recursive game construction, part 2.

Johan G. F. Belinfante
2012 April 25

```
In[1]:= SetDirectory["l:"]; << goedel.12apr25a

        :Package Title: goedel.12apr25a                    2012 April 25 at 2:35 p.m.

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

        It is now:  2012 Apr 25 at 14:38

        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 25 at 14:55
```

summary

This is the second of a series of notebooks in which the theory of well-founded recursion is used to construct the Conway's class **GAMES** of all games. The birthday function **BDAY** for games is introduced. It is shown that **gamerec** is a monotone function, and that its range is a chain of cartesian squares. Since the union of a chain of cartesian squares is a cartesian square, it follows that the class of all games is a cartesian square.

the simplest game

Lemma.

```
In[6]:= SubstTest[cart, P[U[image[gamerec, ord[t]]]], P[U[image[gamerec, ord[t]]], t → 0]
Out[6]= APPLY[gamerec, 0] == cart[set[0], set[0]]

In[7]:= APPLY[gamerec, 0] := cart[set[0], set[0]]
```

Lemma.

```
In[8]:= SubstTest[member, APPLY[gamerec, ord[z]], range[gamerec], z → 0] // Reverse
Out[8]= member[cart[set[0], set[0]], range[gamerec]] == True

In[9]:= member[cart[set[0], set[0]], range[gamerec]] := True
```

Theorem. The simplest game is the ordered pair **(0, 0)**.

```
In[10]:= SubstTest[implies, member[u, v], subclass[u, U[v]],
           {u -> cartsq[set[0]], v -> range[gamerec]}] // Reverse

Out[10]= member[pair[0, 0], GAMES] == True

In[11]:= member[pair[0, 0], GAMES] := True
```

the birthday function

Every game **g** has a birthday, namely the least ordinal β such that **g ∈ APPLY[gamerec, β]**. The **GOEDEL** program can be used to derive an explicit formula for the birthday function as follows:

```
In[12]:= VERTSECT[reify[g, A[class[x, and[member[x, OMEGA], member[g, APPLY[gamerec, x]]]]]]]

Out[12]= VERTSECT[complement[composite[complement[inverse[E]], inverse[gamerec], E]]]
```

This is taken as a definition of the birthday function **BDAY**.

```
In[13]:= VERTSECT[complement[composite[complement[inverse[E]], inverse[gamerec], E]]] := BDAY
```

Theorem. The class **BDAY** is a function.

```
In[14]:= SubstTest[FUNCTION, VERTSECT[t],
           t -> complement[composite[complement[inverse[E]], inverse[gamerec], E]]] // Reverse

Out[14]= FUNCTION[BDAY] == True

In[15]:= FUNCTION[BDAY] := True
```

Theorem. The domain of the birthday function is the class of all games.

```
In[16]:= SubstTest[domain,
           VERTSECT[complement[composite[complement[inverse[E]], inverse[funpart[t]], E]]],
           t → gamerec] // Reverse

Out[16]= domain[BDAY] == GAMES

In[17]:= domain[BDAY] := GAMES
```

Theorem.

```
In[18]:= Map[member[#, OMEGA] &, SubstTest[APPLY, VERTSECT[t], g,
            t -> complement[composite[complement[inverse[E]], inverse[gamerec], E]]]] // Reverse

Out[18]= member[APPLY[BDAY, g], OMEGA] == member[g, GAMES]

In[19]:= member[APPLY[BDAY, g_], OMEGA] := member[g, GAMES]
```

Lemma.

```
In[20]:= Map[domain[reify[g, #]] &,
           SubstTest[case, member[APPLY[funpart[t], g], OMEGA], t → BDAY]]

Out[20]= image[inverse[BDAY], OMEGA] == GAMES

In[21]:= image[inverse[BDAY], OMEGA] := GAMES
```

Theorem.

```
In[22]:= Map[subclass[#, OMEGA] &, ImageComp[BDAY, inverse[BDAY], OMEGA]] // Reverse

Out[22]= subclass[range[BDAY], OMEGA] == True

In[23]:= subclass[range[BDAY], OMEGA] := True
```

Theorem. Vertical section rule for the birthday function.

```
In[24]:= SubstTest[image, funpart[t], set[g], t → BDAY] // Reverse

Out[24]= image[BDAY, set[g]] == set[APPLY[BDAY, g]]

In[25]:= image[BDAY, set[g_]] := set[APPLY[BDAY, g]]
```

Since it will soon become evident that games are ordered pairs, the following variant is generally more useful.

