the clock unary operation

Lee Martie and Johan Belinfante
2006 July 13

```
In[1]:= SetDirectory["l:"]; << goedel83.11b; << tools.m

        :Package Title: goedel83.11b         2006 July 11 at 5:40 p.m.

        It is now:  2006 Jul 13 at 12:5

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 July 8

        weightlimit = 40
```

summary

Birkhoff and Bartee illustrate the concept of unary operation with an example that they call the clock algebra.

```
In[2]:= "Garrett Birkhoff and Thomas C. Bartee, Modern
           Applied Algebra, McGraw-Hill Book Company, 1970. See page 49.";
```

In this notebook, two constructions of the clock algebra are related, and some of its basic properties are derived.

definition

One way to construct the clock algebra is to combine the successor function with reduction modulo **x**.

```
In[3]:= member[x_, clock[y_]] :=
         and[equal[natmod[succ[first[x]], y], second[x]], member[first[x], y]]
```

Lemma.

```
In[4]:= Map[equal[0, #] &, dif[clock[x], cart[V, V]] // Normality]

Out[4]= subclass[clock[x], cart[V, V]] == True

In[5]:= subclass[clock[x_], cart[V, V]] := True
```

Corollary.

```
In[6]:= equal[composite[Id, clock[x]], clock[x]]

Out[6]= True
```

```
In[7]:= composite[Id, clock[x_]] := clock[x]
```

Lemma.

```
In[8]:= symdif[clock[x], composite[modulo[x], SUCC, id[x]]] // RelnNormality

Out[8]= union[composite[clock[x], id[complement[x]]],
          composite[intersection[clock[x], composite[complement[modulo[x]], SUCC]], id[x]],
          composite[intersection[complement[clock[x]], composite[modulo[x], SUCC]], id[x]]] == 0

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

A formula for the clock algebra results:

```
In[10]:= SubstTest[equal, 0, composite[Id, symdif[u, v]],
           {u -> composite[modulo[x], SUCC, id[x]], v → clock[x]}]

Out[10]= True == equal[clock[x], composite[modulo[x], SUCC, id[x]]]

In[11]:= composite[modulo[x_], SUCC, id[x_]] := clock[x]
```

From this formula, one can derive various properties:

```
In[12]:= IminComp[composite[modulo[x], SUCC], id[x], V]

Out[12]= domain[clock[x]] == nat[x]

In[13]:= domain[clock[x_]] := nat[x]

In[14]:= SubstTest[FUNCTION, composite[modulo[x], funpart[y], id[x]], y → SUCC]

Out[14]= FUNCTION[clock[x]] == True

In[15]:= FUNCTION[clock[x_]] := True
```

Corollary. The clock is a set.

```
In[16]:= SubstTest[member, domain[funpart[w]], V, w → clock[x]] // Reverse

Out[16]= member[clock[x], V] == True

In[17]:= member[clock[x_], V] := True
```

clock[x] is a unary operation

Lemma.

```
In[18]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
           {u -> image[modulo[x], image[SUCC, x]], v → range[modulo[x]], w -> nat[x]}]

Out[18]= or[equal[0, x], subclass[image[modulo[x], image[SUCC, x]], nat[x]]] == True
```

```
In[19]:= (% /. x → x_) /. Equal → SetDelayed
```

The same conclusion holds when **x** s zero, so one can get rid of the literal **equal[0,x]**.

```
In[20]:= SubstTest[and, implies[p1, p2], or[p1, p2],
           {p1 → equal[0, x], p2 -> subclass[image[modulo[x], image[SUCC, x]], nat[x]]}] // Reverse
Out[20]= subclass[image[modulo[x], image[SUCC, x]], nat[x]] == True

In[21]:= (% /. x → x_) /. Equal → SetDelayed
```

This clock is a unary operation:

```
In[22]:= Map[subclass[#, nat[x]] &, ImageComp[composite[modulo[x], SUCC], id[x], V]]
Out[22]= subclass[range[clock[x]], nat[x]] == True

In[23]:= (% /. x → x_) /. Equal → SetDelayed
```

Corollary.

```
In[24]:= member[clock[x], map[nat[x], nat[x]]] // AssertTest
Out[24]= member[clock[x], map[nat[x], nat[x]]] == True

In[25]:= member[clock[x_], map[nat[x_], nat[x_]]] := True
```

Corollary.

