well-founded recursion, part 5.

Johan G. F. Belinfante
2004 September 12

```
In[1]:= SetDirectory["i:"]; << goedel61.08a; << tools.m;

        :Package Title: goedel61.08a          2004 September 8 at 6:10 p.m.

        It is now:  2004 Sep 12 at 13:51

        Loading Simplification Rules

        TOOLS.M                      Revised 2004 September 5

        weightlimit = 40
```

---

summary

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

```
In[2]:= assert[forall[z, implies[subclass[image[y, singleton[z]], t], member[z, t]]]] ==
         subvariant[inverse[y], complement[t]]

Out[2]= True
```

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 **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 **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 termporary abbreviation.

```
In[3]:= HISTORY[x_, y_] := composite[
          id[composite[IMAGE[composite[id[x], inverse[FIRST]]], VERTSECT[y]]],
          inverse[FIRST]]
```

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

```
In[4]:= equal[HISTORY[x, y], history[x, y]]

Out[4]= 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]]

Out[13]= subclass[range[history[x, y]], cart[V, V]] == True

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:

```
In[16]:= Map[composite[IMAGE[composite[id[u], inverse[FIRST]]], #] &,
          Assoc[IMAGE[id[z]], id[P[z]], VERTSECT[y]]]

Out[16]= composite[IMAGE[composite[id[composite[u, id[z]]], inverse[FIRST]]],
           VERTSECT[y], id[complement[image[inverse[y], complement[z]]]]] ==
         composite[IMAGE[composite[id[u], inverse[FIRST]]], VERTSECT[y],
          id[complement[image[inverse[y], complement[z]]]]]
```

```
In[17]:= composite[IMAGE[composite[id[composite[u_, id[z_]]], inverse[FIRST]]],
           VERTSECT[y_], id[complement[image[inverse[y_], complement[z_]]]]] :=
         composite[IMAGE[composite[id[u], inverse[FIRST]]], VERTSECT[y],
          id[complement[image[inverse[y], complement[z]]]]]
```

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

```
In[18]:= SubstTest[implies, equal[t, complement[image[inverse[y], complement[z]]]],
          equal[composite[IMAGE[composite[id[composite[u, id[z]]], inverse[FIRST]]],
            VERTSECT[y], id[t], w],
           composite[IMAGE[composite[id[u], inverse[FIRST]]], VERTSECT[y], id[t], w]],
          {w → id[z], t → union[z, complement[image[inverse[y], complement[z]]]]}]

Out[18]= or[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]]], not[subclass[image[y, z], z]]] == True
```

```
In[19]:= (% /. {u → u_, y → y_, z → z_}) /. Equal → SetDelayed
```

Equality substitution is used to transfer facts about **HISTORY** to **history**.

```
In[20]:= 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]]}]

Out[20]= 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]]]]] == True

In[21]:= (% /. {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]]]}]]

Out[22]= or[equal[composite[x, history[u, y], id[z]],
            composite[x, history[composite[u, id[z]], y], id[z]]],
           not[subclass[image[y, z], z]]] == True

In[23]:= (% /. {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]]}]

Out[24]= or[not[equal[composite[x, id[z]], composite[y, id[z]]]],
           not[subclass[composite[w, id[z]], x]],
           subclass[composite[w, id[z]], y]] == True

In[25]:= (% /. {w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

```
In[26]:= Map[not, SubstTest[and, implies[p0, p4], implies[p1, p2],
           implies[p1, p3], implies[and[p3, p4], p5], not[implies[and[p0, p1], p5]],
           {p0 → member[composite[u, id[z]], partrec[x, y]], p1 → invariant[y, z],
            p2 → subclass[image[y, z], z], p3 -> equal[composite[x, history[u, y],
               id[z]], composite[x, history[composite[u, id[z]], y], id[z]]],
            p4 -> subclass[composite[u, id[z]], composite[x,
               history[composite[u, id[z]], y]]],
            p5 → subclass[composite[u, id[z]], composite[x, history[u, y]]]}]]

Out[26]= or[not[member[composite[u, id[z]], partrec[x, y]]],
           not[subclass[image[y, z], z]],
           subclass[composite[u, id[z]], composite[x, history[u, y]]]] == True

In[27]:= (% /. {u → u_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

---

recursion condition

If a partial solution **w** of the recursion condition is a restriction of **u**, then **w** is contained in **t = composite[x, history[u,y]]**.

