMIGPS] // AssertTest
Out[29]= member[composite[modulo[x], NATMUL], SEMIGPS] == True

In[30]:= member[composite[modulo[x_], NATMUL], SEMIGPS] := True
```

examples in Boolean algebra

```
In[31]:= SubstTest[implies, and[associative[w], subclass[image[w, cart[y, y]], y]],
          associative[composite[w, id[cart[y, y]]]], {w → CAP, y → P[x]}]
Out[31]= associative[composite[CAP, id[cart[P[x], P[x]]]]] == True

In[32]:= associative[composite[CAP, id[cart[P[x_], P[x_]]]]] := True
```

```
In[33]:= member[composite[CAP, id[cart[P[x], P[x]]]], SEMIGPS] // AssertTest
Out[33]= member[composite[CAP, id[cart[P[x], P[x]]]], SEMIGPS] == member[x, V]

In[34]:= member[composite[CAP, id[cart[P[x_], P[x_]]]], SEMIGPS] := member[x, V]
```

Lemma.

```
In[35]:= SubstTest[implies, and[associative[w], subclass[image[w, cart[y, y]], y]],
          associative[composite[w, id[cart[y, y]]]], {w → CUP, y → P[x]}]
Out[35]= associative[composite[id[P[x]], CUP]] == True

In[36]:= associative[composite[id[P[x_]], CUP]] := True
```

Theorem.

```
In[37]:= member[composite[id[P[x]], CUP], SEMIGPS] // AssertTest
Out[37]= member[composite[id[P[x]], CUP], SEMIGPS] == member[x, V]

In[38]:= member[composite[id[P[x_]], CUP], SEMIGPS] := member[x, V]
```

---

constant semigroup operations

Any constant binary operation is associative.

```
In[39]:= Map[equal[0, #] &, dif[composite[inverse[E], IMAGE[SINGLETON]],
            composite[image[inverse[CART], ASSOCIATIVE], CART, DUP]] // RelnNormality]
Out[39]= subclass[intersection[BINOPS, CONST], ASSOCIATIVE] == True

In[40]:= subclass[intersection[BINOPS, CONST], ASSOCIATIVE] := True
```

Corollary:

```
In[41]:= SubstTest[subclass, w, intersection[x, y],
          {w -> intersection[BINOPS, CONST], x -> ASSOCIATIVE, y → BINOPS}]
Out[41]= subclass[intersection[BINOPS, CONST], SEMIGPS] == True

In[42]:= subclass[intersection[BINOPS, CONST], SEMIGPS] := True

In[43]:= SubstTest[implies, subclass[u, v], subclass[U[u], U[v]],
          {u -> intersection[BINOPS, CONST], v -> SEMIGPS}]
Out[43]= subclass[cart[cart[V, V], V], U[SEMIGPS]] == True

In[44]:= % /. Equal → SetDelayed
```

The reverse inclusion also holds:

```
In[45]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]],
           subclass[u, w], {u → SEMIGPS, v → BINOPS, w → P[cart[cart[V, V], V]]}]

Out[45]= subclass[U[SEMIGPS], cart[cart[V, V], V]] == True

In[46]:= % /. Equal → SetDelayed
```

Combining these, one finds:

```
In[47]:= SubstTest[and, subclass[u, v], subclass[v, u],
           {u -> U[SEMIGPS], v -> cart[cart[V, V], V]}]

Out[47]= True == equal[cart[cart[V, V], V], U[SEMIGPS]]

In[48]:= U[SEMIGPS] := cart[cart[V, V], V]
```

SEMIGPS is a proper class

```
In[50]:= SubstTest[member, U[x], V, x → SEMIGPS] // Reverse

Out[50]= member[SEMIGPS, V] == False

In[51]:= % /. Equal → SetDelayed

In[52]:= member[SEMIGPS, x] // AssertTest

Out[52]= member[SEMIGPS, x] == False

In[53]:= member[SEMIGPS, x_] := False
```