well-founded recursion, part 5.

Johan G. F. Belinfante  
2004 September 12

Summary

In this fifth notebook on well-founded recursion, a sufficient condition is derived for the solution \( \text{rec}[x,y] \) of a recursion relation to have domain \( V \). It is shown that if \( x \) is a function with domain \( \text{cart}[V,V] \), and if \( y \) is a thin relation whose inverse is well-founded, then \( \text{rec}[x,y] \) is a total function. The main idea is to use the well-foundedness of \( \text{inverse}[y] \) to show that the complement of the domain of \( \text{rec}[x,y] \) is empty. To establish this fact about the subvariance condition, the following characterization of \textit{subvariance} is useful:

\[
\text{assert[\forall z, \text{implies[subclass[image[y, \text{singleton}[z]], t], \text{member}[z, t]]]} = \text{subvariant[\text{inverse}[y], \text{complement}[t]]}
\]

The derivation involves several mildly complex constructions for which single letters are introduced by means of equality literals. Although this makes the formulas more readable than they would otherwise be, the presence of numerous equality literals in a statement tends to produce combinatorial explosions due to a conditional equality substitution rewrite rule which says that for any proposition \( p \), the statement \( \text{implies[equal[x,y], p]} \) is true if \( p \) is true when \( x \) is substituted for \( y \), or vice versa. A corresponding negative form of this rule has been removed from the GOEDEL program, and an \textit{equality} flag has been added to permit one to turn this substitution rule on
or off. This problem of combinatorial explosion has been circumvented by the following stratagem. First, the equality flag is turned on, and the negative form of a desired result is derived, using double negation. After making this negative result into a rewrite rule, the equality flag is then turned off and the corresponding positive result is derived using double negation a second time. A slight drawback of this approach is that the positive result can generally only be used while the equality flag remains off.

facts about history[x,y]

The introduction of the abbreviation history[x,y] serves two purposes. The first and most obvious purpose is to make it easier to read statements concerning well-founded recursion. The second purpose, which is just as important, is to shield some of its properties from rewrite rules. This is necessary to prevent uncontrolled access to the numerous rewrite rules for IMAGE and VERTSECT that would cause excessive time to be spent on automatic equality substitutions. To this end, only a few rewrite rules concerning history[x,y] will be retained, mainly ones needed for reasoning. In particular, two previously derived general rewrite rules for APPLY[history[x,y],z] and for domain[history[x,y]] have been removed from the GOEDEL program. To derive properties of history[x,y] it is convenient to introduce the following temporary abbreviation.

\[
\text{In}[3]:= \text{HISTORY}[_x, _y] := \text{composite}[
\text{id[composite[ IMAGE[composite[\text{id}[x], \text{inverse}[\text{FIRST}]]], \text{VERTSECT}[y]]],
\text{inverse}[\text{FIRST}]]
\]

Because history[x,y] has not been normalized, properties of HISTORY can be used to derive corresponding properties of history using equality substitution.

\[
\text{In}[4]:= \text{equal[HISTORY}[_x, _y], \text{history}[x, y]]
\]

\[
\text{Out}[4]= \text{True}
\]

The following related equations will be needed later.
In[5]:= SubstTest[implies, equal[u, v], equal[composite[x, u], composite[x, v]],
{u → history[w, y], v → HISTORY[w, y]}

Out[5]= equal[composite[x, history[w, y]], composite[x,
id[composite[IMAGE[composite[id[w], inverse[FIRST]]], VERTSECT[y]]],
inverse[FIRST]]] = True

In[6]:= (% / (w → w_, x → x_, y → y_)) /. Equal → SetDelayed

In[7]:= SubstTest[implies, equal[u, v],
equal[composite[u, id[z]], composite[v, id[z]]],
{u → history[w, y], v → composite[
id[composite[IMAGE[composite[id[w], inverse[FIRST]]], VERTSECT[y]]],
inverse[FIRST]]}

Out[7]= equal[composite[history[w, y], id[z]], composite[
id[composite[IMAGE[composite[id[w], inverse[FIRST]]], VERTSECT[y], id[z]]],
inverse[FIRST]]] = True

