role reversal in a game

Johan G. F. Belinfante
2012 May 20

```
In[1]:= SetDirectory["l:"]; << goedel.12may19a
```

```
:Package Title: goedel.12may19a                    2012 May 19 at 2:30 p.m.

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

It is now:  2012 May 20 at 16:55

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 20 at 17:12
```

summary

The relation **PRELUDE** is used to recursively define the function **REVGAME** which turns the tables for any game, reversing the roles of the two players of a game. It is shown that **REVGAME** is an involution on the class of games.

prelude-induction

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

```
In[2]:= subvariant[PRELUDE, x]
Out[2]= equal[0, x]
```

Another form of prelude induction used below is this no-first-counterexample form:

```
subclass[GAMES, union[x, image[PRELUDE, intersection[GAMES, complement[x]]]]]

subclass[GAMES, x]
```

definition of REVGAME

A temporary abbreviation.

```
In[3]:= standard := {x → union[cart[cart[complement[cart[V, V]], V], set[0]],
            composite[SWAP, cross[composite[IMG, SWAP], composite[IMG, SWAP]],
             TWIST, cross[Id, DUP]]], y → inverse[PRELUDE]}
```

Definition. The function **REVGAME** is defined as the restriction of **rec[x, y]** under **standard** to the class **GAMES**.

```
In[4]:= composite[rec[union[cart[cart[complement[cart[V, V]], V], set[0]],
            composite[SWAP, cross[composite[IMG, SWAP], composite[IMG, SWAP]],
             TWIST, cross[Id, DUP]]], inverse[PRELUDE]], id[GAMES]] := REVGAME
```

REVGAME is a function

Lemma.

```
In[5]:= SubstTest[implies, and[FUNCTION[x], FUNCTION[y], disjoint[domain[x], domain[y]]],
          FUNCTION[union[x, y]], {x -> cart[cart[complement[cart[V, V]], V], set[0]],
           y -> composite[SWAP, cross[composite[IMG, SWAP], composite[IMG, SWAP]],
            TWIST, cross[Id, DUP]]}] // Reverse

Out[5]= FUNCTION[union[cart[cart[complement[cart[V, V]], V], set[0]], composite[SWAP,
            cross[composite[IMG, SWAP], composite[IMG, SWAP]], TWIST, cross[Id, DUP]]]] == True

In[6]:= % /. Equal→ SetDelayed
```

Lemma. The class **rec[x,y]** under the **standard** substitution is a function.

```
In[7]:= SubstTest[implies, and[FUNCTION[x], equal[domain[x], cart[V, V]],
           WELLFOUNDED[inverse[y]], thin[y]], FUNCTION[rec[x, y]], standard] // Reverse

Out[7]= FUNCTION[rec[union[cart[cart[complement[cart[V, V]], V], set[0]],
            composite[SWAP, cross[composite[IMG, SWAP], composite[IMG, SWAP]],
             TWIST, cross[Id, DUP]]], inverse[PRELUDE]]] == True

In[8]:= % /. Equal→ SetDelayed
```

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

```
In[9]:= SubstTest[FUNCTION, composite[funpart[t], id[GAMES]],
          t → (rec[x, y] /. standard)] // Reverse

Out[9]= FUNCTION[REVGAME] == True

In[10]:= FUNCTION[REVGAME] := True
```

Lemma.

```
In[11]:= SubstTest[implies, and[FUNCTION[x], equal[domain[x], cart[V, V]],
           WELLFOUNDED[inverse[y]], thin[y]], equal[V, domain[rec[x, y]]], standard] // Reverse

Out[11]= equal[V, domain[rec[union[cart[cart[complement[cart[V, V]], V], set[0]],
           composite[SWAP, cross[composite[IMG, SWAP], composite[IMG, SWAP]],
            TWIST, cross[Id, DUP]]], inverse[PRELUDE]]]] == True

In[12]:= domain[rec[union[cart[cart[complement[cart[V, V]], V], set[0]],
           composite[SWAP, cross[composite[IMG, SWAP], composite[IMG, SWAP]],
            TWIST, cross[Id, DUP]]], inverse[PRELUDE]]] := V
```

