iterating clock[x]

Johan G. F. Belinfante and Lee Martie
2006 July 23

```
In[1]:= SetDirectory["l:"]; << goedel83.20a; << tools.m

        :Package Title: goedel83.20a         2006 July 20 at 5:00 p.m.

        It is now:  2006 Jul 23 at 16:29

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 July 18

        weightlimit = 40
```

summary

A formula for **iterate[clock[x], set[0]]** is derived for the case that **x** is a nonzero natural number.

a generic way to remove succ[nat[x]]

When deriving results about natural numbers it is often convenient to use the **nat** wrapper, defined by

```
In[2]:= intersection[x, image[V, intersection[omega, set[x]]]]

Out[2]= nat[x]
```

The set **nat[x]** is equal to **x** itself when **x** is a natural number, and is **0** otherwise. Since the empty set is a natural number, it follows that **nat[x]** is a natural number for all **x**. One can think of **nat[x]** as a generic natural number. By using **nat** wrappers one can omit numberhood literals, yielding compact formulations of theorems and rewrite rules. It is also easy to go back, eliminating the **nat** wrappers in favor of literals when one desires to do so. To do so, one uses this fact:

```
In[3]:= equal[x, nat[x]]

Out[3]= member[x, omega]
```

When working with **clock[x]**, one often encounters statements of the following form, in which **nat** is combined with **succ**.

```
In[4]:= prop[succ[nat[x_]]] := True
```

In this section it is shown in a generic manner how to eliminate both **succ** and **nat** at the same time, replacing them with two literals. The first step is to use **SubstTest**, replacing the natural number **nat[x]** with its predecessor:

```
In[5]:= SubstTest[prop, succ[nat[y]], y → U[nat[x]]]

Out[5]= prop[union[nat[x], set[0]]] == True

In[6]:= prop[union[nat[x_], set[0]]] := True
```

The next step uses **SubstTest** and equality substitution to eliminate the **union**:

```
In[7]:= SubstTest[implies, equal[y, union[nat[x], set[0]]], prop[y], y → nat[x]]

Out[7]= or[equal[0, nat[x]], prop[nat[x]]] == True

In[8]:= or[equal[0, nat[x_]], prop[nat[x_]]] := True
```

The final step is just the standard method for removing **nat** wrappers, which again uses **SubstTest** and equality substitution.

```
In[9]:= SubstTest[implies, equal[x, nat[y]], or[equal[0, x], prop[x]], y → x]

Out[9]= or[equal[0, x], not[member[x, omega]], prop[x]] == True

In[10]:= or[equal[0, x_], not[member[x_, omega]], prop[x_]] := True
```

The next section provides an example of this technique.

---

iterating clock

In this section, a formula for **iterate[clock[x], set[0]]** is derived. The main tool is the uniqueness theorem for **iterate**, which states:

```
In[11]:= implies[and[equal[composite[u, w], composite[w, SUCC]],
                     equal[image[w, set[0]], v]],
                 equal[composite[w, id[omega]], iterate[u, v]]]

Out[11]= True
```

Lemma 1.

```
In[12]:= SubstTest[composite, modulo[x], plus[y], modulo[x], y → set[0]]

Out[12]= composite[modulo[x], SUCC, modulo[x]] == composite[modulo[x], SUCC]

In[13]:= composite[modulo[x_], SUCC, modulo[x_]] := composite[modulo[x], SUCC]
```

Lemma 2.

```
In[14]:= Map[equal[#, composite[clock[succ[nat[x]]], modulo[succ[nat[x]]]]] &,
          Assoc[composite[modulo[succ[nat[x]]], SUCC], id[succ[nat[x]]], modulo[succ[nat[x]]]]]

Out[14]= equal[composite[clock[succ[nat[x]]], modulo[succ[nat[x]]]],
           composite[modulo[succ[nat[x]]], SUCC]] == True

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

The uniqueness theorem for **iterate** now yields this proposition, which involves **succ[nat[x]]**.