In[8]:= (% / (w → w_, y → y_, z → z_)) /. Equal → SetDelayed

One can replace either argument of history by its composite with Id

In[9]:= SubstTest[implies, and[equal[t, u], equal[u, v], equal[v, w]], equal[t, w],
{t → history[z, y], u → HISTORY[z, y], v → HISTORY[x, y], w → history[x, y]}} /. 
z → composite[Id, x]

Out[9]= equal[history[x, y], history[composite[Id, x], y]] = True

In[10]:= history[composite[Id, x_], y_] := history[x, y]

In[11]:= SubstTest[implies, and[equal[t, u], equal[u, v], equal[v, w]], equal[t, w],
{t → history[x, z], u → HISTORY[x, z], v → HISTORY[x, y], w → history[x, y]}} /. 
z → composite[Id, y]

Out[11]= equal[history[x, y], history[x, composite[Id, y]]] = True

In[12]:= history[x_, composite[Id, y_]] := history[x, y]

The following elementary fact about the range of history is needed to convert
image[inverse[history[x, y]], cart[V, V]] to domain[history[x, y]].

In[13]:= SubstTest[implies, equal[t, HISTORY[x, y]],
subclass[range[t], cart[V, V]], t → history[x, y]]


In[14]:= subclass[range[history[x_, y_]], cart[V, V]] := True

In[15]:= (% / (x → x_, y → y_)) /. Equal → SetDelayed
a technical lemma

In this section a technical lemma about partial solutions of the recursion relation is derived. This result will eventually be applied to the case that \( u \) is obtained from a partial solution \( w \) by adding a single new point. The variable \( w \) is not introduced explicitly here, but only as a restriction of \( u \) to some class \( z \) that is assumed to be invariant under \( y \). An application of the Assoc test yields:

\[
\text{In} [16]:= \text{Map}[\text{composite}[\text{IMAGE}[\text{composite}[\text{id}[u], \text{inverse}[\text{FIRST}]]], \#] \&,
\text{Assoc}[\text{IMAGE}[\text{id}[z]], \text{id}[p[z]], \text{VERTSECT}[y]]
\]
\[
\text{Out} [16]= \text{composite}[\text{IMAGE}[\text{composite}[\text{id}[\text{composite}[u, \text{id}[z]]], \text{inverse}[\text{FIRST}]]], \text{VERTSECT}[y], \text{id}[\text{complement}[\text{image}[\text{inverse}[y], \text{complement}[z]]])]
\]

\[
\text{In} [17]:= \text{composite}[\text{IMAGE}[\text{composite}[\text{id}[\text{composite}[u_, \text{id}[z_]]], \text{inverse}[\text{FIRST}]]], \text{VERTSECT}[y_], \text{id}[\text{complement}[\text{image}[\text{inverse}[y], \text{complement}[z]]])]
\]

The somewhat tricky substitution on \( t \) in the following is needed to produce the condition that \( z \) is invariant under \( y \).

\[
\text{In} [18]:= \text{SubstTest}[\text{implies}, \text{equal}[t, \text{complement}[\text{image}[\text{inverse}[y], \text{complement}[z]]]), \text{equal}[\text{composite}[\text{IMAGE}[\text{composite}[\text{id}[\text{composite}[u, \text{id}[z]]], \text{inverse}[\text{FIRST}]]], \text{VERTSECT}[y], \text{id}[t], w], \text{composite}[\text{IMAGE}[\text{composite}[\text{id}[u], \text{inverse}[\text{FIRST}]]], \text{VERTSECT}[y], \text{id}[t], w]], \{w \rightarrow \text{id}[z], t \rightarrow \text{union}[z, \text{complement}[\text{image}[\text{inverse}[y], \text{complement}[z]]])}
\]
\[
\text{Out} [18]= \text{or}[\text{equal}[\text{composite}[\text{IMAGE}[\text{composite}[\text{id}[u], \text{inverse}[\text{FIRST}]]], \text{VERTSECT}[y], \text{id}[z]], \text{composite}[\text{IMAGE}[\text{composite}[\text{id}[\text{composite}[u, \text{id}[z]]], \text{inverse}[\text{FIRST}]]], \text{VERTSECT}[y], \text{id}[z]], \text{not}[	ext{subclass}[\text{image}[y, z], z]] = \text{True}
\]

\[
\text{In} [19]:= (\% / (u \rightarrow u_\_, \text{y} \rightarrow \text{y}_\_, \text{z} \rightarrow \text{z}_\_)) / \text{Equal} \rightarrow \text{SetDelayed}
\]

