recursive game construction, part 3.

Johan G. F. Belinfante
2012 April 26

```
In[1]:= SetDirectory["l:"]; << goedel.12apr25b

        :Package Title: goedel.12apr25b                  2012 April 25 at 4:00 p.m.

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

        It is now:  2012 Apr 26 at 13:53

        Loading Simplification Rules

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

        weightlimit = 40

        Loading completed.

        It is now:  2012 Apr 26 at 14:11
```

summary

This is the third of a series of notebooks in which the theory of well-founded recursion is used to construct Conway's class **GAMES** of all games. For any set of games, there exists is an ordinal that is greater than the birthdays of all its members. This key observation is used to derive the equation **fix[GAMES] = P[GAMES]**, and hence **P[GAMES] × P[GAMES] = GAMES**. A simple application of transfinite induction is used to show that if **P[x] × P[x] ⊂ x**, then **GAMES ⊂ x**.

derivation

Conway's class of games **GAMES = U[range[gamerec]]** is defined as in terms of the function

gamerec = rec[CART ∘ DUP ∘ POWER ∘ BIGCUP ∘ IMAGE[SECOND] ∘ SECOND, inverse[E] ∘ id[Ω]] ∘ id[Ω]]

whose domain is the class **Ω** of all ordinals, and satisfies the recursion equation

$$\mathbf{APPLY[gamerec, \alpha] = P[U[image[gamerec, \alpha]]] \times P[U[image[gamerec, \alpha]]]}$$

for every ordinal $\alpha \in \Omega$. This function is monotone and its values provide an increasing chain of cartesian squares that are lower bounds for the class **GAMES**.

```
In[7]:= subclass[APPLY[gamerec, ord[t]], GAMES]

Out[7]= True
```

The class **GAMES** is the union of this chain of cartesian squares.

```
In[8]:= U[class[y, exists[x, and[member[x, OMEGA], equal[y, APPLY[gamerec, x]]]]]]

Out[8]= GAMES
```

Lemma. A simplification rule.

```
In[9]:= SubstTest[cartsq, P[U[image[gamerec, ord[t]]]], t → succ[ord[z]]] // Reverse

Out[9]= cart[P[U[image[gamerec, succ[ord[z]]]]], P[U[image[gamerec, succ[ord[z]]]]] ==
         APPLY[gamerec, succ[ord[z]]]

In[10]:= cart[P[U[image[gamerec, succ[ord[z_]]]]], P[U[image[gamerec, succ[ord[z_]]]]] :=
          APPLY[gamerec, succ[ord[z]]]
```

Lemma.

```
In[11]:= SubstTest[subclass, cartsq[s], cartsq[t], {s -> P[U[image[gamerec, ord[z]]]],
            t -> P[U[image[gamerec, succ[ord[z]]]]]}] // Reverse

Out[11]= subclass[APPLY[gamerec, ord[z]], APPLY[gamerec, succ[ord[z]]]] == True

In[12]:= (% /. z → z_) /. Equal→ SetDelayed
```

Lemma. A property of monotone functions.

```
In[13]:= SubstTest[implies, subclass[P[t], monotone[S, S]],
           subclass[U[image[t, ord[z]]], APPLY[t, ord[z]]], t → gamerec] // Reverse

Out[13]= subclass[U[image[gamerec, ord[z]]], APPLY[gamerec, ord[z]]] == True

In[14]:= subclass[U[image[gamerec, ord[z_]]], APPLY[gamerec, ord[z_]]] := True
```

Theorem. Simplification rule.

```
In[15]:= Map[equal[APPLY[gamerec, ord[z]], U[#]] &,
           SubstTest[image, gamerec, union[u, v], {u → ord[z], v → set[ord[z]]}]] // Reverse

Out[15]= equal[APPLY[gamerec, ord[z]], U[image[gamerec, succ[ord[z]]]]] == True

In[16]:= U[image[gamerec, succ[ord[z_]]]] := APPLY[gamerec, ord[z]]
```

persistence of games

Lemma. A statement of the monotonicity property of **gamerec**.

```
In[17]:= SubstTest[implies,
           and[subclass[P[t], monotone[S, S]], subclass[u, v], member[u, domain[t]]],
           subclass[APPLY[t, u], APPLY[t, v]], t → gamerec] // Reverse

Out[17]= or[not[member[u, OMEGA]], not[subclass[u, v]],
           subclass[APPLY[gamerec, u], APPLY[gamerec, v]]] == True

In[18]:= (% /. {u → u_, v → v_}) /. Equal → SetDelayed
```