```
In[28]:= Map[not, SubstTest[and, implies[p2, p3],
           implies[and[p1, p2, p3, p5], p6], implies[and[p2, p4], p5],
           implies[and[p4, p6], p7], not[implies[and[p1, p2, p4], p7]],
           {p1 -> member[w, partrec[x, y]], p2 -> member[w, partrec[x, y]],
            p3 → invariant[y, domain[w]], p4 -> equal[w, composite[u, id[domain[w]]]],
            p5 → member[composite[u, id[domain[w]]], partrec[x, y]],
            p6 → subclass[composite[u, id[domain[w]]], composite[x, history[u, y]]],
            p7 → subclass[w, composite[x, history[u, y]]]}]]

Out[28]= or[not[equal[w, composite[u, id[domain[w]]]]], not[member[w, partrec[x, y]]],
           subclass[w, composite[x, history[u, y]]]] == True

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

If **u** is obtained from a function **w** by adding a new point **PAIR[z,v]**, and if **z** is not in the domain of **w**, then **u** is a function.

```
In[30]:= Map[not, SubstTest[and, implies[and[p1, p2], p5], implies[and[p2, p4], p6],
           implies[and[p3, p5, p6], p7], not[implies[and[p1, p2, p3, p4], p7]],
           {p1 → FUNCTION[x], p2 → member[w, partrec[x, y]],
            p3 → equal[u, union[w, cart[singleton[z], singleton[v]]]],
            p4 → not[member[z, domain[rec[x, y]]]], p5 → FUNCTION[w],
            p6 → not[member[z, domain[w]]], p7 → FUNCTION[u]}]]

Out[30]= or[FUNCTION[u], member[z, domain[rec[x, y]]],
           not[equal[u, union[w, cart[singleton[z], singleton[v]]]]],
           not[FUNCTION[x]], not[member[w, partrec[x, y]]] == True

In[31]:= (% /. {u → u_, v → v_, w → w_, x → x_, y → y_, z -> z_}) /. Equal → SetDelayed
```

Any subclass of a function is a restriction, so **w** is a restriction of **u**.

```
In[32]:= Map[not, SubstTest[and, implies[and[p1, p2, p3, p4], p5],
           implies[p4, p6], implies[and[p5, p6], p7],
           not[implies[and[p1, p2, p3, p4], p7]], {p1 → FUNCTION[x],
            p2 → member[w, partrec[x, y]], p3 → not[member[z, domain[rec[x, y]]]],
            p4 → equal[u, union[w, cart[singleton[z], singleton[v]]]], p5 → FUNCTION[u],
            p6 → subclass[w, u], p7 → equal[w, composite[u, id[domain[w]]]]}]]

Out[32]= or[equal[w, composite[u, id[domain[w]]]], member[z, domain[rec[x, y]]],
           not[equal[u, union[w, cart[singleton[z], singleton[v]]]]],
           not[FUNCTION[x]], not[member[w, partrec[x, y]]] == True

In[33]:= (% /. {u → u_, v → v_, w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

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

```
In[34]:= Map[not, SubstTest[and, implies[and[p1, p2, p3, p4], p5],
           implies[p2, p6], implies[and[p2, p5, p6], p7],
           not[implies[and[p1, p2, p3, p4], p7]], {p1 -> FUNCTION[x],
            p2 -> member[w, partrec[x, y]], p3 -> not[member[z, domain[rec[x, y]]]],
            p4 -> equal[u, union[w, cart[singleton[z], singleton[v]]]],
            p5 → equal[w, composite[u, id[domain[w]]]], p6 → invariant[y, domain[w]],
            p7 → subclass[w, composite[x, history[u, y]]]}]]