Equality substitution is used to transfer facts about History to history.
SubstTest[implies, and[equal[p, q], equal[q, r], equal[r, s]], equal[p, s],
  {p -> composite[history[u, y], id[z]],
   q -> composite[HISTORY[u, y], id[z]],
   r -> composite[HISTORY[composite[u, id[z]], y], id[z]],
   s -> composite[history[composite[u, id[z]], y], id[z]]}]

or[equal[composite[history[u, y], id[z]],
      composite[history[composite[u, id[z]], y], id[z]]],
    not[equal[composite[id[composite[IMAGE[composite[id[u], inverse[FIRST]]],
                             VERTSECT[y], id[z]]], inverse[FIRST]],
         composite[id[composite[IMAGE[composite[id[composite[u, id[z]]], inverse[FIRST]]],
                             VERTSECT[y], id[z]]], inverse[FIRST]]],
    VERTSECT[y], id[z]],
  p1 -> equal[composite[HISTORY[u, y], id[z]],
              composite[HISTORY[composite[u, id[z]], y], id[z]],
              p2 -> equal[composite[IMAGE[composite[id[u], inverse[FIRST]]],
                             VERTSECT[y], id[z]],
              composite[IMAGE[composite[id[composite[u, id[z]]], inverse[FIRST]]],
                             VERTSECT[y], id[z]],
              p3 -> equal[composite[composite[HISTORY[u, y], id[z]],
                             composite[HISTORY[composite[u, id[z]], y], id[z]]],
              p4 -> equal[composite[history[u, y], id[z]],
                             composite[history[composite[u, id[z]], y], id[z]],
              p5 -> equal[composite[x, history[u, y], id[z]],
                             composite[x, history[composite[u, id[z]], y], id[z]]}]

or[equal[composite[x, history[u, y], id[z]],
      composite[x, history[composite[u, id[z]], y], id[z]]],
    notsubclass[composite[y, id[z]], y]] = True

(% / (u -> u_, y -> y_, z -> z_) /. Equal -> SetDelayed)

```
In[22]:= Map[not, SubstTest[and, implies[p1, p2],
                              implies[p2, p3], implies[p3, p4], implies[p4, p5],
                              not[implies[p1, p5]], {p1 -> subclass[IMAGE[y, z], z]},
                              p2 -> equal[composite[IMAGE[composite[id[u], inverse[FIRST]]],
                                           VERTSECT[y], id[z]],
                              composite[IMAGE[composite[id[composite[u, id[z]]], inverse[FIRST]]],
                                           VERTSECT[y], id[z]],
                              p3 -> equal[composite[HISTORY[u, y], id[z]],
                                          composite[HISTORY[composite[u, id[z]], y], id[z]]],
                              p4 -> equal[composite[history[u, y], id[z]],
                                          composite[history[composite[u, id[z]], y], id[z]]],
                              p5 -> equal[composite[x, history[u, y], id[z]],
                                          composite[x, history[composite[u, id[z]], y], id[z]]}]]
```

(% / (u -> u_, x -> x_, y -> y_, z -> z_) /. Equal -> SetDelayed)

Lemma.

```
In[24]:= SubstTest[implies, and[subclass[r, s], equal[s, t]], subclass[r, t],
                   {r -> composite[w, id[z]], s -> composite[x, id[z]], t -> composite[y, id[z]]}]
```

(% / (not[equal[composite[x, id[z]], composite[y, id[z]]]],
      notsubclass[composite[w, id[z]], x]],
      subclass[composite[w, id[z]], y]] = True

(% / (w -> w_, x -> x_, y -> y_, z -> z_) /. Equal -> SetDelayed)
If a partial solution \( w \) of the recursion condition is a restriction of \( u \), then \( w \) is contained in \( t = \text{composite}[x, \text{history}[u,y]] \).