Theorem. The domain of **REVGAME**.

```
In[13]:= IminComp[(rec[x, y] /. standard), id[GAMES], V]

Out[13]= domain[REVGAME] == GAMES

In[14]:= domain[REVGAME] := GAMES
```

a recursion equation for REVGAME

Lemma.

```
In[15]:= ApComp[(rec[x, y] /. standard), id[GAMES], game[x]]

Out[15]= APPLY[rec[union[cart[cart[complement[cart[V, V]], V], set[0]], composite[SWAP,
           cross[composite[IMG, SWAP], composite[IMG, SWAP]], TWIST, cross[Id, DUP]]],
         inverse[PRELUDE]], game[x]] == APPLY[REVGAME, game[x]]

In[16]:= APPLY[rec[union[cart[cart[complement[cart[V, V]], V], set[0]], composite[SWAP,
           cross[composite[IMG, SWAP], composite[IMG, SWAP]], TWIST, cross[Id, DUP]]],
         inverse[PRELUDE]], game[x_]] := APPLY[REVGAME, game[x]]
```

Lemma.

```
In[17]:= ImageComp[(rec[x, y] /. standard), id[GAMES], z] // Reverse

Out[17]= image[rec[union[cart[cart[complement[cart[V, V]], V], set[0]], composite[SWAP,
           cross[composite[IMG, SWAP], composite[IMG, SWAP]], TWIST, cross[Id, DUP]]],
         inverse[PRELUDE]], intersection[GAMES, z]] == image[REVGAME, z]

In[18]:= image[rec[union[cart[cart[complement[cart[V, V]], V], set[0]], composite[SWAP,
           cross[composite[IMG, SWAP], composite[IMG, SWAP]], TWIST, cross[Id, DUP]]],
         inverse[PRELUDE]], intersection[GAMES, z_]] := image[REVGAME, z]
```

Lemma.

```
In[19]:= ImageComp[(rec[x, y] /. standard), id[GAMES], first[game[x]]] // Reverse

Out[19]= image[rec[union[cart[cart[complement[cart[V, V]], V], set[0]], composite[SWAP,
           cross[composite[IMG, SWAP], composite[IMG, SWAP]], TWIST, cross[Id, DUP]]],
         inverse[PRELUDE]], first[game[x]]] == image[REVGAME, first[game[x]]]

In[20]:= image[rec[union[cart[cart[complement[cart[V, V]], V], set[0]], composite[SWAP,
           cross[composite[IMG, SWAP], composite[IMG, SWAP]], TWIST, cross[Id, DUP]]],
         inverse[PRELUDE]], first[game[x_]]] := image[REVGAME, first[game[x]]]
```

Lemma.

```
In[21]:= ImageComp[(rec[x, y] /. standard), id[GAMES], second[game[x]]] // Reverse

Out[21]= image[rec[union[cart[cart[complement[cart[V, V]], V], set[0]], composite[SWAP,
           cross[composite[IMG, SWAP], composite[IMG, SWAP]], TWIST, cross[Id, DUP]]],
         inverse[PRELUDE]], second[game[x]]] == image[REVGAME, second[game[x]]]

In[22]:= image[rec[union[cart[cart[complement[cart[V, V]], V], set[0]], composite[SWAP,
           cross[composite[IMG, SWAP], composite[IMG, SWAP]], TWIST, cross[Id, DUP]]],
         inverse[PRELUDE]], second[game[x_]]] := image[REVGAME, second[game[x]]]
```

It is not clear how best to orient the following rewrite rule. The choice taken here is tentative.

Theorem. An **APPLY** rule for **REVGAME**.

```
In[23]:= (SubstTest[or, equal[APPLY[funpart[x], PAIR[z, composite[rec[funpart[x], thinpart[y]],
             id[image[thinpart[y], set[z]]]]]], APPLY[rec[funpart[x], thinpart[y]], z]],
           not[member[z, domain[rec[funpart[x], thinpart[y]]]]],
           not[WELLFOUNDED[inverse[y]]], standard] // Reverse) /. z → game[x]

Out[23]= equal[APPLY[REVGAME, game[x]],
           PAIR[image[REVGAME, second[game[x]]], image[REVGAME, first[game[x]]]]] == True

In[24]:= PAIR[image[REVGAME, second[game[x_]]], image[REVGAME, first[game[x_]]]] :=
          APPLY[REVGAME, game[x]]
```