Out[34]= or[member[z, domain[rec[x, y]]],
           not[equal[u, union[w, cart[singleton[z], singleton[v]]]]],
           not[FUNCTION[x]], not[member[w, partrec[x, y]]],
           subclass[w, composite[x, history[u, y]]]] == True

In[35]:= (% /. {u → u_, v → v_, w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

If **u** is obtained from **w** be adding a single point **PAIR[z,v]**, where **z** is not in the domain of **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[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 produces 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.condition follows. The condition that **w** is contained in **t** was established previously. The above statement can thus be simplified:

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

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

In[42]:= (% /. {u → u_, v → v_, w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

It only remains to deal with the literal involving **pair[z,v]**.

---

totality for t

In this section, a sufficient condition is derived for **t = composite[x,history[u,y]]** to be total.

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

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

In[44]:= (% /. {t → t_, u → u_, v → v_, w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

The **funpart** and **thinpart** wrappers can be removed:

```
In[45]:= SubstTest[implies, and[equal[r, funpart[w]],
           equal[s, thinpart[y]], equal[t, composite[x, history[u, s]]],
           equal[u, union[cart[singleton[z], singleton[v]], r]],
           equal[cart[V, V], domain[x]]],
          equal[V, domain[t]], {r → w, s → composite[Id, y]}]

Out[45]= or[equal[V, domain[t]], not[equal[t, composite[x, history[u, y]]]],
           not[equal[u, union[w, cart[singleton[z], singleton[v]]]]],
           not[equal[V, domain[VERTSECT[y]]]],
           not[equal[cart[V, V], domain[x]]], not[FUNCTION[w]]] == True

In[46]:= (% /. {t → t_, u → u_, v → v_, w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

---

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.

```
In[47]:= SubstTest[and, implies[p1, p2], implies[and[p1, p3, p4], p5],
          implies[and[p2, p5, p6], p7], not[implies[and[p1, p3, p4, p6], p7]],
          {p1 -> and[FUNCTION[x], thin[y], equal[t, composite[x, history[u, y]]]],
           p2 → FUNCTION[t], p3 → equal[cart[V, V], domain[x]],
           p4 → and[FUNCTION[w], equal[u, union[w, cart[singleton[z], singleton[v]]]]],
           p5 → equal[V, domain[t]], p6 → member[z, V],
           p7 -> member[pair[z, APPLY[t, z]], t]}]

Out[47]= and[equal[t, composite[x, history[u, y]]],
           equal[u, union[w, cart[singleton[z], singleton[v]]]],
           equal[V, domain[VERTSECT[y]]], equal[cart[V, V], domain[x]], FUNCTION[w],
           FUNCTION[x], member[z, V], not[member[pair[z, APPLY[t, z]], t]]] == False

In[48]:= (% /. {t → t_, u → u_, v → v_, w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

