a uniqueness theorem for \text{id}[\text{GAMES}]

Johan G. F. Belinfante
2012 May 19

\textit{In[1]} := \texttt{SetDirectory["l:"]; \textless \textless \texttt{goedel.12may16a}}

\begin{verbatim}
: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
\end{verbatim}

\textbf{summary}

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

\begin{verbatim}
\texttt{In[2]} := \texttt{union[composite[id[\texttt{GAMES}], inverse[\texttt{FIRST}], \texttt{E}], composite[id[\texttt{GAMES}], inverse[\texttt{SECOND}], \texttt{E}]}}
\end{verbatim}

\begin{verbatim}
\texttt{Out[2]} := \texttt{PRELUDE}
\end{verbatim}

The relation \texttt{PRELUDE} is well-founded and its inverse is thin, that is, every vertical section of \texttt{PRELUDE} is a set. The
relation \texttt{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 \texttt{id[\texttt{GAMES}]} which takes any game to itself as the unique solution of a certain recursion equation.

\textbf{prelude-induction}

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

\textbf{Theorem}. A basic form of prelude induction.
Corollary. (A better rewrite rule for prelude induction.)

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 \( \text{id[GAMES]} \).

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

\[ \text{Theorem. A recursion equation satisfied by } \text{id[GAMES].} \]

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

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

**Lemma.** (This contains the redundant literal \text{FUNCTION}[w].)
\textbf{id-games.nb}

\begin{verbatim}
In[12]:= SubstTest[implies, 
   and[equal[w, funpart[t]], equal[w, composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]]], 
   equal[domain[w], GAMES], t \rightarrow 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]:= (% /. \texttt{w} \rightarrow \texttt{w_}) /. Equal \rightarrow \texttt{SetDelayed}

Lemma. The vertical section of \texttt{game[x]}

Theorem. The recursion equation implies \texttt{domain[w] = GAMES}.

In[14]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3], not[implies[p1, p3]], 
   \{p1 \rightarrow equal[w, composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]]], 
   p2 \rightarrow FUNCTION[w], p3 \rightarrow 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 \texttt{APPLY}.

In[16]:= SubstTest[implies, equal[w, t], equal[APPLY[w, z], APPLY[t, z]], 
   \{t \rightarrow composite[cross[IMAGE[w], IMAGE[w]], id[GAMES]], z \rightarrow 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

\end{verbatim}

\begin{center}
\underline{Simplification rules for game[x]}
\end{center}

\begin{verbatim}
It will be useful to introduce the wrapper \texttt{game[x]} in subsequent work.

Lemma. The vertical section of \texttt{inverse[PRELUDE]} at \texttt{games[x]}.

In[18]:= SubstTest[APPLY, VERTSEC[t], game[x], t \rightarrow 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
\end{verbatim}
Lemma. Dual simplification rule.

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

Lemma.

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

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]]]],
      notsubclass[first[game[x]], fix[funpart[w]]]],
      notsubclass[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[second[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[second[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.)
The elimination of the game wrapper in the following theorem is taken as an opportunity to formally introduce the PRELUDE relation.

Theorem.

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].

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

Lemma.
Main Theorem. The only solution of the recursion equation \( w = (\text{IMAGE}[w] \otimes \text{IMAGE}[w]) \circ \text{id}[\text{GAMES}] \) is \( w = \text{id}[\text{GAMES}] \).

\[\begin{align*}
\text{In[45]} & : \text{Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],}
\text{ implies[p1, p4], implies[and[p2, p3, p4], p5], not[implies[p1, p5]],}
\text{ {p1 -> equal[w, composite[cross[IMAGE[w], IMAGE[w]], id[\text{GAMES}]]}],}
\text{ p2 -> FUNCTION[w], p3 -> equal[domain[w], \text{GAMES}],}
\text{ p4 -> subclass[\text{GAMES}, fix[w]], p5 -> equal[w, id[\text{GAMES}]]]}} \quad \text{// Reverse}
\text{Out[45]} = \text{or[equal[w, id[\text{GAMES}]],}
\text{ not[equal[w, composite[cross[\text{IMAGE}[w], \text{IMAGE}[w]], id[\text{GAMES}]]]]} \quad \text{=} \quad \text{True}
\text{In[46]} : = \text{(\% /. \text{w} -> \text{w}_\_)} /. \text{Equal} \rightarrow \text{SetDelayed}
\text{Corollary. A better rewrite rule.}
\text{In[47]} : = \text{equiv[equal[w, composite[cross[\text{IMAGE}[w], \text{IMAGE}[w]], id[\text{GAMES}]]], equal[w, id[\text{GAMES}]]]}
\text{Out[47]} = \text{True}
\text{In[49]} : = \text{equal[w\_, composite[cross[\text{IMAGE}[w\_], \text{IMAGE}[w\_]], id[\text{GAMES}]]] := equal[w, id[\text{GAMES}]]}