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

```
In[25]:= SubstTest[implies, equal[x, game[t]], equal[APPLY[REVGAME, x],
           PAIR[image[REVGAME, second[x]], image[REVGAME, first[x]]]], t → x] // Reverse

Out[25]= or[equal[APPLY[REVGAME, x], PAIR[image[REVGAME, second[x]], image[REVGAME, first[x]]]],
           not[member[x, GAMES]]] == True

In[26]:= or[equal[APPLY[REVGAME, x_], PAIR[image[REVGAME, second[x_]],
            image[REVGAME, first[x_]]]], not[member[x_, GAMES]]] := True
```

Lemma.

```
In[27]:= or[and[
           equal[APPLY[REVGAME, x], PAIR[image[REVGAME, second[x]], image[REVGAME, first[x]]]],
           member[first[x], V]], not[member[x, GAMES]]] // NotNotTest

Out[27]= or[and[
           equal[APPLY[REVGAME, x], PAIR[image[REVGAME, second[x]], image[REVGAME, first[x]]]],
           member[first[x], V]], not[member[x, GAMES]]] == True

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

Theorem. Eliminate the variable **x**.

```
In[29]:= Map[equal[V, domain[#]] &, SubstTest[reify, x,
           case[implies[member[x, GAMES], equal[APPLY[funpart[u], x], APPLY[funpart[v], x]]],
           {u → composite[SWAP, cross[IMAGE[REVGAME], IMAGE[REVGAME]]], v → REVGAME}]]

Out[29]= subclass[GAMES, fix[composite[SWAP,
             cross[inverse[IMAGE[REVGAME]], inverse[IMAGE[REVGAME]]], REVGAME]]] == True

In[30]:= % /. Equal → SetDelayed
```

Lemma.

```
In[31]:= SubstTest[subclass, domain[funpart[x]],
          fix[composite[inverse[funpart[y]], funpart[x]]],
          {x → REVGAME, y -> composite[SWAP, cross[IMAGE[REVGAME], IMAGE[REVGAME]]]}]

Out[31]= subclass[REVGAME, composite[SWAP, cross[IMAGE[REVGAME], IMAGE[REVGAME]]]] == True

In[32]:= subclass[REVGAME, composite[SWAP, cross[IMAGE[REVGAME], IMAGE[REVGAME]]]] := True
```

Main Theorem. Recursion relation for **REVGAME**.

```
In[33]:= SubstTest[equal, funpart[x], composite[funpart[y], id[domain[funpart[x]]]],
           {x -> REVGAME, y -> composite[SWAP, cross[IMAGE[REVGAME], IMAGE[REVGAME]]]}] // Reverse

Out[33]= equal[REVGAME,
           composite[SWAP, cross[IMAGE[REVGAME], IMAGE[REVGAME]], id[GAMES]]] == True

In[34]:= composite[SWAP, cross[IMAGE[REVGAME], IMAGE[REVGAME]], id[GAMES]] := REVGAME
```

range[REVGAME]

It is not immediately obvious from the definition of game-reveral that the reverse of a game is a game. Here we will use prelude-induction to establish this.

Lemma. Inverse image rule.

```
In[35]:= (member[x, image[inverse[funpart[t]], y]] // AssertTest) /. t → REVGAME

Out[35]= member[x, image[inverse[REVGAME], y]] == member[APPLY[REVGAME, x], y]
```

```
In[36]:= member[x_, image[inverse[REVGAME], y_]] := member[APPLY[REVGAME, x], y]
```

Theorem. A simplification rule.

```
In[37]:= SubstTest[member, PAIR[u, v], cartsq[P[w]],
          {u -> image[REVGAME, second[game[x]]], v -> image[REVGAME, first[game[x]]], w → GAMES}]

Out[37]= and[subclass[image[REVGAME, first[game[x]]], GAMES],
           subclass[image[REVGAME, second[game[x]]], GAMES]] ==
          member[APPLY[REVGAME, game[x]], GAMES]

In[38]:= and[subclass[image[REVGAME, first[game[x_]]], GAMES],
           subclass[image[REVGAME, second[game[x_]]], GAMES]] :=
          member[APPLY[REVGAME, game[x]], GAMES]
```

The removal of the **game** wrapper provides an idea occasion to bring the **PRELUDE** relation into play.

Theorem. (The **game** wrapper is eliminated.)

