setting a clock back

Johan G. F. Belinfante
2009 November 3

```
In[1]:= SetDirectory["l:"]; << goedel.09nov02a;<< tools.m

        :Package Title: goedel.09nov02a            2009 November 2 at 10:45 p.m.

        It is now:  2009 Nov 3 at 15:35

        Loading Simplification Rules

        TOOLS.M                                    Revised 2009 November 2

        weightlimit = 40
```

summary

One can set a clock back by advancing it sufficiently far forward: the inverse of **clock[x]** is a subclass of **trv[clock[x]]**. This observation is used to prove that the transitive closure of **clock[x]** is the cartesian square of **nat[x]**. A variable-free reformulation of this result is derived. An elementary application of these results is to provide a counterexample showing that an asymmetric relation need not be acyclic. Rewrite rules for **clock[x]** are generally simpler when **x** is replaced by **succ[nat[x]]**. The general case can sometimes be recovered from this special case by additional reasoning.

inverse[clock[x]]

Rewrite rules with a **nat[x]** wrapper do not automatically apply to the case of the compound wrapper **succ[nat[x]]**, but one can easily derive such rules when needed.

Lemma. A simplification rule.

```
In[2]:= SubstTest[composite, modulo[nat[t]], id[nat[t]], t → succ[nat[x]]] // Reverse

Out[2]= composite[modulo[succ[nat[x]]], id[succ[nat[x]]]] == id[succ[nat[x]]]

In[3]:= composite[modulo[succ[nat[x_]]], id[succ[nat[x_]]]] := id[succ[nat[x]]]
```

Lemma. An inclusion. (Comment. The non-emptiness literal can not be omitted here.)

```
In[4]:= SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]],
          {t -> power[clock[succ[nat[x]]]], u → set[nat[x]], v → complement[set[0]]}] // Reverse

Out[4]= or[equal[0, nat[x]], subclass[
           image[power[clock[succ[nat[x]]]], set[nat[x]]], trv[clock[succ[nat[x]]]]]] == True
```

```
In[5]:= (% /. x → x_) /. Equal→ SetDelayed
```

Corollary.

```
In[6]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
          not[implies[p1, p3]], {p1 → not[empty[nat[x]]], p2 -> subclass[
             image[power[clock[succ[nat[x]]]], set[nat[x]]], trv[clock[succ[nat[x]]]]],
           p3 -> subclass[image[power[clock[succ[nat[x]]]], set[nat[x]]],
             cart[succ[nat[x]], V]]}]] // Reverse

Out[6]= or[equal[0, nat[x]], subclass[
          image[power[clock[succ[nat[x]]]], set[nat[x]]], cart[succ[nat[x]], V]]] == True
```

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

Lemma. A simplification rule.

```
In[8]:= SubstTest[composite, modulo[nat[t]], plus[nat[t]], t → succ[nat[x]]] // Reverse

Out[8]= composite[modulo[succ[nat[x]]], SUCC, plus[nat[x]]] == modulo[succ[nat[x]]]

In[9]:= composite[modulo[succ[nat[x_]]], SUCC, plus[nat[x_]]] := modulo[succ[nat[x]]]
```

Corollary.

```
In[10]:= SubstTest[composite, composite[modulo[succ[nat[x]]], plus[nat[x]], id[succ[nat[x]]]],
           composite[modulo[succ[nat[x]]], plus[nat[y]], id[succ[nat[x]]]],
           y → set[0]] // Reverse

Out[10]= composite[modulo[succ[nat[x]]], plus[nat[x]], clock[succ[nat[x]]]] == id[succ[nat[x]]]

In[11]:= composite[modulo[succ[nat[x_]]], plus[nat[x_]], clock[succ[nat[x_]]]] :=
          id[succ[nat[x]]]
```

Theorem. An explicit formula for the inverse of **clock[succ[nat[x]]**.

```
In[12]:= Assoc[composite[modulo[succ[nat[x]]], plus[nat[x]]],
          clock[succ[nat[x]]], inverse[clock[succ[nat[x]]]]]

Out[12]= composite[modulo[succ[nat[x]]], plus[nat[x]], id[succ[nat[x]]]] ==
          inverse[clock[succ[nat[x]]]]

In[13]:= composite[modulo[succ[nat[x_]]], plus[nat[x_]], id[succ[nat[x_]]]] :=
          inverse[clock[succ[nat[x]]]]
```