If \( u \) is obtained from a function \( w \) by adding a new point \( \text{PAIR}[z,v] \), and if \( z \) is not in the domain of \( w \), then \( u \) is a function.
Any subclass of a function is a restriction, so \( w \) is a restriction of \( u \).

If \( z \) is not in the domain of \( \text{rec}[x,y] \), and if \( u \) is obtained from a partial solution \( w \) by adding the point \( \text{PAIR}[z,y] \), then \( w \) is contained in \( t = \text{composite}[x,\text{history}[u,y]] \).

If \( u \) is obtained from \( w \) by adding a single point \( \text{PAIR}[z,v] \), where \( z \) is not in the domain of \( \text{rec}[x,y] \), then \( u \) cannot be a partial solution.
In[36]:=  Map[not, SubstTest[and, implies[and[p0, p2], p3],
     implies[and[p1, p3], p4], implies[and[p4, p5], implies[and[p3, p5], p6],
     not[implies[and[p0, p1, p2], p6]],
     {p0 -> equal[u, union[w, cart[singleton[z], singleton[v]]]],
     p1 -> member[u, partrec[x, y]], p2 -> and[member[z, V], member[v, V]],
     p3 -> member[z, domain[u]], p4 -> subclass[u, rec[x, y]],
     p5 -> subclass[domain[u], domain[rec[x, y]]],
     p6 -> member[z, domain[rec[x, y]]]]]

Out[36]= or[member[z, domain[rec[x, y]]],
     not[equal[u, union[w, cart[singleton[z], singleton[v]]]],
     not[member[u, partrec[x, y]]], not[member[v, V]],
     not[member[z, V]]] = True

In[37]:=  (% /. {u -> u_, v -> v_, w -> w_, x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed

This can be made more explicit by rewriting the literal member[u, partrec[x,y]] in
terms of history[u,y], using the following:

In[38]:=  equiv[member[u, partrec[x, y]], and[member[u, V],
     invariant[y, domain[u]],
     subclass[u, composite[x, history[u, y]]]]] // not // not

Out[38]=  True

This yields

In[39]:=  Map[not, SubstTest[and, implies[and[p1, p2, p3], p4],
     implies[and[p4, p5, p6], p7], implies[and[p1, p7, p8], p9],
     not[implies[and[p1, p2, p3, p5, p6, p8], p9]],
     {p1 -> equal[u, union[w, cart[singleton[z], singleton[v]]]],
     p2 -> subclass[w, composite[x, history[u, y]]], p3 ->
     subclass[cart[singleton[z], singleton[v]], composite[x, history[u, y]]],
     p4 -> subclass[u, composite[x, history[u, y]]],
     p5 -> member[u, V], p6 -> invariant[y, domain[u]],
     p7 -> member[u, partrec[x, y]], p8 -> and[member[z, V], member[v, V]],
     p9 -> member[z, domain[rec[x, y]]]]]

Out[39]=  or[member[z, domain[rec[x, y]]],
     not[equal[u, union[w, cart[singleton[z], singleton[v]]]],
     not[member[u, V]], not[member[pair[z, v],
    composite[x, history[u, y]]]],
     not[subclass[w, composite[x, history[u, y]]]],
     not[subclass[image[y, domain[u]], domain[u]]]] = True

In[40]:=  (% /. {u -> u_, v -> v_, w -> w_, x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed

The statement that u is a set follows from the fact that adding a singleton to a set pro-
duces another set. The invariance of the domain of u introduces the special hypothesis
subclass[image[y, singleton[z]], domain[w]] that will later need to be
removed. The condition that \( w \) is contained in \( t \) was established previously. The above statement can thus be simplified:

