a uniqueness theorem for id[GAMES]

Johan G. F. Belinfante
2012 May 19

In[1]:= **SetDirectory["l:"]; << goedel.12may16a**

```
:Package Title: goedel.12may16a                    2012 May 16 at 10:30 a.m.

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

It is now:  2012 May 19 at 11:59

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 May 19 at 12:15
```

summary

The class **GAMES** of Conway games is constructed recursively, each game being an ordered pair of sets of previously constructed games, called the **options** for the players "left" and "right". A game **x** is a **prelude** to a game **y** if **x** is one of the left or right options of the game **y**. Formally the relation **PRELUDE** is defined by the following rewrite rule (equation):

In[2]:= **union[composite[id[GAMES], inverse[FIRST], E], composite[id[GAMES], inverse[SECOND], E]]**

Out[2]= PRELUDE

The relation **PRELUDE** is well-founded and its inverse is thin, that is, every vertical section of **PRELUDE** is a set. The relation **PRELUDE** is used in the theory of Conway games both to provide recursive definitions of important theorems, and to provide a convenient way to prove theorems about games inductively. In this notebook, the basic theorems for proofs by prelude induction are derived, and illustrated on the proof of a basic theorem that which characterizes the identity function **id[GAMES]** which takes any game to itself as the unique solution of a certain recursion equation.

prelude-induction

The relation **PRELUDE** is well-founded and its inverse is thin. On account of this, the following form of well-founded induction holds:

Theorem. A basic form of prelude induction.

```
In[3]:= SubstTest[implies, and[WELLFOUNDED[w], thin[inverse[w]], subvariant[w, x]],
          empty[x], w → PRELUDE] // Reverse

Out[3]= or[equal[0, x], not[subclass[x, image[PRELUDE, x]]]] == True

In[4]:= (% /. x → x_) /. Equal → SetDelayed
```

Corollary. (A better rewrite rule for prelude induction.)

```
In[5]:= equiv[subclass[x, image[PRELUDE, x]], equal[0, x]]

Out[5]= True

In[6]:= subclass[x_, image[PRELUDE, x_]] := equal[0, x]
```

Many proofs using prelude induction operate by showing that there can be no least counterexample. The following theorem provides this version of prelude induction. It will be used in the proof of theorem characterizing **id[GAMES]**.

Theorem. A "no least counterexample" form of prelude induction that will be used for an application provided below.

```
In[7]:= SubstTest[subclass, t, image[PRELUDE, t], t → dif[GAMES, x]] // Reverse

Out[7]= subclass[GAMES, union[x, image[PRELUDE, complement[x]]]] == subclass[GAMES, x]

In[8]:= subclass[GAMES, union[x_, image[PRELUDE, complement[x_]]]] := subclass[GAMES, x]
```

---

a recursion equation satisfied by id[GAMES].

Theorem. A recursion equation for **id[GAMES]**.

```
In[9]:= SubstTest[composite, cross[IMAGE[id[GAMES]], IMAGE[id[GAMES]]],
          id[cart[P[t], P[t]]], t → GAMES] // Reverse

Out[9]= composite[cross[IMAGE[id[GAMES]], IMAGE[id[GAMES]]], id[GAMES]] == id[GAMES]

In[10]:= composite[cross[IMAGE[id[GAMES]], IMAGE[id[GAMES]]], id[GAMES]] := id[GAMES]
```

It will be shown later that **w = id[GAMES]** is the only solution of the recursion equation **w = (IMAGE[w] ⊗ IMAGE[w]) ∘ id[GAMES]**. In the present section some elementary consequences of the recursion equation are derived. In the first place, **w** must be a function. The **GOEDEL** program recognizes this automatically using the built-in rewrite rules for equality; no proof is needed.

```
In[11]:= implies[equal[w, composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]]], FUNCTION[w]]

Out[11]= True
```

It is often useful in deriving consequences of the recursion equation to begin by adding a **funpart** wrapper, and then removing this wrapper. An example of this is the following.

Lemma. (This contains the redundant literal **FUNCTION[w]**.)

```
In[12]:= SubstTest[implies,
           and[equal[w, funpart[t]], equal[w, composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]]]],
           equal[domain[w], GAMES], t → w] // Reverse

Out[12]= or[equal[GAMES, domain[w]],
           not[equal[w, composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]]]],
           not[FUNCTION[w]]] == True

In[13]:= (% /. w → w_) /. Equal → SetDelayed
```

Theorem. The recursion equation implies **domain[w] = GAMES**.

```
In[14]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3], not[implies[p1, p3]],
            {p1 -> equal[w, composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]]],
             p2 → FUNCTION[w], p3 → equal[domain[w], GAMES]}]] // Reverse

Out[14]= or[equal[GAMES, domain[w]],
           not[equal[w, composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]]]]] == True

In[15]:= or[equal[GAMES, domain[w_]],
           not[equal[w_, composite[cross[IMAGE[w_], IMAGE[w_]], id[GAMES]]]]] := True
```