Lemma. A rewrite rule that eliminates the expression **power[SUCC]**.

```
In[14]:= Assoc[modulo[x], id[omega], image[power[SUCC], y]] // Reverse

Out[14]= composite[modulo[x], image[power[SUCC], y]] ==
          composite[modulo[x], NATADD, id[cart[V, y]], inverse[FIRST]]

In[15]:= composite[modulo[x_], image[power[SUCC], y_]] :=
          composite[modulo[x], NATADD, id[cart[V, y]], inverse[FIRST]]
```

Corollary. The inverse of **clock[succ[nat[x]]** is the **nat[x]**-th power of **clock[succ[nat[x]]**.

```
In[16]:= SubstTest[composite, image[power[t, u], iterate[t, v], id[w],
           {t → clock[succ[nat[x]]], u → set[nat[x]], v → set[0], w → succ[nat[x]]}] // Reverse

Out[16]= composite[image[power[clock[succ[nat[x]]]], set[nat[x]]], id[succ[nat[x]]] ==
          inverse[clock[succ[nat[x]]]]

In[17]:= composite[image[power[clock[succ[nat[x_]]]], set[nat[x_]]], id[succ[nat[x_]]]] :=
          inverse[clock[succ[nat[x]]]]
```

Corollary. A special case needed later.

```
In[18]:= SubstTest[composite, image[power[clock[succ[nat[x]]]], set[nat[x]]],
           id[succ[nat[x]]], x → succ[set[0]]] // Reverse

Out[18]= composite[clock[succ[succ[set[0]]]], clock[succ[succ[set[0]]]]] ==
          inverse[clock[succ[succ[set[0]]]]]

In[19]:= composite[clock[succ[succ[set[0]]]], clock[succ[succ[set[0]]]]] :=
          inverse[clock[succ[succ[set[0]]]]]
```

The following observation is used to derive the next lemma.

```
In[20]:= abstract[t, composite[image[power[clock[succ[nat[x]]]], t], id[succ[nat[x]]]]]

Out[20]= composite[id[cart[succ[nat[x]], V]], power[clock[succ[nat[x]]]]]
```

Lemma.

```
In[21]:= SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]],
           {t -> composite[id[cart[succ[nat[x]], V]], power[clock[succ[nat[x]]]]],
            u → set[nat[x]], v → complement[set[0]]}] // Reverse

Out[21]= or[equal[0, nat[x]],
           subclass[inverse[clock[succ[nat[x]]]], trv[clock[succ[nat[x]]]]]] == True

In[22]:= (% /. x → x_) /. Equal → SetDelayed
```

The non-emptiness literal in this lemma is redundant, and will now be removed.

Theorem.

```
In[23]:= SubstTest[and, implies[p, q], or[p, q], {p -> equal[0, nat[x]],
           q -> subclass[inverse[clock[succ[nat[x]]]], trv[clock[succ[nat[x]]]]]}]

Out[23]= subclass[inverse[clock[succ[nat[x]]]], trv[clock[succ[nat[x]]]]] == True

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

The wrappers can also be removed. This is done in several steps. The first step is to replace **x** with **U[nat[x]]**.

Lemma.

```
In[25]:= SubstTest[implies, equal[x, succ[nat[t]]],
           subclass[inverse[clock[x]], trv[clock[x]]], t → U[nat[x]]] // Reverse

Out[25]= or[not[equal[x, union[nat[x], set[0]]]],
           subclass[inverse[clock[x]], trv[clock[x]]]] == True

In[26]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[27]:= SubstTest[implies, empty[t], subclass[inverse[t], trv[t]], t → clock[x]] // Reverse

Out[27]= or[not[equal[0, nat[x]]], subclass[inverse[clock[x]], trv[clock[x]]]] == True

In[28]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[29]:= SubstTest[implies, equal[x, nat[t]],
           or[equal[0, x], equal[x, union[nat[x], set[0]]]], t → x] // Reverse

Out[29]= or[equal[0, x], equal[x, union[nat[x], set[0]]], not[member[x, omega]]] == True

In[30]:= (% /. x → x_) /. Equal → SetDelayed
```

The general case can now be derived.

Theorem. The inverse of **clock[x]** is a subclass of its transitive closure.

```
In[31]:= Map[not, SubstTest[and, implies[not[p1], p4], implies[p2, p4],
            implies[and[p1, not[p2]], p3], implies[p3, p5], implies[p4, p5], not[p5],
            {p1 → member[x, omega], p2 → empty[x], p3 -> equal[x, union[nat[x], set[0]]],
             p4 → empty[nat[x]], p5 -> subclass[inverse[clock[x]], trv[clock[x]]]}]] // Reverse

Out[31]= subclass[inverse[clock[x]], trv[clock[x]]] == True

In[32]:= subclass[inverse[clock[x_]], trv[clock[x_]]] := True
```

trv[clock[x]] is a cartesian square

Since transitive closure is monotone, it follows that **trv[clock[x]]** is symmetric.

Corollary.