Lemma. (Specialization of the preceding lemma to the case that **u** is the birthday of a game **g**.)

```
In[19]:= SubstTest[or, not[member[u, OMEGA]], not[subclass[u, v]],
           subclass[APPLY[gamerec, u], APPLY[gamerec, v]], u → APPLY[BDAY, g]] // Reverse

Out[19]= or[not[member[g, GAMES]], not[subclass[APPLY[BDAY, g], v]],
           subclass[APPLY[gamerec, APPLY[BDAY, g]], APPLY[gamerec, v]]] == True

In[20]:= (% /. {g → g_, v → v_}) /. Equal → SetDelayed
```

Lemma. A simplification rule.

```
In[21]:= (member[x, image[inverse[funpart[w]], P[complement[set[g]]]]] // AssertTest) /.
          w → gamerec

Out[21]= member[x, image[inverse[gamerec], P[complement[set[g]]]]] ==
          and[member[x, OMEGA], not[member[g, APPLY[gamerec, x]]]]

In[22]:= member[x_, image[inverse[gamerec], P[complement[set[g_]]]]] :=
          and[member[x, OMEGA], not[member[g, APPLY[gamerec, x]]]]
```

Theorem. A game is born on its birthday.

```
In[23]:= Map[implies[member[g, GAMES], member[#, intersection[OMEGA,
              complement[image[inverse[gamerec], P[complement[set[g]]]]]]]] &,
           SubstTest[APPLY, VERTSECT[t], g, t -> complement[
              composite[complement[inverse[E]], inverse[gamerec], E]]]] // Reverse

Out[23]= or[member[g, APPLY[gamerec, APPLY[BDAY, g]]], not[member[g, GAMES]]] == True

In[24]:= or[member[g_, APPLY[gamerec, APPLY[BDAY, g_]]], not[member[g_, GAMES]]] := True
```

Theorem. Once a game is born, it persists forever.

```
In[25]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p2, p3], p4], implies[p1, p5],
            implies[and[p4, p5], p6], not[implies[and[p1, p3], p6]], {p1 → member[g, GAMES],
             p2 → member[APPLY[BDAY, g], OMEGA], p3 → subclass[APPLY[BDAY, g], ord[v]],
             p4 → subclass[APPLY[gamerec, APPLY[BDAY, g]], APPLY[gamerec, ord[v]]],
             p5 → member[g, APPLY[gamerec, APPLY[BDAY, g]]],
             p6 -> member[g, APPLY[gamerec, ord[v]]]}]] // Reverse

Out[25]= or[member[g, APPLY[gamerec, ord[v]]],
           not[member[g, GAMES]], not[subclass[APPLY[BDAY, g], ord[v]]]] == True

In[26]:= (% /. {g → g_, v → v_}) /. Equal → SetDelayed
```

Corollary. (Eliminate the **ord** wrapper.)

```
In[27]:= SubstTest[implies, equal[v, ord[t]], or[member[g, APPLY[gamerec, v]],
           not[member[g, GAMES]], not[subclass[APPLY[BDAY, g], v]]], t → v] // Reverse

Out[27]= or[member[g, APPLY[gamerec, v]], not[member[g, GAMES]],
           not[member[v, OMEGA]], not[subclass[APPLY[BDAY, g], v]]] == True

In[28]:= (% /. {g → g_, v → v_}) /. Equal → SetDelayed
```

The plan now is to use the fact that if **s** is any set of games, then there is an ordinal greater than all the birthdays of its members **g ∈ s**.

Lemma. Specialization of the preceding result to **v = tc[image[BDAY, s]]**

```
In[29]:= SubstTest[or, member[g, APPLY[gamerec, v]],
           not[member[g, GAMES]], not[member[v, OMEGA]],
           not[subclass[APPLY[BDAY, g], v]], v → tc[image[BDAY, s]]] // Reverse

Out[29]= or[member[g, APPLY[gamerec, tc[image[BDAY, s]]]],
           not[member[g, GAMES]], not[member[image[BDAY, s], V]],
           not[subclass[APPLY[BDAY, g], tc[image[BDAY, s]]]]] == True

In[30]:= (% /. {g → g_, s → s_}) /. Equal → SetDelayed
```