Lemma. A recursion equation involving **APPLY**.

```
In[16]:= SubstTest[implies, equal[w, t], equal[APPLY[w, z], APPLY[t, z]],
           {t -> composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]], z → game[x]}] // Reverse

Out[16]= or[equal[APPLY[w, game[x]], PAIR[image[w, first[game[x]]], image[w, second[game[x]]]]],
           not[equal[w, composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]]]]] == True

In[17]:= or[equal[APPLY[w_, game[x_]],
            PAIR[image[w_, first[game[x_]]], image[w_, second[game[x_]]]]],
           not[equal[w_, composite[cross[IMAGE[w_], IMAGE[w_]], id[GAMES]]]]] := True
```

simplification rules for game[x]

It will be useful to introduce the wrapper **game[x]** in subsequent work.

Lemma. The vertical section of **inverse[PRELUDE]** at **games[x]**.

```
In[18]:= SubstTest[APPLY, VERTSECT[t], game[x], t → inverse[PRELUDE]]

Out[18]= image[inverse[PRELUDE], set[game[x]]] == union[first[game[x]], second[game[x]]]

In[19]:= image[inverse[PRELUDE], set[game[x_]]] := union[first[game[x]], second[game[x]]]
```

Lemma. Simplification rule.

```
In[20]:= equal[intersection[GAMES, first[game[x]]], first[game[x]]]

Out[20]= True
```

```
In[21]:= intersection[GAMES, first[game[x_]]] := first[game[x]]
```

Lemma. Dual simplification rule.

```
In[22]:= equal[intersection[GAMES, second[game[x]]], second[game[x]]]

Out[22]= True
```

```
In[23]:= intersection[GAMES, second[game[x_]]] := second[game[x]]
```

Lemma.

```
In[24]:= equal[PAIR[first[game[x]], second[game[x]]], game[x]]

Out[24]= True
```

```
In[25]:= PAIR[first[game[x_]], second[game[x_]]] := game[x]
```

Lemma.

```
In[26]:= SubstTest[set, PAIR[first[t], second[t]], t → game[x]]

Out[26]= cart[set[first[game[x]]], set[second[game[x]]]] == set[game[x]]
```

```
In[27]:= cart[set[first[game[x_]]], set[second[game[x_]]]] := set[game[x]]
```

---

derivation of the main theorem

The variable **w** will be wrapped with **funpart** for most of the derivation.

Lemma.

```
In[28]:= Map[implies[subclass[union[first[game[x]], second[game[x]]], fix[funpart[w]]], #] &,
            SubstTest[member, PAIR[u, v], cart[set[first[t]], set[second[t]]],
             {u -> image[funpart[w], first[game[x]]],
              v -> image[funpart[w], second[game[x]]], t → game[x]}]] // MapNotNot // Reverse

Out[28]= or[equal[game[x],
            pair[image[funpart[w], first[game[x]]], image[funpart[w], second[game[x]]]]],
          not[subclass[first[game[x]], fix[funpart[w]]]],
          not[subclass[second[game[x]], fix[funpart[w]]]]] == True
```

```
In[29]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

Theorem. If every prelude of **game[x]** is a fixed point of a solution **funpart[w]** of the recursion equation, then so is **game[x]** itself.

```
In[30]:= Map[not, SubstTest[and, implies[p0, p4], implies[p1, p2],
            implies[p2, p3], implies[and[p3, p4], p5], not[implies[and[p0, p1], p5]],
            {p0 → and[subclass[first[game[x]], fix[funpart[w]]],
               subclass[second[game[x]], fix[funpart[w]]]],
             p1 → equal[composite[cross[IMAGE[funpart[w]], IMAGE[funpart[w]]], id[GAMES]],
               funpart[w]], p2 → equal[APPLY[funpart[w], game[x]],
               PAIR[image[funpart[w], first[game[x]]], image[funpart[w], second[game[x]]]]],
             p3 -> equal[APPLY[funpart[w], game[x]], pair[image[funpart[w], first[game[x]]],
                image[funpart[w], second[game[x]]]]], p4 -> equal[game[x],
               pair[image[funpart[w], first[game[x]]], image[funpart[w], second[game[x]]]]],
             p5 → equal[APPLY[funpart[w], game[x]], game[x]]}]] // Reverse

Out[30]= or[equal[APPLY[funpart[w], game[x]], game[x]],
           not[equal[composite[cross[IMAGE[funpart[w]], IMAGE[funpart[w]]], id[GAMES]],
             funpart[w]]], not[subclass[first[game[x]], fix[funpart[w]]]],
           not[subclass[second[game[x]], fix[funpart[w]]]]] == True