\[
\text{In}[41] := \text{Map}[\text{not}, \text{SubstTest}[\text{and, implies[and[p2, p3], p4]}, \text{implies[and[p1, p2, p3], p5]}],
\text{implies[and[p0, p2, p3, p6], p7]}, \text{implies[and[p3, p4, p5, p7, p6], p8]}],
\text{not}[\text{implies[and[p0, p1, p2, p3, p6], p8]}],
\{p0 \rightarrow \text{FUNCTION}[x], p1 \rightarrow \text{subclass[image[y, singleton[z]], domain[w]}],
\text{p2 \rightarrow member[w, partrec[x, y]}],
\text{p3 \rightarrow equal[u, union[w, cart[singleton[z], singleton[v]]]}],
\text{p4 \rightarrow member[u, V], p5 \rightarrow invariant[y, domain[u]}],
\text{p6 \rightarrow not[member[z, domain[rec[x, y]]]}],
\text{p7 \rightarrow subclass[w, composite[x, history[u, y]]]}],
\text{p8 \rightarrow not[member[pair[z, v], composite[x, history[u, y]]]}]]
\]

\[
\text{Out}[41] = \text{or[member[z, domain[rec[x, y]]]}],
\text{not[equal[u, union[w, cart[singleton[z], singleton[v]]]}],
\text{not[FUNCTION[x]], not[member[w, partrec[x, y]]]}],
\text{not[member[pair[z, v], composite[x, history[u, y]]]}],
\text{not[subclass[image[y, singleton[z]], domain[w]]]}] = \text{True}
\]

\[
\text{In}[42] := (\% /. \{u \rightarrow _-, v \rightarrow _-, w \rightarrow _-, x \rightarrow _-, y \rightarrow _-, z \rightarrow _-\}) /. \text{Equal} \rightarrow \text{SetDelayed}
\]

It only remains to deal with the literal involving \text{pair[z,v]}. 

totality for \( t \)

In this section, a sufficient condition is derived for \( t = \text{composite}[x,\text{history}[u,y]] \) to be total.

\[
\text{In}[43] := \text{SubstTest[implies, and[equal[s, composite[x, HISTORY[u, thinpart[y]]]}],
\text{equal[domain[x], cart[V, V]}],
\text{equal[u, union[cart[singleton[z], singleton[v]], funpart[w]]]}], \text{equal[t, s]}],
\text{equal[V, domain[t]}], s \rightarrow \text{composite[x, history[u, thinpart[y]]]}]]
\]

\[
\text{Out}[43] = \text{or[equal[V, domain[t]}], \text{not[equal[t, composite[x, history[u, thinpart[y]]]}]]],
\text{not[equal[u, union[cart[singleton[z], singleton[v]], funpart[w]]]}],
\text{not[equal[cart[V, V], domain[x]]]}] = \text{True}
\]

\[
\text{In}[44] := (\% /. \{t \rightarrow _-, u \rightarrow _-, v \rightarrow _-, w \rightarrow _-, x \rightarrow _-, y \rightarrow _-, z \rightarrow _-\}) /. \text{Equal} \rightarrow \text{SetDelayed}
\]

The \text{funpart} and \text{thinpart} wrappers can be removed:
eliminating the literal involving pair[z,v]

The literal involving the term pair[z,v] will be rewritten as v = APPLY[t, z]. In this section, the trick of turning the equality flag on and off is heavily used. With the flag on, a falsehood is derived. Even with all this effort, the following still takes 24 seconds.

The corresponding positive result is now obtained by double negation after turning the equality flag off.