```
In[33]:= SubstTest[implies, subclass[u, v], subclass[trv[u], trv[v]],
           {u → inverse[clock[x]], v → trv[clock[x]]}] // Reverse

Out[33]= subclass[inverse[trv[clock[x]]], trv[clock[x]]] == True

In[34]:= (% /. x → x_) /. Equal → SetDelayed
```

The next step will be to derive a formula for the vertical section of **trv[clock[succ[nat[x]]]]** at **0**.

Lemma.

```
In[35]:= SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]],
           {t -> modulo[succ[nat[x]]], u → set[succ[nat[x]]], v → complement[set[0]]}] // Reverse

Out[35]= member[0, image[modulo[succ[nat[x]]], complement[set[0]]]] == True

In[36]:= member[0, image[modulo[succ[nat[x_]]], complement[set[0]]]] := True
```

Theorem.

```
In[37]:= SubstTest[image, modulo[succ[nat[x]]],
          union[u, v], {u → set[0], v → complement[set[0]]}]

Out[37]= image[modulo[succ[nat[x]]], complement[set[0]]] == succ[nat[x]]

In[38]:= image[modulo[succ[nat[x_]]], complement[set[0]]] := succ[nat[x]]
```

Corollary. An explicit formula for the vertical section of **trv[clock[succ[nat[x]]]]** at **0**.

```
In[39]:= SubstTest[image, iterate[u, v],
          complement[set[0]], {u → clock[succ[nat[x]]], v → set[0]}]

Out[39]= image[trv[clock[succ[nat[x]]]], set[0]] == succ[nat[x]]

In[40]:= image[trv[clock[succ[nat[x_]]]], set[0]] := succ[nat[x]]
```

Since **trv[clock[succ[nat[x]]]]** is symmetric, this is also the horizontal section of **trv[clock[succ[nat[x]]]]** at **0**.

Lemma.

```
In[41]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]],
           subclass[u, w], {u -> cart[succ[nat[x]], set[0]],
            v → inverse[trv[clock[succ[nat[x]]]]], w -> trv[clock[succ[nat[x]]]]}] // Reverse

Out[41]= and[member[pair[nat[x], 0], trv[clock[succ[nat[x]]]]],
           subclass[nat[x], image[inverse[trv[clock[succ[nat[x]]]]], set[0]]]] == True

In[42]:= (% /. x → x_) /. Equal → SetDelayed
```

The composite of these horizontal and vertical sections is the cartesian square of **succ[nat[x]]**.

Lemma. An inclusion.

```
In[43]:= SubstTest[implies, and[subclass[u, trv[w]], subclass[v, trv[w]]],
            subclass[composite[u, v], trv[w]], {u → cart[set[0], succ[nat[x]]],
             v -> cart[succ[nat[x]], set[0]], w → clock[succ[nat[x]]]}] // Reverse // MapNotNot

Out[43]= subclass[cart[succ[nat[x]], succ[nat[x]]], trv[clock[succ[nat[x]]]]] == True

In[44]:= (% /. x → x_) /. Equal → SetDelayed
```

This inclusion can be sharpened to an equation.

Corollary.

```
In[45]:= equal[cart[succ[nat[x]], succ[nat[x]]], trv[clock[succ[nat[x]]]]] // AssertTest

Out[45]= equal[cart[succ[nat[x]], succ[nat[x]]], trv[clock[succ[nat[x]]]]] == True

In[46]:= (% /. x → x_) /. Equal → SetDelayed
```

The wrappers will now be removed in that same fashion that was done earlier. The first step is to replace **x** with **U[nat[x]]**.

```
In[47]:= SubstTest[implies, equal[x, succ[nat[t]]],
           equal[trv[clock[x]], cartsq[nat[x]]], t → U[nat[x]]] // Reverse

Out[47]= or[equal[cart[nat[x], nat[x]], trv[clock[x]]],
           not[equal[x, union[nat[x], set[0]]]]] == True

In[48]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[49]:= SubstTest[implies, empty[t], equal[trv[t], cartsq[domain[t]]], t → clock[x]] // Reverse

Out[49]= or[equal[cart[nat[x], nat[x]], trv[clock[x]]], not[equal[0, nat[x]]]] == True

In[50]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. The transitive closure of **clock[x]** is the cartesian square of **nat[x]**.

```
In[51]:= Map[not, SubstTest[and, implies[not[p1], p4], implies[p2, p4],
            implies[and[p1, not[p2]], p3], implies[p3, p5], implies[p4, p5], not[p5],
            {p1 → member[x, omega], p2 → empty[x], p3 -> equal[x, union[nat[x], set[0]]],
             p4 → empty[nat[x]], p5 → equal[trv[clock[x]], cart[nat[x], nat[x]]]}]] // Reverse

Out[51]= equal[cart[nat[x], nat[x]], trv[clock[x]]] == True

In[52]:= trv[clock[x_]] := cart[nat[x], nat[x]]
```

A variable-free version of this will be derived using **reify**. The following lemma is needed to clean up the result.