Corollary. Another vertical section rule.

```
In[26]:= SubstTest[image, funpart[u], set[v], {u → BDAY, v → PAIR[x, y]}] // Reverse

Out[26]= image[BDAY, cart[set[x], set[y]]] == set[APPLY[BDAY, PAIR[x, y]]]

In[27]:= image[BDAY, cart[set[x_], set[y_]]] := set[APPLY[BDAY, PAIR[x, y]]]
```

Lemma.

```
In[28]:= member[0, image[inverse[gamerec], P[complement[cart[set[0], set[0]]]]]] // AssertTest

Out[28]= member[0, image[inverse[gamerec], P[complement[cart[set[0], set[0]]]]]] == False

In[29]:= % /. Equal→ SetDelayed
```

Theorem. The birthday of the simplest game is **0**.

```
In[30]:= SubstTest[APPLY, VERTSECT[t], pair[0, 0],
           t -> complement[composite[complement[inverse[E]], inverse[gamerec], E]]] // Reverse

Out[30]= APPLY[BDAY, pair[0, 0]] == 0

In[31]:= APPLY[BDAY, pair[0, 0]] := 0
```

monotonicity of gamerec

Lemma.

```
In[32]:= SubstTest[implies, subclass[u, v],
           subclass[image[z, u], image[z, v]], {u → ord[x], v -> ord[y]}] // Reverse

Out[32]= or[member[ord[y], ord[x]], subclass[image[z, ord[x]], image[z, ord[y]]]] == True

In[33]:= or[member[ord[y_], ord[x_]], subclass[image[z_, ord[x_]], image[z_, ord[y_]]]] := True
```

Lemma.

```
In[34]:= SubstTest[subclass, cartsq[s], cartsq[t],
          {s → P[U[image[gamerec, ord[u]]]], t → P[U[image[gamerec, ord[v]]]]}]

Out[34]= subclass[U[image[gamerec, ord[u]]], U[image[gamerec, ord[v]]]] ==
          subclass[APPLY[gamerec, ord[u]], APPLY[gamerec, ord[v]]]

In[35]:= subclass[U[image[gamerec, ord[u_]]], U[image[gamerec, ord[v_]]]] :=
          subclass[APPLY[gamerec, ord[u]], APPLY[gamerec, ord[v]]]
```

Theorem.

```
In[36]:= SubstTest[or, member[ord[y], ord[x]], subclass[image[z, ord[x]], image[z, ord[y]]],
           z → composite[inverse[E], gamerec]] // Reverse

Out[36]= or[member[ord[y], ord[x]],
           subclass[APPLY[gamerec, ord[x]], APPLY[gamerec, ord[y]]]] == True

In[37]:= or[member[ord[y_], ord[x_]],
           subclass[APPLY[gamerec, ord[x_]], APPLY[gamerec, ord[y_]]]] := True
```

Corollary. (Removing the **ord** wrappers.)

```
In[38]:= SubstTest[implies, and[equal[u, ord[x]], equal[v, ord[y]]], implies[subclass[u, v],
            subclass[APPLY[gamerec, u], APPLY[gamerec, v]]], {x → u, y → v}] // Reverse

Out[38]= or[not[member[u, OMEGA]], not[member[v, OMEGA]],
           not[subclass[u, v]], subclass[APPLY[gamerec, u], APPLY[gamerec, v]]] == True

In[39]:= (% /. {u → u_, v → v_}) /. Equal → SetDelayed
```

Lemma. (Eliminating the variables **u** and **v**.)

```
In[40]:= Map[composite[Id, complement[#]] &, SubstTest[class, pair[u, v], implies[
            and[member[u, domain[funpart[t]]], member[v, domain[funpart[t]]], subclass[u, v]],
            subclass[APPLY[funpart[t], u], APPLY[funpart[t], v]]], t → gamerec]]

Out[40]= composite[id[OMEGA],
           intersection[S, composite[inverse[gamerec], complement[S], gamerec]], id[OMEGA]] == 0
```

```
In[41]:= % /. Equal → SetDelayed
```

Lemma.

```
In[42]:= SubstTest[empty, composite[id[OMEGA], dif[u, v], id[OMEGA]],
          {u → S, v → composite[inverse[gamerec], S, gamerec]}]

Out[42]= subclass[composite[id[OMEGA], S, id[OMEGA]],
           composite[inverse[gamerec], S, gamerec]] == True

In[43]:= % /. Equal → SetDelayed
```

Theorem. Monotonicity of **gamerec**.

```
In[44]:= SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]],
           {t → cross[gamerec, gamerec], u -> composite[id[OMEGA], S, id[OMEGA]],
            v -> composite[inverse[gamerec], S, gamerec]}] // Reverse

Out[44]= subclass[composite[gamerec, S, inverse[gamerec]], S] == True

In[45]:= subclass[composite[gamerec, S, inverse[gamerec]], S] := True
```

Corollary. The range of **gamerec** is a chain.

```
In[46]:= SubstTest[implies,
           and[subclass[P[domain[t]], chains[S]], subclass[P[t], monotone[S, S]]],
           subclass[P[range[t]], chains[S]], t → gamerec] // Reverse

Out[46]= subclass[cart[range[gamerec], range[gamerec]], union[S, inverse[S]]] == True

In[47]:= subclass[cart[range[gamerec], range[gamerec]], union[S, inverse[S]]] := True
```

The union of a chain of cartesian squares is a cartesian square.

Theorem. The class of all games is a cartesian square.

```
In[48]:= SubstTest[implies, and[subclass[x, image[CART, Id]], subclass[P[x], chains[S]]],
           equal[U[x], cartsq[fix[U[x]]]], x → range[gamerec]] // Reverse

Out[48]= equal[GAMES, cart[fix[GAMES], fix[GAMES]]] == True

In[49]:= equal[GAMES, cart[fix[GAMES], fix[GAMES]]] := True
```