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

```
In[49]:= equality = False;
```

```
In[50]:= or[member[pair[z, APPLY[t, z]], t], not[equal[t, composite[x, history[u, y]]]],
           not[equal[u, union[w, cart[singleton[z], singleton[v]]]]],
           not[equal[V, domain[VERTSECT[y]]]], not[equal[cart[V, V], domain[x]]],
           not[FUNCTION[w]], not[FUNCTION[x]], not[member[z, V]]] // NotNotTest

Out[50]= or[member[pair[z, APPLY[t, z]], t], not[equal[t, composite[x, history[u, y]]]],
           not[equal[u, union[w, cart[singleton[z], singleton[v]]]]],
           not[equal[V, domain[VERTSECT[y]]]], not[equal[cart[V, V], domain[x]]],
           not[FUNCTION[w]], not[FUNCTION[x]], not[member[z, V]]] == True

In[51]:= (% /. {t → t_, u → u_, v → v_, w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed

In[52]:= equality = True;

In[53]:= and[equal[t, composite[x, history[u, y]]],
           equal[u, union[w, cart[singleton[z], singleton[v]]]],
           FUNCTION[x], member[w, partrec[x, y]],
           member[pair[z, v], t], not[member[z, domain[rec[x, y]]]],
           subclass[image[y, singleton[z]], domain[w]]] // NotNotTest

Out[53]= and[equal[t, composite[x, history[u, y]]],
           equal[u, union[w, cart[singleton[z], singleton[v]]]],
           FUNCTION[x], member[w, partrec[x, y]],
           member[pair[z, v], t], not[member[z, domain[rec[x, y]]]],
           subclass[image[y, singleton[z]], domain[w]]] == False

In[54]:= (% /. {t → t_, u → u_, v → v_, w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed

In[55]:= equality = False;

In[56]:= implies[and[FUNCTION[x], member[w, partrec[x, y]],
             equal[u, union[w, cart[singleton[z], singleton[v]]]],
             equal[t, composite[x, history[u, y]]], member[pair[z, v], t],
             subclass[image[y, singleton[z]], domain[w]]],
           member[z, domain[rec[x, y]]]] // NotNotTest

Out[56]= or[member[z, domain[rec[x, y]]], not[equal[t, composite[x, history[u, y]]]],
           not[equal[u, union[w, cart[singleton[z], singleton[v]]]]],
           not[FUNCTION[x]], not[member[w, partrec[x, y]]], not[member[pair[z, v], t]],
           not[subclass[image[y, singleton[z]], domain[w]]]] == True

In[57]:= (% /. {t → t_, u → u_, v → v_, w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

At this point one can eliminate the literal involving **pair[z,v]**.

```
In[58]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
           implies[and[p1, p3, p4, p5], p6], implies[and[p1, p2, p5, p6, p7], p8],
           not[implies[and[p1, p2, p4, p5, p7], p8]],
           {p1 → FUNCTION[x], p2 -> member[w, partrec[x, y]], p3 → FUNCTION[w],
            p4 → and[equal[cart[V, V], domain[x]], thin[y], member[z, V]],
            p5 → and[equal[u, union[w, cart[singleton[z], singleton[APPLY[t, z]]]]],
              equal[t, composite[x, history[u, y]]]],
            p6 -> member[pair[z, APPLY[t, z]], t],
            p7 -> subclass[image[y, singleton[z]], domain[w]],
            p8 -> member[z, domain[rec[x, y]]]}]]

Out[58]= or[member[z, domain[rec[x, y]]], not[equal[t, composite[x, history[u, y]]]],
           not[equal[u, union[w, cart[singleton[z], singleton[APPLY[t, z]]]]]],
           not[equal[V, domain[VERTSECT[y]]]], not[equal[cart[V, V], domain[x]]],
           not[FUNCTION[x]], not[member[w, partrec[x, y]]], not[member[z, V]],
           not[subclass[image[y, singleton[z]], domain[w]]]] == True

In[59]:= (% /. {t → t_, u → u_, w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

APPLY formulas used to eliminate t

One by one all the introduced variables **t**, **u**, **v**, **w** and **z** will be removed, starting with the variable **t** in this section. To prepare for this, the variable **t** needs to be removed from all literals except the one equation that defines it. Fortunately there is only one occurrence of **t** that needs to be eliminated, namely its occurrence in the term **APPLY[t,z]**. This is easy to do when **y** is wrapped with **thinpart**.

```
In[60]:= equality = True;

In[61]:= Map[not,
          SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 -> equal[t, composite[x, history[u, thinpart[y]]]],
            p2 -> equal[t, composite[x, HISTORY[u, thinpart[y]]]],
            p3 -> equal[APPLY[t, z], APPLY[composite[x, HISTORY[u, thinpart[y]]], z]]}]]

Out[61]= or[equal[APPLY[t, z],
            APPLY[x, PAIR[z, composite[u, id[image[thinpart[y], singleton[z]]]]]]],
           not[equal[t, composite[x, history[u, thinpart[y]]]]]] == True

In[62]:= (% /. {t → t_, u → u_, x → x_, y → y_}) /. Equal → SetDelayed
```

