regular representations of monoids

Johan G. F. Belinfante
2013 June 7

```
In[1]:= SetDirectory["l:"]; << goedel.13jun06a

        :Package Title: goedel.13jun06a                 2013 June 6 at 11:40 p.m.

        Loading takes about sixteen minutes, half that time due to builtin pauses.

        It is now:  2013 Jun 7 at 11:45

        Loading Simplification Rules

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

        weightlimit = 40

        Loading completed.

        It is now:  2013 Jun 7 at 12:1
```

summary

The regular representation of a monoid is obtained by Currying. This binary homomorphism takes each element of the range of a monoid to left multiplication by that element. It is shown that the regular representation is faithful.

vertical section rules

Theorem. Vertical section rule for a Curried binary operation.

```
In[2]:= SubstTest[image, funpart[t], set[y], t → APPLY[CURRY, binop[x]]] // Reverse

Out[2]= image[APPLY[CURRY, binop[x]], set[y]] ==
         intersection[image[V, intersection[fix[domain[binop[x]]], set[y]]],
          set[composite[binop[x], LEFT[y]]]]

In[3]:= image[APPLY[CURRY, binop[x_]], set[y_]] :=
         intersection[image[V, intersection[fix[domain[binop[x]]], set[y]]],
          set[composite[binop[x], LEFT[y]]]]
```

Corollary. Vertical section rule for a Curried monoid.

```
In[4]:= SubstTest[image, funpart[t], set[y], t → APPLY[CURRY, monoid[x]]] // Reverse

Out[4]= image[APPLY[CURRY, monoid[x]], set[y]] == intersection[
          image[V, intersection[range[monoid[x]], set[y]]], set[composite[monoid[x], LEFT[y]]]]
```

```
In[5]:= image[APPLY[CURRY, monoid[x_]], set[y_]] := intersection[
          image[V, intersection[range[monoid[x]], set[y]]], set[composite[monoid[x], LEFT[y]]]]
```

the regular representation

It is shown in this section that **APPLY[CURRY, monoid[x]]** is a binary homomorphism from **monoid[x]** to **COMPOSE**. This is called the **regular representation** of a monoid.

Lemma.

```
In[6]:= dif[composite[COMPOSE, cross[APPLY[CURRY, monoid[x]], APPLY[CURRY, monoid[x]]]],
          composite[APPLY[CURRY, monoid[x]], monoid[x]]] // FastReifNormality

Out[6]= intersection[
          composite[COMPOSE, cross[APPLY[CURRY, monoid[x]], APPLY[CURRY, monoid[x]]]],
          composite[complement[APPLY[CURRY, monoid[x]]], monoid[x]]] == 0

In[7]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma. An inclusion.

```
In[8]:= SubstTest[empty, dif[u, v],
         {u -> composite[COMPOSE, cross[APPLY[CURRY, monoid[x]], APPLY[CURRY, monoid[x]]]],
          v -> composite[APPLY[CURRY, monoid[x]], monoid[x]]}]

Out[8]= subclass[composite[COMPOSE, cross[APPLY[CURRY, monoid[x]], APPLY[CURRY, monoid[x]]]],
          composite[APPLY[CURRY, monoid[x]], monoid[x]]] == True

In[9]:= (% /. x → x_) /. Equal → SetDelayed
```

Inclusions of functions can be replaced by equations.

Theorem.

```
In[10]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
           equal[u, composite[v, id[domain[u]]]],
           {u -> composite[COMPOSE, cross[APPLY[CURRY, monoid[x]], APPLY[CURRY, monoid[x]]]],
            v -> composite[APPLY[CURRY, monoid[x]], monoid[x]]}] // Reverse

Out[10]= equal[composite[COMPOSE, cross[APPLY[CURRY, monoid[x]], APPLY[CURRY, monoid[x]]]],
           composite[APPLY[CURRY, monoid[x]], monoid[x]]] == True

In[11]:= composite[APPLY[CURRY, monoid[x_]], monoid[x_]] :=
          composite[COMPOSE, cross[APPLY[CURRY, monoid[x]], APPLY[CURRY, monoid[x]]]]
```

Corollary. The regular representation is a binary homomorphism.

```
In[12]:= member[APPLY[CURRY, monoid[x]], binhom[monoid[x], COMPOSE]] // AssertTest

Out[12]= member[APPLY[CURRY, monoid[x]], binhom[monoid[x], COMPOSE]] == True
```

```
In[13]:= member[APPLY[CURRY, monoid[x_]], binhom[monoid[x_], COMPOSE]] := True
```

Lemma.