```
In[26]:= SubstTest[member, y, map[domain[y], domain[y]], y → clock[x]] // Reverse
Out[26]= member[clock[x], UNOPS] == True

In[27]:= member[clock[x_], UNOPS] := True
```

clock[x] is one-to-one

It is a bit tricky to show that **clock[x]** is one-to-one. The basic idea is to use the pigeon-hole principle, showing that its domain and range are equal. The finiteness is easy:

```
In[28]:= SubstTest[member, domain[funpart[y]], FINITE, y → clock[x]] // Reverse
Out[28]= member[clock[x], FINITE] == True

In[29]:= member[clock[x_], FINITE] := True
```

The range of **clock[x]** gets rewritten when one adds a **nat** wrapper. So, if one wants to proceed with such wrappers, one needs this corollary of an inclusion derived in the preceding section.

```
In[30]:= SubstTest[subclass, image[modulo[w], image[SUCC, w]], nat[w], w → nat[x]]

Out[30]= subclass[
           image[modulo[nat[x]], intersection[complement[set[0]], succ[nat[x]]]], nat[x]] == True

In[31]:= (% /. x → x_) /. Equal → SetDelayed
```

The reverse inclusion must also be shown. One can start with these observations:

```
In[32]:= SubstTest[image, modulo[nat[x]], union[u, v], {u → nat[x], v → set[nat[x]]}]

Out[32]= image[modulo[nat[x]], succ[nat[x]]] == union[nat[x], set[0]]

In[33]:= image[modulo[nat[x_]], succ[nat[x_]]] := union[nat[x], set[0]]

In[34]:= SubstTest[image, modulo[nat[x]], union[u, v],
           {u → dif[succ[nat[x]], set[0]], v → set[0]}] // Reverse

Out[34]= union[image[modulo[nat[x]], intersection[complement[set[0]], succ[nat[x]]]], set[0]] ==
          union[nat[x], set[0]]

In[35]:= union[image[modulo[nat[x_]], intersection[complement[set[0]], succ[nat[x_]]]],
           set[0]] := union[nat[x], set[0]]

In[36]:= Map[union[set[0], #] &, ImageComp[composite[modulo[nat[x]], SUCC], id[nat[x]], V]]

Out[36]= union[range[clock[nat[x]]], set[0]] == union[nat[x], set[0]]

In[37]:= union[range[clock[nat[x_]]], set[0]] := union[nat[x], set[0]]
```

Lemma. If **nat[x]** is not **0**, then **0** belongs to **range[clock[nat[x]]]**.

```
In[38]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u → set[nat[x]],
           v -> intersection[complement[set[0]], succ[nat[x]]], w → modulo[nat[x]]}]

Out[38]= or[equal[0, nat[x]], member[0,
            image[modulo[nat[x]], intersection[complement[set[0]], succ[nat[x]]]]]] == True

In[39]:= (% /. x → x_) /. Equal → SetDelayed
```

Restatement:

```
In[40]:= Map[or[equal[0, nat[x]], member[0, #]] &,
          ImageComp[composite[modulo[nat[x]], SUCC], id[nat[x]], V]]

Out[40]= or[equal[0, nat[x]], member[0, range[clock[nat[x]]]]] == True

In[41]:= (% /. x → x_) /. Equal → SetDelayed
```

Corollary.

```
In[42]:= SubstTest[implies, and[subclass[u, union[v, set[0]]], member[0, v]],
          subclass[u, v], {u → nat[x], v → range[clock[nat[x]]]}]

Out[42]= or[not[member[0, range[clock[nat[x]]]]], subclass[nat[x], range[clock[nat[x]]]]] == True

In[43]:= (% /. x → x_) /. Equal → SetDelayed
```

This suffices to establish the this inclusion:

```
In[44]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p2, p3], not[implies[p1, p3]], {p1 → not[empty[nat[x]]],
            p2 → member[0, range[clock[nat[x]]]], p3 → subclass[nat[x], range[clock[nat[x]]]]}]]

Out[44]= subclass[nat[x], range[clock[nat[x]]]] == True

In[45]:= (% /. x → x_) /. Equal → SetDelayed
```

The reverse inclusion also holds:

```
In[46]:= SubstTest[subclass, range[clock[w]], nat[w], w → nat[x]]

Out[46]= subclass[range[clock[nat[x]]], nat[x]] == True

In[47]:= (% /. x → x_) /. Equal → SetDelayed
```

The two inclusions can be combined into an equation:

```
In[48]:= SubstTest[and, subclass[u, v], subclass[v, u],
           {u → range[clock[nat[x]]], v -> nat[x]}] // Reverse

Out[48]= equal[nat[x], range[clock[nat[x]]]] == True

In[49]:= range[clock[nat[x_]]] := nat[x]
```

The next step is to remove the **nat** wrapper.