The **thinpart** wrapper can now be removed.

```
In[63]:= 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[63]= 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

In[64]:= (% /. {t → t_, u → u_, x → x_, y → y_}) /. Equal → SetDelayed
```

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.

```
In[65]:= 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[65]= 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

In[66]:= (% /. {t → t_, u → u_, v → v_, w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed

In[67]:= and[equal[t, composite[x, history[u, y]]],
           equal[v, APPLY[x, PAIR[z, composite[u, id[image[y, singleton[z]]]]]]],
           equal[V, domain[VERTSECT[y]]], not[equal[v, APPLY[t, z]]]] // NotNotTest

Out[67]= and[equal[t, composite[x, history[u, y]]],
           equal[v, APPLY[x, PAIR[z, composite[u, id[image[y, singleton[z]]]]]]],
           equal[V, domain[VERTSECT[y]]], not[equal[v, APPLY[t, z]]]] == False

In[68]:= (% /. {t → t_, u → u_, v → v_, w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed

In[69]:= equality = False;

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

In[71]:= (% /. {t → t_, u → u_, v → v_, w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

```
In[72]:= implies[and[equal[u, union[w, cart[singleton[z], singleton[v]]]],
           equal[v, APPLY[t, z]]],
         equal[u, union[w, cart[singleton[z], singleton[APPLY[t, z]]]]]] // NotNotTest

Out[72]= or[equal[u, union[w, cart[singleton[z], singleton[APPLY[t, z]]]]],
         not[equal[u, union[w, cart[singleton[z], singleton[v]]]]],
         not[equal[v, APPLY[t, z]]]] == True

In[73]:= (% /. {t → t_, u → u_, v → v_, w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

This can be reformulated so that **t** only occurs in its definition and the literal **v = APPLY[t,z]**.

```
In[74]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p2, p3], p4],
         not[implies[and[p1, p3], p4]], {p1 → and[equal[v, APPLY[t, z]],
           equal[u, union[w, cart[singleton[z], singleton[v]]]]],
          p2 -> equal[u, union[w, cart[singleton[z], singleton[APPLY[t, z]]]]],
          p3 → and[equal[t, composite[x, history[u, y]]],
           equal[V, domain[VERTSECT[y]]], equal[cart[V, V], domain[x]],
           FUNCTION[x], member[w, partrec[x, y]], member[z, V],
           subclass[image[y, singleton[z]], domain[w]]],
          p4 -> member[z, domain[rec[x, y]]]}]]

Out[74]= or[member[z, domain[rec[x, y]]], not[equal[t, composite[x, history[u, y]]]],
         not[equal[u, union[w, cart[singleton[z], singleton[v]]]]],
         not[equal[v, APPLY[t, z]]], not[equal[V, domain[VERTSECT[y]]]],
         not[equal[cart[V, V], domain[x]]], not[FUNCTION[x]],
         not[member[w, partrec[x, y]]], not[member[z, V]],
         not[subclass[image[y, singleton[z]], domain[w]]]] == True

In[75]:= (% /. {t → t_, u → u_, v → v_, w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

At this point the variable **t** is eliminated, with the **equality** flag still off.

```
In[76]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
           implies[and[p1, p3, p4], p5], not[implies[and[p1, p2, p4], p5]],
           {p1 → and[thin[y], equal[t, composite[x, history[u, y]]]], p2 ->
             equal[v, APPLY[x, PAIR[z, composite[u, id[image[y, singleton[z]]]]]]],
            p3 → equal[v, APPLY[t, z]],
            p4 → and[equal[u, union[w, cart[singleton[z], singleton[v]]]],
              equal[cart[V, V], domain[x]], FUNCTION[x], member[w, partrec[x, y]],
              member[z, V], subclass[image[y, singleton[z]], domain[w]]],
            p5 -> member[z, domain[rec[x, y]]]}]] /. t -> composite[x, history[u, y]]