Lemma.

```
In[53]:= IminComp[id[P[cart[V, V]]], CLOCK, V] // Reverse

Out[53]= image[inverse[CLOCK], P[cart[V, V]]] == omega

In[54]:= image[inverse[CLOCK], P[cart[V, V]]] := omega
```

Theorem. Variable-free formulation of the fact that the transitive closure of a clock is a cartesian square.

```
In[55]:= Map[composite[VERTSECT[#], id[omega]] &, SubstTest[reify, x, trv[f[x]], f → clock]]

Out[55]= composite[HULL[TRV], CLOCK] == composite[CART, DUP, id[omega]]
```

```
In[56]:= composite[HULL[TRV], CLOCK] := composite[CART, DUP, id[omega]]
```

application: a counterexample

Corollary.

```
In[57]:= IminComp[HULL[TRV], CLOCK, P[Di]] // Reverse
Out[57]= image[inverse[CLOCK], ACYCLIC] == set[0]

In[58]:= image[inverse[CLOCK], ACYCLIC] := set[0]
```

Corollary.

```
In[59]:= ImageComp[CLOCK, inverse[CLOCK], ACYCLIC]
Out[59]= intersection[ACYCLIC, range[CLOCK]] == set[0]

In[60]:= intersection[ACYCLIC, range[CLOCK]] := set[0]
```

Lemma.

```
In[61]:= (SubstTest[fix, union[u, v], {u → cart[set[nat[x]], set[0]],
              v -> composite[SUCC, id[nat[x]]]}] // Reverse) /. x → succ[set[0]]
Out[61]= fix[clock[succ[succ[set[0]]]]] == 0

In[62]:= fix[clock[succ[succ[set[0]]]]] := 0
```

Counterexample. An asymmetric relation need not be acyclic.

```
In[63]:= Map[not, SubstTest[implies, and[member[t, u], subclass[u, v]],
            member[t, v], {t → clock[succ[succ[set[0]]]],
             u → fix[composite[DISJOINT, INVERSE]], v → ACYCLIC}]] // Reverse
Out[63]= subclass[fix[composite[DISJOINT, INVERSE]], ACYCLIC] == False

In[64]:= subclass[fix[composite[DISJOINT, INVERSE]], ACYCLIC] := False
```

serendipity

In the course of this study some additional rewrite rules were discovered.

Lemma.

```
In[65]:= SubstTest[implies, equal[x, nat[t]], equal[clock[nat[x]], clock[x]], t → x] // Reverse
Out[65]= or[equal[clock[x], clock[nat[x]]], not[member[x, omega]]] == True
```

```
In[66]:= (% /. x → x_) /. Equal → SetDelayed

In[67]:= SubstTest[implies, and[equal[u, v], equal[v, w]],
           equal[u, w], {u → clock[nat[x]], v → 0, w → clock[x]}] // Reverse

Out[67]= or[equal[clock[x], clock[nat[x]]], not[equal[0, nat[x]]]] == True

In[68]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. Automatic removal of a **nat** wrapper.

```
In[69]:= Map[not,
           SubstTest[and, implies[p1, p3], implies[not[p1], p2], not[p3], {p1 → member[x, omega],
             p2 → empty[nat[x]], p3 -> equal[clock[x], clock[nat[x]]]}]] // Reverse

Out[69]= equal[clock[x], clock[nat[x]]] == True

In[70]:= clock[nat[x_]] := clock[x]
```

Theorem.

```
In[71]:= SubstTest[member, APPLY[funpart[t], nat[x]], range[funpart[t]], t → CLOCK] // Reverse

Out[71]= member[clock[x], range[CLOCK]] == True

In[72]:= member[clock[x], range[CLOCK]] := True
```

The following was derived in the course of studying **clock[x]** for **x = 4**.

Theorem. A special case of the fact that any power set belongs to the Russell class.

```
In[73]:= member[succ[set[0]], RUSSELL] // AssertTest

Out[73]= member[succ[set[0]], RUSSELL] == True

In[74]:= member[succ[set[0]], RUSSELL] := True
```