```
In[50]:= SubstTest[implies, equal[x, nat[y]], equal[range[clock[x]], nat[x]], y → x]

Out[50]= or[equal[nat[x], range[clock[x]]], not[member[x, omega]]] == True

In[51]:= (% /. x → x_) /. Equal → SetDelayed
```

The same conclusion can be derived when **x** is not a number:

```
In[52]:= ImageComp[composite[modulo[x], SUCC], id[x], V] // Reverse

Out[52]= image[modulo[x], image[SUCC, x]] == range[clock[x]]

In[53]:= image[modulo[x_], image[SUCC, x_]] := range[clock[x]]

In[54]:= SubstTest[implies, and[equal[0, z], not[member[x, omega]]],
          equal[image[z, image[SUCC, x]], nat[x]], z → modulo[x]]

Out[54]= or[equal[nat[x], range[clock[x]]], member[x, omega]] == True
```

```
In[55]:= (% /. x → x_) /. Equal → SetDelayed
```

Combining the cases of numbers and non-numbers yields:

```
In[56]:= SubstTest[and, implies[p1, p2], or[p1, p2],
          {p1 → member[x, omega], p2 → equal[range[clock[x]], nat[x]]}]

Out[56]= True == equal[nat[x], range[clock[x]]]

In[57]:= range[clock[x_]] := nat[x]
```

Applying the pigeonhole principle, one finds that **clock[x]** is also one-to-one.

```
In[58]:= SubstTest[implies, and[FUNCTION[w], member[w, FINITE], equal[domain[w], range[w]]],
          FUNCTION[inverse[w]],
          w → clock[x]]

Out[58]= FUNCTION[inverse[clock[x]]] == True

In[59]:= FUNCTION[inverse[clock[x_]]] := True
```

another construction of the clock algebra

One can construct a clock algebra as the function **union[cart[set[nat[x]], set[0]], composite[SUCC, id[nat[x]]]]** which takes each number less than **nat[x]** to its successor, and **nat[x]** itself to **0**. Lee Martie made the following key observation about this construction:

```
In[60]:= SubstTest[dif, union[cart[set[nat[x]], set[0]], composite[SUCC, id[nat[x]]]],
          composite[modulo[succ[nat[x]]], SUCC, id[y]], y → succ[nat[x]]]

Out[60]= composite[intersection[SUCC, complement[clock[succ[nat[x]]]]], id[nat[x]]] == 0

In[61]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[62]:= SubstTest[equal, 0, dif[u, v],
            {u → union[cart[set[nat[x]], set[0]], composite[SUCC, id[nat[x]]]],
             v -> clock[succ[nat[x]]]}] // Reverse

Out[62]= subclass[composite[SUCC, id[nat[x]]], clock[succ[nat[x]]]] == True

In[63]:= (% /. x → x_) /. Equal → SetDelayed
```

The equality of the two clocks follows:

```
In[64]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
          equal[u, composite[v, id[domain[u]]]],
          {u → union[cart[set[nat[x]], set[0]], composite[SUCC, id[nat[x]]]],
           v → clock[succ[nat[x]]]}]

Out[64]= equal[clock[succ[nat[x]]],
           union[cart[set[nat[x]], set[0]], composite[SUCC, id[nat[x]]]]] == True

In[65]:= union[cart[set[nat[x_]], set[0]], composite[SUCC, id[nat[x_]]]] := clock[succ[nat[x]]]
```

This is a unary operation, with domain and range equal to **succ[nat[x]]**.

```
In[66]:= SubstTest[member, clock[w], map[nat[w], nat[w]], w → succ[nat[x]]]

Out[66]= member[clock[succ[nat[x]]], map[succ[nat[x]], succ[nat[x]]]] == True

In[67]:= member[clock[succ[nat[x_]]], map[succ[nat[x_]], succ[nat[x_]]]] := True
```

---

reify rule

```
In[68]:= SubstTest[reify, x, composite[modulo[f[x]], z, id[f[x]]], z → SUCC]

Out[68]= reify[x, clock[f[x]]] == composite[SWAP, RIF, id[cart[V, SUCC]],
           intersection[composite[inverse[SECOND], inverse[FIRST], reify[x, f[x]]], composite[
             inverse[FIRST], SWAP, inverse[rotate[NATMOD]], VERTSECT[reify[x, f[x]]]]]]

In[69]:= reify[x_, clock[y_]] := composite[SWAP, RIF, id[cart[V, SUCC]],
           intersection[composite[inverse[SECOND], inverse[FIRST], reify[x, y]],
            composite[inverse[FIRST], SWAP, inverse[rotate[NATMOD]], VERTSECT[reify[x, y]]]]]
```