```
In[39]:= SubstTest[implies, equal[x, game[t]],
           implies[subclass[image[composite[REVGAME, inverse[PRELUDE]], set[x]], GAMES],
            member[x, image[inverse[REVGAME], GAMES]]], t → x] // Reverse

Out[39]= or[member[APPLY[REVGAME, x], GAMES], not[member[x, GAMES]],
           not[subclass[image[REVGAME, image[inverse[PRELUDE], set[x]]], GAMES]]] == True

In[40]:= (% /. x → x_) /. Equal→ SetDelayed
```

In the course of eliminating the variable x the "no-first-counterexample" form of prelude induction automatically simplifies the result.

Lemma. (Eliminating the variable **x**.)

```
In[41]:= Map[equal[V, domain[#]] &,
          SubstTest[reify, x, case[or[member[APPLY[r, x], GAMES], not[member[x, GAMES]],
             not[subclass[image[r, image[inverse[PRELUDE], set[x]]], GAMES]]]], r → REVGAME]]

Out[41]= subclass[range[REVGAME], GAMES] == True

In[42]:= % /. Equal→ SetDelayed
```

Lemma. Temporary simplification rule.

```
In[43]:= equal[intersection[GAMES, range[REVGAME]], range[REVGAME]]

Out[43]= True

In[44]:= intersection[GAMES, range[REVGAME]] := range[REVGAME]
```

Lemma. Simplification rule.

```
In[45]:= Assoc[id[GAMES], id[range[REVGAME]], REVGAME]
```

```
Out[45]= composite[id[GAMES], REVGAME] == REVGAME
```

```
In[46]:= composite[id[GAMES], REVGAME] := REVGAME
```

Corollary.

```
In[47]:= Assoc[composite[SWAP, cross[IMAGE[REVGAME], IMAGE[REVGAME]]], id[GAMES], REVGAME]
```

```
Out[47]= composite[SWAP, cross[IMAGE[REVGAME], IMAGE[REVGAME]], REVGAME] ==
          composite[REVGAME, REVGAME]
```

```
In[48]:= composite[SWAP, cross[IMAGE[REVGAME], IMAGE[REVGAME]], REVGAME] :=
          composite[REVGAME, REVGAME]
```

Corollary.

```
In[49]:= Assoc[composite[SWAP, cross[IMAGE[REVGAME], IMAGE[REVGAME]]], SWAP,
           composite[cross[IMAGE[REVGAME], IMAGE[REVGAME]], id[GAMES]]] // Reverse
```

```
Out[49]= composite[cross[composite[IMAGE[REVGAME], IMAGE[REVGAME]],
            composite[IMAGE[REVGAME], IMAGE[REVGAME]]], id[GAMES]] == composite[REVGAME, REVGAME]
```

```
In[50]:= % /. Equal → SetDelayed
```

Theorem.

```
In[51]:= SubstTest[equal, w, composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]],
          w -> composite[REVGAME, REVGAME]]
```

```
Out[51]= equal[composite[REVGAME, REVGAME], id[GAMES]] == True
```

```
In[52]:= composite[REVGAME, REVGAME] := id[GAMES]
```

Corollary.

```
In[53]:= Map[subclass[#, range[REVGAME]] &, ImageComp[REVGAME, REVGAME, V]]
```

```
Out[53]= subclass[GAMES, range[REVGAME]] == True
```

```
In[54]:= % /. Equal → SetDelayed
```

Theorem.

```
In[55]:= SubstTest[and, subclass[u, v], subclass[v, u], {u -> range[REVGAME], v → GAMES}]
```

```
Out[55]= equal[GAMES, range[REVGAME]] == True
```

```
In[56]:= range[REVGAME] := GAMES
```

Corollary. The function **REVGAME** is an involution.

In[57]:= **Assoc[REVGAME, REVGAME, inverse[REVGAME]] // Reverse**

Out[57]= inverse[REVGAME] == REVGAME

In[58]:= **inverse[REVGAME] := REVGAME**