Lemma. If a game **g** belongs to a set **s** of games, then the birthday of **g** belongs to the set of birthdays of members of **s**.

```
In[31]:= SubstTest[implies, and[FUNCTION[t], member[g, s], member[g, domain[t]]],
           member[APPLY[t, g], image[t, s]], t → BDAY] // Reverse

Out[31]= or[member[APPLY[BDAY, g], image[BDAY, s]],
           not[member[g, GAMES]], not[member[g, s]]] == True

In[32]:= (% /. {g → g_, s → s_}) /. Equal → SetDelayed
```

Theorem. Every game **g** in a set of games **s** is alive at the time **tc[image[BDAY, s]] ∈ Ω**.

```
In[33]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
            implies[p2, p4], implies[and[p1, p3], p5], implies[p5, p6],
            (*implies[and[p3,p4,p6],p7],*) not[implies[and[p1, p2], p7]],
            {p1 → member[g, s], p2 → member[s, P[GAMES]], p3 → member[g, GAMES],
             p4 -> member[image[BDAY, s], V], p5 → member[APPLY[BDAY, g], image[BDAY, s]],
             p6 -> subclass[APPLY[BDAY, g], tc[image[BDAY, s]]],
             p7 -> member[g, APPLY[gamerec, tc[image[BDAY, s]]]]}]] // Reverse

Out[33]= or[member[g, APPLY[gamerec, tc[image[BDAY, s]]]],
           not[member[g, s]], not[member[s, V]], not[subclass[s, GAMES]]] == True

In[34]:= (% /. {g → g_, s → s_}) /. Equal → SetDelayed
```

Lemma. A simplification rule.

```
In[35]:= SubstTest[fix, cartsq[t], t → P[U[image[gamerec, ord[z]]]]] // Reverse

Out[35]= fix[APPLY[gamerec, ord[z]]] == P[U[image[gamerec, ord[z]]]]

In[36]:= fix[APPLY[gamerec, ord[z_]]] := P[U[image[gamerec, ord[z]]]]
```

Theorem.

```
In[37]:= SubstTest[implies, subclass[u, v], subclass[fix[u], fix[v]],
           {u → APPLY[gamerec, ord[z]], v → GAMES}] // Reverse

Out[37]= subclass[P[U[image[gamerec, ord[z]]]], fix[GAMES]] == True

In[38]:= subclass[P[U[image[gamerec, ord[z_]]]], fix[GAMES]] := True
```

Lemma. If there is a time when all games in a set **s** of games are all alive, then **s ⊂ fix[GAMES]**.

```
In[39]:= (SubstTest[implies, and[member[u, v], subclass[v, w]], member[u, w], {u → s,
             v -> P[U[image[gamerec, ord[u]]]], w -> fix[GAMES]}] // Reverse) /. u → succ[ord[t]]

Out[39]= or[member[s, fix[GAMES]], not[member[s, V]],
           not[subclass[s, APPLY[gamerec, ord[t]]]]] == True

In[40]:= (% /. {s → s_, t → t_}) /. Equal → SetDelayed
```

Lemma. (Remove the **ord** wrapper from the preceding lemma, and specialize to the time **tc[image[BDAY, s]]**.)

```
In[41]:= (SubstTest[implies, equal[t, ord[w]], or[member[s, fix[GAMES]], not[member[s, V]],
             not[subclass[s, APPLY[gamerec, t]]]], w → t] // Reverse) /. t → tc[image[BDAY, s]]

Out[41]= or[member[s, fix[GAMES]], not[member[s, V]], not[member[image[BDAY, s], V]],
           not[subclass[s, APPLY[gamerec, tc[image[BDAY, s]]]]]] == True

In[42]:= (% /. s → s_) /. Equal → SetDelayed
```

Lemma. For any set **s** of games, there is a time $\alpha \in \Omega$ when all members of **s** are alive: **s ⊂ APPLY[r, α]**.

```
In[43]:= Map[equal[V, domain[#]] &, SubstTest[reify, g,
           case[implies[and[member[g, s], member[s, P[GAMES]]], member[g, t]]],
           t -> APPLY[gamerec, tc[image[BDAY, s]]]]]

Out[43]= or[not[member[s, V]], not[subclass[s, GAMES]],
           subclass[s, APPLY[gamerec, tc[image[BDAY, s]]]]] == True

In[44]:= (% /. s → s_) /. Equal → SetDelayed
```

The next theorem removes the explicit reference to the birthday function.