\[
equality = \text{False};
\]
At this point one can eliminate the literal involving `pair[z,v]`. 
The \texttt{Thinpart} wrapper can now be removed.
One cannot immediately use this to eliminate `APPLY[t,z]` with the equality flag turned off because two needed results are only recognized as true with the equality flag on. These results must first be made available for use with the equality flag off. This is accomplished by using double negation and turning that flag on and off.

```plaintext
In[64]:= (% /. {t -> t_, u -> u_, v -> v_, w -> w_, x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed

Out[64]= True
```

```plaintext
In[65]:= SubstTest[implies, and[equal[s, thinpart[y]],
    equal[t, composite[x, history[u, s]]],
    equal[APPLY[t, z], APPLY[x, PAIR[z, composite[u, id[image[s, singleton[z]]]]]]], s -> composite[Id, y]]

Out[65]= or[equal[APPLY[t, z],
            APPLY[x, PAIR[z, composite[u, id[image[y, singleton[z]]]]]]],
         not[equal[t, composite[x, history[u, y]]]],
         not[equal[V, domain[VERTSECT[y]]]]] = True
```

```plaintext
In[66]:= (% /. {t -> t_, u -> u_, x -> x_, y -> y_}) /. Equal -> SetDelayed

Out[66]= False
```

```plaintext
In[67]:= and[equal[u, union[w, cart[singleton[z], singleton[v]]]],
    equal[v, APPLY[t, z]],
    not[equal[u, union[w, cart[singleton[z], singleton[APPLY[t, z]]]]]]] // NotNotTest

Out[67]= and[equal[u, union[w, cart[singleton[z], singleton[v]]]],
    equal[v, APPLY[t, z]],
    not[equal[u, union[w, cart[singleton[z], singleton[APPLY[t, z]]]]]]] = False
```

```plaintext
In[68]:= (% /. {t -> t_, u -> u_, x -> x_, y -> y_}) /. Equal -> SetDelayed

Out[68]= False
```

```plaintext
In[69]:= equality = False;
```

```plaintext
In[70]:= implies[and[thin[y], equal[t, composite[x, history[u, y]]],
    equal[v, APPLY[x, PAIR[z, composite[u, id[image[y, singleton[z]]]]]]]],
    equal[APPLY[t, z], v]] // NotNotTest

Out[70]= or[equal[v, APPLY[t, z]], not[equal[t, composite[x, history[u, y]]]],
         not[equal[v, APPLY[x, PAIR[z, composite[u, id[image[y, singleton[z]]]]]]]],
         not[equal[V, domain[VERTSECT[y]]]]] = True
```

```plaintext
In[71]:= (% /. {t -> t_, u -> u_, v -> v_, w -> w_, x -> x_, y -> y_}) /. Equal -> SetDelayed
```
This can be reformulated so that \( t \) only occurs in its definition and the literal \( v = \text{APPLY}[t, z] \).

At this point the variable \( t \) is eliminated, with the \texttt{equality} flag still off.
the elimination of \( u \) and \( v \)

The variable \( u \) still occurs in two literals, namely its definition and one that involves \texttt{APPLY}. The latter occurrence can be eliminated if one adds the assumption that \( \text{fix}[y] = 0 \).

\begin{verbatim}
In[78]:= equality = True;

In[79]:= and[equal[0, fix[y]], equal[u, union[w, cart[singleton[z], singleton[v]]]],
          equal[v, APPLY[x, PAIR[z, composite[w, id[image[y, singleton[z]]]]]]],
          not[equal[v, APPLY[x, PAIR[z, composite[u, id[image[y, singleton[z]]]]]]]]] // NotNotTest

Out[79]= and[equal[0, fix[y]], equal[u, union[w, cart[singleton[z], singleton[v]]]],
          equal[v, APPLY[x, PAIR[z, composite[w, id[image[y, singleton[z]]]]]]],
          not[equal[v, APPLY[x, PAIR[z, composite[u, id[image[y, singleton[z]]]]]]]]] = False

In[80]:= (% //. {u -> u_, v -> v_, w -> w_, x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed

In[81]:= equality = False;
\end{verbatim}
The only class subvariant under a well-founded relation whose inverse is thin is the empty set. The intention is to apply this to the special case that the subvariant class is the complement of the domain of \texttt{rec[x,y]}. 

eliminating the partial solution \texttt{w}
Recall that the requisite subvariance condition can be derived by showing that \(\text{subclass}[\text{image}[y, \text{singleton}[z]], \text{domain}[\text{rec}[x,y]]]\) implies that \(\text{member}[z, \text{domain}[\text{rec}[x,y]]]\). The first step is to replace \(y\) by its transitive closure \(\text{trv}[y]\) to obtain a set that is invariant under \(y\).

If \(y\) is thin, so is its transitive closure, so \(\text{image}[\text{trv}[y], \text{singleton}[z]]\) is a set.

When \(x\) is a function and \(y\) is a thin relation whose inverse is well-founded, any restriction of \(\text{rec}[x,y]\) to a set that is invariant under \(y\) is a partial solution.
This can be simplified:

One can introduce the abbreviation \( w \) for this partial solution.