In[31]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

Corollary. (Eliminating **APPLY** from the statement in the theorem.)

```
In[32]:= Map[not, SubstTest[and, implies[and[p0, p1], p3], implies[p1, p2],
            implies[p2, p4], implies[and[p3, p4], p5], not[implies[and[p0, p1], p5]],
            {p0 → and[subclass[first[game[x]], fix[funpart[w]]],
               subclass[second[game[x]], fix[funpart[w]]]],
             p1 → equal[composite[cross[IMAGE[funpart[w]], IMAGE[funpart[w]]], id[GAMES]],
               funpart[w]], p2 → equal[domain[funpart[w]], GAMES],
             p3 -> equal[APPLY[funpart[w], game[x]], game[x]],
             p4 → member[game[x], domain[funpart[w]]],
             p5 → member[game[x], fix[funpart[w]]]}]] // Reverse

Out[32]= or[member[game[x], fix[funpart[w]]],
           not[equal[composite[cross[IMAGE[funpart[w]], IMAGE[funpart[w]]], id[GAMES]],
             funpart[w]]], not[subclass[first[game[x]], fix[funpart[w]]]],
           not[subclass[second[game[x]], fix[funpart[w]]]]] == True

In[33]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

Theorem. (Eliminate the **funpart** wrapper.)

```
In[34]:= SubstTest[implies, equal[w, funpart[t]], or[member[game[x], fix[w]],
            not[equal[w, composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]]]],
            not[subclass[first[game[x]], fix[w]]],
            not[subclass[second[game[x]], fix[w]]]], t → w] // Reverse

Out[34]= or[member[game[x], fix[w]],
           not[equal[w, composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]]]], not[FUNCTION[w]],
           not[subclass[first[game[x]], fix[w]]], not[subclass[second[game[x]], fix[w]]]] == True

In[35]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

Theorem. (Eliminate the redundant **FUNCTION[w]** literal.)

```
In[36]:= Map[not, SubstTest[and, implies[p1, p3],
            implies[and[p1, p2, p3], p4], not[implies[and[p1, p2], p4]],
            {p1 → equal[w, composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]]],
             p2 → and[subclass[first[game[x]], fix[w]], subclass[second[game[x]], fix[w]]],
             p3 → FUNCTION[w], p4 → member[game[x], fix[w]]}]] // Reverse

Out[36]= or[member[game[x], fix[w]],
           not[equal[w, composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]]]],
           not[subclass[first[game[x]], fix[w]]], not[subclass[second[game[x]], fix[w]]]] == True

In[37]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

The elimination of the **game** wrapper in the following theorem is taken as an opportunity to formally introduce the **PRELUDE** relation.

Theorem.

```
In[38]:= SubstTest[implies,
           and[equal[x, game[t]], equal[w, composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]]],
            subclass[image[inverse[PRELUDE], set[x]], fix[w]]],
           member[x, fix[w]], t → x] // Reverse

Out[38]= or[member[x, fix[w]], not[equal[w, composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]]]],
           not[member[x, GAMES]], not[subclass[image[inverse[PRELUDE], set[x]], fix[w]]]] == True

In[39]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

The variable **x** here will be eliminated using **reify** and **case**, and the no-least-counterexample form of prelude induction automatically simplifies the result.

Theorem. If **w** is a solution of the recursion equation, then **GAMES ⊂ fix[w]**.

```
In[40]:= Map[equal[V, domain[#]] &,
          SubstTest[reify, x, case[or[member[x, fix[w]], not[equal[w, t]],
             not[member[x, GAMES]], not[subclass[image[inverse[PRELUDE], set[x]], fix[w]]]]],
           t -> composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]]]]

Out[40]= or[not[equal[w, composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]]]],
           subclass[GAMES, fix[w]]] == True

In[41]:= (% /. w → w_) /. Equal → SetDelayed
```

Identity functions can be characterized as functions **w** that satisfy **domain[w] ⊂ fix[w]**.

Lemma.

```
In[43]:= SubstTest[implies, equal[w, funpart[t]], or[equal[w, id[x]],
            not[equal[x, domain[w]]], not[subclass[x, fix[w]]]], t → w] // Reverse

Out[43]= or[equal[w, id[x]], not[equal[x, domain[w]]],
           not[FUNCTION[w]], not[subclass[x, fix[w]]]] == True

In[44]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

Main Theorem. The only solution of the recursion equation **w = (IMAGE[w] ⊗ IMAGE[w]) ∘ id[GAMES]** is **w = id[GAMES]**.

```
In[45]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],
            implies[p1, p4], implies[and[p2, p3, p4], p5], not[implies[p1, p5]],
            {p1 -> equal[w, composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]]],
             p2 → FUNCTION[w], p3 → equal[domain[w], GAMES],
             p4 → subclass[GAMES, fix[w]], p5 → equal[w, id[GAMES]]}]] // Reverse

Out[45]= or[equal[w, id[GAMES]],
           not[equal[w, composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]]]]] == True

In[46]:= (% /. w → w_) /. Equal → SetDelayed
```

Corollary. A better rewrite rule.

```
In[47]:= equiv[equal[w, composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]]], equal[w, id[GAMES]]]

Out[47]= True

In[49]:= equal[w_, composite[cross[IMAGE[w_], IMAGE[w_]], id[GAMES]]] := equal[w, id[GAMES]]
```