```
In[14]:= SubstTest[implies, member[t, binhom[u, v]], member[range[t], binclosed[v]],
           {t -> APPLY[CURRY, monoid[x]], u -> monoid[x], v -> COMPOSE}] // Reverse

Out[14]= subclass[image[COMPOSE,
            cart[range[APPLY[CURRY, monoid[x]]], range[APPLY[CURRY, monoid[x]]]]],
           range[APPLY[CURRY, monoid[x]]]] == True
```

```
In[15]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[16]:= Map[implies[not[equal[0, monoid[x]]], #] &,
          SubstTest[member, APPLY[funpart[t], u], range[funpart[t]],
            {t -> APPLY[CURRY, monoid[x]], u -> e[monoid[x]]}] // Reverse]

Out[16]= or[equal[0, monoid[x]],
           member[id[range[monoid[x]]], range[APPLY[CURRY, monoid[x]]]]] == True
```

```
In[17]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem.

```
In[18]:= equiv[member[id[range[monoid[x]]], range[APPLY[CURRY, monoid[x]]]],
          not[equal[0, monoid[x]]]]

Out[18]= True
```

```
In[19]:= member[id[range[monoid[x_]]], range[APPLY[CURRY, monoid[x_]]]] :=
          not[equal[0, monoid[x]]]
```

Lemma.

```
In[23]:= SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]],
           {t → COMPOSE, u → cart[set[id[range[monoid[x]]]], range[APPLY[CURRY, monoid[x]]]],
            v → cart[range[APPLY[CURRY, monoid[x]]], range[APPLY[CURRY, monoid[x]]]]}] // Reverse

Out[23]= subclass[range[APPLY[CURRY, monoid[x]]], image[COMPOSE,
            cart[range[APPLY[CURRY, monoid[x]]], range[APPLY[CURRY, monoid[x]]]]]] == True
```

```
In[24]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem.

```
In[25]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> range[APPLY[CURRY, monoid[x]]], v -> image[COMPOSE,
             cart[range[APPLY[CURRY, monoid[x]]], range[APPLY[CURRY, monoid[x]]]]]}]

Out[25]= equal[image[COMPOSE,
            cart[range[APPLY[CURRY, monoid[x]]], range[APPLY[CURRY, monoid[x]]]],
           range[APPLY[CURRY, monoid[x]]]] == True
```

```
In[26]:= image[COMPOSE, cart[range[APPLY[CURRY, monoid[x_]]],
            range[APPLY[CURRY, monoid[x_]]]]] := range[APPLY[CURRY, monoid[x]]]
```

special cases

Some special cases of interest are worth noting.

Theorem.

```
In[27]:= SubstTest[member, APPLY[CURRY, monoid[x]],
           binhom[monoid[x], COMPOSE], x → INTADD] // Reverse
Out[27]= member[INTPLUS, binhom[INTADD, COMPOSE]] == True

In[28]:= member[INTPLUS, binhom[INTADD, COMPOSE]] := True
```

Theorem.

```
In[29]:= SubstTest[member, APPLY[CURRY, monoid[x]],
           binhom[monoid[x], COMPOSE], x → RATADD] // Reverse
Out[29]= member[RATPLUS, binhom[RATADD, COMPOSE]] == True

In[30]:= member[RATPLUS, binhom[RATADD, COMPOSE]] := True
```

Theorem.

```
In[31]:= SubstTest[member, APPLY[CURRY, monoid[x]],
           binhom[monoid[x], COMPOSE], x → RATMUL] // Reverse
Out[31]= member[RATTIMES, binhom[RATMUL, COMPOSE]] == True

In[32]:= member[RATTIMES, binhom[RATMUL, COMPOSE]] := True
```

functor rules

The function **COMPOSE** is not a monoid; it is not a set and it has no neutral elements. Suitable restrictions of **COMPOSE** are monoids. It is shown in this section that for one such restriction, the regular representation preserves the unity element and is therefore a functor.

Theorem.

```
In[33]:= Map[equal[V, domain[#]] &, SubstTest[reify, y, case[
            implies[member[y, range[monoid[x]]], member[composite[monoid[x], LEFT[y]], t]]],
           t -> map[range[monoid[x]], range[monoid[x]]]]]
Out[33]= subclass[range[APPLY[CURRY, monoid[x]]],
           map[range[monoid[x]], range[monoid[x]]]] == True
```

```
In[34]:= subclass[range[APPLY[CURRY, monoid[x_]]],
           map[range[monoid[x_]], range[monoid[x_]]]] := True
```

Corollary.