Theorem.

```
In[45]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p1, p2, p3], p4],
           not[implies[p1, p4]], {p1 → member[s, P[GAMES]], p2 -> member[image[BDAY, s], V],
            p3 -> subclass[s, APPLY[gamerec, tc[image[BDAY, s]]]],
            p4 → member[s, fix[GAMES]]}]] // Reverse

Out[45]= or[member[s, fix[GAMES]], not[member[s, V]], not[subclass[s, GAMES]]] == True

In[46]:= (% /. s → s_) /. Equal→ SetDelayed
```

Lemma. (Eliminate the variable **s**.)

```
In[47]:= Map[equal[V, #] &,
          SubstTest[class, s, not[member[s, t]], t -> dif[P[GAMES], fix[GAMES]]]]

Out[47]= subclass[P[GAMES], fix[GAMES]] == True

In[48]:= % /. Equal→ SetDelayed
```

Lemma. (Combine the above inclusion with the reverse inclusion to obtain an equation.)

```
In[49]:= SubstTest[and, subclass[u, v], subclass[v, u], {u -> P[GAMES], v -> fix[GAMES]}]

Out[49]= equal[fix[GAMES], P[GAMES]] == True

In[50]:= % /. Equal→ SetDelayed
```

The above result can now be combined with the statement that **GAMES** is a cartesian square in the form of two rewrite rules.

Corollary. The first of two rewrite rules.

```
In[51]:= SubstTest[implies, and[equal[x, cartsq[y]], equal[y, z]],
           equal[x, cartsq[z]], {x → GAMES, y→ fix[GAMES], z → P[GAMES]}] // Reverse

Out[51]= equal[GAMES, cart[P[GAMES], P[GAMES]]] == True

In[52]:= cart[P[GAMES], P[GAMES]] := GAMES
```

Corollary. The second rewrite rule.

```
In[53]:= SubstTest[fix, cartsq[t], t → P[GAMES]] // Reverse

Out[53]= fix[GAMES] == P[GAMES]

In[54]:= fix[GAMES] := P[GAMES]
```

---

minimality

In this final section transfinite induction is used to show that Conway's class **GAMES** is the least class satisfying **P[x] × P[x] ⊂ x**.

Lemma. The induction step.

```
In[55]:= (SubstTest[implies, and[subclass[u, v], subclass[v, w]],
            subclass[u, w], {u → cartsq[t], v → cartsq[P[x]], w → x}] //
           Reverse) /. t → P[U[image[gamerec, ord[z]]]]

Out[55]= or[not[subclass[cart[P[x], P[x]], x]], not[subclass[U[image[gamerec, ord[z]]], x]],
          subclass[APPLY[gamerec, ord[z]], x]] == True

In[56]:= (% /. {x → x_, z → z_}) /. Equal → SetDelayed
```

Lemma. A technical simplification rule.

```
In[57]:= equal[union[image[inverse[r], P[x]], image[inverse[r],
            P[intersection[x, complement[cart[set[0], set[0]]]]]]], image[inverse[r], P[x]]]

Out[57]= True

In[58]:= union[image[inverse[r_], P[x_]], image[inverse[r_],
            P[intersection[x_, complement[cart[set[0], set[0]]]]]]] := image[inverse[r], P[x]]
```

Observation. In the **GOEDEL** program the theorem about transfinite induction is stated concisely by the following rewrite rule.

```
In[59]:= subclass[intersection[OMEGA, P[t]], t]

Out[59]= subclass[OMEGA, t]
```

In the present case, this theorem is tranformed as follows.

Lemma. (An application of transfinite induction.)

```
In[60]:= SubstTest[subclass, intersection[OMEGA, P[t]],
           t, t -> image[inverse[gamerec], P[x]]] // Reverse

Out[60]= subclass[U[image[gamerec, intersection[OMEGA, P[image[inverse[gamerec], P[x]]]]], x] ==
          subclass[GAMES, x]

In[61]:= subclass[U[image[gamerec, intersection[OMEGA, P[image[inverse[gamerec], P[x_]]]]],
           x_] := subclass[GAMES, x]
```

Main Theorem. The class **GAMES** is the smallest class satisfying **P[x] × P[x] ⊂ x**.

```
In[62]:= Map[equal[V, domain[#]] &, SubstTest[reify, z,
            case[or[not[subclass[w, x]], not[subclass[U[image[gamerec, ord[z]]], x]],
              subclass[APPLY[gamerec, ord[z]], x]]], {w → cartsq[P[x]]}]] // MapNotNot

Out[62]= or[not[subclass[cart[P[x], P[x]], x]], subclass[GAMES, x]] == True

In[63]:= or[not[subclass[cart[P[x_], P[x_]], x_]], subclass[GAMES, x_]] := True
```