Note that the condition \( \text{subclass}[\text{image}[y, \text{singleton}[z]], \text{domain}[w]] \) holds for this partial solution.
It is also convenient to dispose of the condition \( \text{fix}[y] = 0 \) now.
In[110]:=
Map[not, SubstTest[and, implies[and[p1, p2], p5], implies[and[p2, p4], p6],
   implies[p1, p7], implies[and[p1, p3, p5, p6, p7], p8],
   not[implies[and[p1, p2, p3, p4], p8]], (p1 \[Rightarrow] and[FUNCTION[x],
       thin[y], WELLFOUNDED[Inverse[y]], equal[cart[V, V], domain[x]]],
       p2 \[Rightarrow] equal[w, composite[rec[x, y], id[image[trv[y], singleton[z]]]]],
       p3 \[Rightarrow] member[z, V],
       p4 \[Rightarrow] subclass[image[y, singleton[z]], domain[rec[x, y]]],
       p5 \[Rightarrow] member[w, partrec[x, y]],
       p6 \[Rightarrow] subclass[image[y, singleton[z]], domain[w]],
       p7 \[Rightarrow] equal[0, fix[y]], p8 \[Rightarrow] member[z, domain[rec[x, y]]]]] /. w \[Rightarrow] composite[rec[x, y], id[image[trv[y], singleton[z]]]]

Out[110]=
or[member[z, domain[rec[x, y]]], not[equal[V, domain[VERTSECT[y]]]]],
not[equal[cart[V, V], domain[x]]], not[FUNCTION[x]], not[member[z, V]],
not[subclass[image[y, singleton[z]], domain[rec[x, y]]]],
not[WELLFOUNDED[Inverse[y]]]] == True

In[111]:=

eliminating \(z\)

Temporary rule.

In[112]:=
member[z, image[Inverse[y], complement[x]]] // AssertTest

Out[112]=
member[z, image[Inverse[y], complement[x]]] =
not[subclass[image[y, singleton[z]], x]]

In[113]:=
member[z_, image[Inverse[y_], complement[x_]]] :=
not[subclass[image[y, singleton[z]], x]]

The final step is to eliminate \(z\). It helps here to turn off all the flags for conditional rewrite rules, and to sequester various quantities from the action of \texttt{class}.

In[114]:=
simplify = False; cond = False;
In[115]:=
Map[equal[V, #] & , SubstTest[class, z,
    or[member[z, r], not[equal[V, s]], not[equal[cart[V, V], t]],
    not[subclass[u, v]], not[member[z, w]], not[subclass[p, q]]],
{p -> P[inverse[y]], q -> WF, r -> domain[rec[x, y]],
    w -> complement[image[inverse[y], complement[domain[rec[x, y]]]]],
    s -> domain[VERTSET[y]], t -> domain[x], u -> P[x], v -> FUNS}] // Reverse

Out[115]=
    or[equal[V,
      union[domain[rec[x, y]], image[inverse[y], complement[domain[rec[x, y]]]]],
      not[equal[V, domain[VERTSET[y]]]], not[equal[cart[V, V], domain[x]]],
      not[FUNCTION[x]], not[WELLFOUNDED[inverse[y]]]] == True

In[116]:=
    (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed

The main result of this notebook is the following sufficient condition for \texttt{rec[x\_y]} to be total.

In[117]:=
Map[not, SubstTest[and, implies[p1, p2],
    implies[and[p1, p2], p3], not[implies[p1, p3]], {p1 -> and[FUNCTION[x],
    equal[cart[V, V], domain[x]], thin[y], WELLFOUNDED[inverse[y]]],
    p2 -> equal[V, union[domain[rec[x, y]], image[inverse[y],
      complement[domain[rec[x, y]]]]], p3 -> equal[V, domain[rec[x, y]]]}]

Out[117]=
    or[equal[V, domain[rec[x, y]]],
      not[equal[V, domain[VERTSET[y]]]], not[equal[cart[V, V], domain[x]]],
      not[FUNCTION[x]], not[WELLFOUNDED[inverse[y]]]] == True

In[118]:=
    or[equal[V, domain[rec[x\_y]]],
      not[equal[V, domain[VERTSET[y\_]]]], not[equal[cart[V, V], domain[x\_]]],
      not[FUNCTION[x\_]], not[WELLFOUNDED[inverse[y\_]]]] == True