```
In[16]:= SubstTest[implies,
          and[equal[composite[u, w], composite[w, SUCC]], equal[image[w, set[0]], v]],
          equal[composite[w, id[omega]], iterate[u, v]],
          {u → clock[succ[nat[x]]], v → set[0], w → modulo[succ[nat[x]]]}]

Out[16]= equal[iterate[clock[succ[nat[x]]], set[0]], modulo[succ[nat[x]]]] == True

In[17]:= iterate[clock[succ[nat[x_]]], set[0]] := modulo[succ[nat[x]]]
```

The method of the preceding section is now used to get rid of **succ[nat[x]]**. This is the first step:

```
In[18]:= Map[equal[#, modulo[union[nat[x], set[0]]]] &,
          SubstTest[iterate, clock[succ[nat[x]]], set[0], x → U[nat[x]]]]

Out[18]= equal[iterate[clock[union[nat[x], set[0]]], set[0]],
           modulo[union[nat[x], set[0]]]] == True

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

Second step.

```
In[20]:= SubstTest[implies, equal[y, union[nat[x], set[0]]],
          equal[iterate[clock[y], set[0]], modulo[y]], y → nat[x]]

Out[20]= or[equal[0, nat[x]], equal[iterate[clock[nat[x]], set[0]], modulo[nat[x]]]] == True

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

Third step.

```
In[22]:= SubstTest[implies, equal[y, nat[x]],
          or[equal[0, y], equal[iterate[clock[y], set[0]], modulo[y]]], y → x]

Out[22]= or[equal[0, x],
           equal[iterate[clock[x], set[0]], modulo[x]], not[member[x, omega]]] == True

In[23]:= or[equal[0, x],
           equal[iterate[clock[x], set[0]], modulo[x]], not[member[x, omega]]] := True
```

This theorem states that **iterate[clock[x],set[0]]** is equal to **modulo[x]** when **x** is a nonzero natural number.

a corollary about clock-invariant sets

When **f** is a function, and **s** is a set, the range of **iterate[f, set[s]]** is the smallest set which is invariant under **f** and holds **s**. Applying this theorem to the **iterate** formula derived in the preceding section, one finds:

```
In[24]:= SubstTest[range, iterate[funpart[w], set[0]], w -> clock[succ[nat[x]]]] // Reverse

Out[24]= hull[invar[clock[succ[nat[x]]], set[0]] == succ[nat[x]]
```

```
In[25]:= hull[invar[clock[succ[nat[x_]]]], set[0]] := succ[nat[x]]
```

Once again one can obtain a wrapper-free formulation using three applications of **SubstTest**. The first step:

```
In[26]:= Map[equal[#, union[nat[x], set[0]]] &,
          SubstTest[hull, invar[clock[succ[nat[y]]]], set[0], y → U[nat[x]]]]

Out[26]= equal[hull[invar[clock[union[nat[x], set[0]]]], set[0]], union[nat[x], set[0]]] == True

In[27]:= (% /. x → x_) /. Equal → SetDelayed
```

Step 2:

```
In[28]:= SubstTest[implies, equal[y, union[nat[x], set[0]]],
          equal[hull[invar[clock[y]], set[0]], y], y → nat[x]]

Out[28]= or[equal[0, nat[x]], equal[hull[invar[clock[nat[x]]], set[0]], nat[x]]] == True

In[29]:= (% /. x → x_) /. Equal → SetDelayed
```

Step 3.

```
In[30]:= SubstTest[implies, equal[x, nat[y]],
          or[equal[0, x], equal[hull[invar[clock[x]], set[0]], x]], y → x]

Out[30]= or[equal[0, x], equal[x, hull[invar[clock[x]], set[0]]], not[member[x, omega]]] == True

In[31]:= or[equal[0, x_],
           equal[x, hull[invar[clock[x_]], set[0]]], not[member[x_, omega]]] := True
```

This theorem says that if **x** is any nonzero natural number, then **x** is the smallest **clock[x]**-invariant set that holds **0**.