```
In[35]:= subclass[APPLY[CURRY, monoid[x]],
           cart[V, map[range[monoid[x]], range[monoid[x]]]]] // AssertTest

Out[35]= subclass[APPLY[CURRY, monoid[x]],
           cart[V, map[range[monoid[x]], range[monoid[x]]]]] == True

In[36]:= subclass[APPLY[CURRY, monoid[x_]],
           cart[V, map[range[monoid[x_]], range[monoid[x_]]]]] := True
```

Corollary. A simplification rule.

```
In[37]:= equal[composite[id[map[range[monoid[x]], range[monoid[x]]]], APPLY[CURRY, monoid[x]]],
           APPLY[CURRY, monoid[x]]]

Out[37]= True

In[38]:= composite[id[map[range[monoid[x_]], range[monoid[x_]]]], APPLY[CURRY, monoid[x_]]] :=
           APPLY[CURRY, monoid[x]]
```

Theorem. A functor rule.

```
In[39]:= (functor[APPLY[CURRY, monoid[x]], monoid[x], composite[COMPOSE, id[cart[t, t]]]] //
            AssertTest) /. t -> map[range[monoid[x]], range[monoid[x]]]

Out[39]= functor[APPLY[CURRY, monoid[x]], monoid[x],
           composite[COMPOSE, id[cart[map[range[monoid[x]], range[monoid[x]]],
              map[range[monoid[x]], range[monoid[x]]]]]] == True

In[40]:= functor[APPLY[CURRY, monoid[x_]], monoid[x_],
           composite[COMPOSE, id[cart[map[range[monoid[x_]], range[monoid[x_]]],
              map[range[monoid[x_]], range[monoid[x_]]]]]] := True
```

faithfulness

Theorem. A simplification rule.

```
In[41]:= Assoc[id[P[cart[V, V]]], id[FUNS], APPLY[CURRY, monoid[x]]]

Out[41]= composite[id[P[cart[V, V]]], APPLY[CURRY, monoid[x]]] == APPLY[CURRY, monoid[x]]

In[42]:= composite[id[P[cart[V, V]]], APPLY[CURRY, monoid[x_]]] := APPLY[CURRY, monoid[x]]
```

Theorem.

```
In[43]:= Map[inverse,
          Assoc[eval[e[monoid[x]]], APPLY[CURRY, monoid[x]], inverse[APPLY[CURRY, monoid[x]]]]]

Out[43]= composite[id[range[APPLY[CURRY, monoid[x]]]], inverse[eval[e[monoid[x]]]] ==
          APPLY[CURRY, monoid[x]]

In[44]:= composite[id[range[APPLY[CURRY, monoid[x_]]]], inverse[eval[e[monoid[x_]]]] :=
          APPLY[CURRY, monoid[x]]
```

Corollary.

```
In[45]:= composite[eval[e[monoid[x]]], id[range[APPLY[CURRY, monoid[x]]]]] // DoubleInverse

Out[45]= composite[eval[e[monoid[x]]], id[range[APPLY[CURRY, monoid[x]]]]] ==
          inverse[APPLY[CURRY, monoid[x]]]

In[46]:= composite[eval[e[monoid[x_]]], id[range[APPLY[CURRY, monoid[x_]]]]] :=
          inverse[APPLY[CURRY, monoid[x]]]
```

Theorem. The regular representation is faithful.

```
In[47]:= SubstTest[FUNCTION, composite[eval[e[monoid[x]]], id[t]],
           t -> range[APPLY[CURRY, monoid[x]]]] // Reverse

Out[47]= FUNCTION[inverse[APPLY[CURRY, monoid[x]]]] == True

In[48]:= FUNCTION[inverse[APPLY[CURRY, monoid[x_]]]] := True
```

Corollary.

```
In[49]:= SubstTest[member, pair[domain[oopart[t]], range[oopart[t]]],
           Q, t -> APPLY[CURRY, monoid[x]]] // Reverse

Out[49]= member[pair[range[monoid[x]], range[APPLY[CURRY, monoid[x]]], Q] == True

In[50]:= member[pair[range[monoid[x_]], range[APPLY[CURRY, monoid[x_]]], Q] := True
```