Out[76]= or[member[z, domain[rec[x, y]]],
          not[equal[u, union[w, cart[singleton[z], singleton[v]]]]],
          not[equal[v, APPLY[x, PAIR[z, composite[u, id[image[y, singleton[z]]]]]]]],
          not[equal[V, domain[VERTSECT[y]]]], not[equal[cart[V, V], domain[x]]],
          not[FUNCTION[x]], not[member[w, partrec[x, y]]], not[member[z, V]],
          not[subclass[image[y, singleton[z]], domain[w]]]] == True

In[77]:= (% /. {u → u_, v → v_, w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

---

the elimination of u and v

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

```
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;
```

```
In[82]:= implies[
           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]]]]]]],
           equal[v, APPLY[x, PAIR[z, composite[u, id[image[y, singleton[z]]]]]]] //
          NotNotTest

Out[82]= or[equal[v, APPLY[x, PAIR[z, composite[u, id[image[y, singleton[z]]]]]]],
           not[equal[0, fix[y]]],
           not[equal[u, union[w, cart[singleton[z], singleton[v]]]]], not[equal[v,
             APPLY[x, PAIR[z, composite[w, id[image[y, singleton[z]]]]]]]] == True

In[83]:= (% /. {u → u_, v → v_, w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

One can now eliminate the variable **u**, and then the variable **v**, one after the other (but not simultaneously).

```
In[84]:= (Map[not, SubstTest[and, implies[and[p1, p2], p3],
              implies[and[p1, p3, p4], p5], not[implies[and[p1, p2, p4], p5]],
              {p1 → equal[u, union[w, cart[singleton[z], singleton[v]]]],
               p2 → and[equal[0, fix[y]], equal[v,
                  APPLY[x, PAIR[z, composite[w, id[image[y, singleton[z]]]]]]]],
               p3 -> equal[v, APPLY[x, PAIR[z, composite[u,
                    id[image[y, singleton[z]]]]]]],
               p4 → and[thin[y], equal[cart[V, V], domain[x]], FUNCTION[x],
                 member[w, partrec[x, y]], member[z, V],
                 subclass[image[y, singleton[z]], domain[w]]],
               p5 -> member[z, domain[rec[x, y]]]}]] /.
            u -> union[w, cart[singleton[z], singleton[v]]]) /.
          v -> APPLY[x, PAIR[z, composite[w, id[image[y, singleton[z]]]]]]

Out[84]= or[member[z, domain[rec[x, y]]], not[equal[0, fix[y]]],
           not[equal[V, domain[VERTSECT[y]]]], not[equal[cart[V, V], domain[x]]],
           not[FUNCTION[x]], not[member[w, partrec[x, y]]], not[member[z, V]],
           not[subclass[image[y, singleton[z]], domain[w]]]] == True

In[85]:= (% /. {w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

---

eliminating the partial solution w

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 **rec[x,y]**.

```
In[86]:= SubstTest[implies, and[WELLFOUNDED[u], thin[inverse[u]], subvariant[u, v]],
          equal[0, v], {u → inverse[y], v → complement[domain[rec[x, y]]]}]

Out[86]= or[equal[V, domain[rec[x, y]]],
           not[equal[V, domain[VERTSECT[y]]]], not[equal[V, union[domain[rec[x, y]],
              image[inverse[y], complement[domain[rec[x, y]]]]]]],
           not[WELLFOUNDED[inverse[y]]]] == True

In[87]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Recall that the requisite subvariance condition can be derived by showing that **subclass[image[y,singleton[z]], domain[rec[x,y]]]** implies that **member[z, domain[rec[x,y]]]**. The first step is to replace **y** by its transitive closure **trv[y]** to obtain a set that is invariant under **y**.

```
In[88]:= equality = True;

In[89]:= SubstTest[implies, and[subclass[image[y, singleton[z]], t], invariant[y, t]],
          subclass[image[trv[y], singleton[z]], t], t → domain[rec[x, y]]]

Out[89]= or[not[subclass[image[y, singleton[z]], domain[rec[x, y]]]],
           subclass[image[trv[y], singleton[z]], domain[rec[x, y]]]] == True

In[90]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

If **y** is thin, so is its transitive closure, so **image[trv[y],singleton[z]]** is a set.

```
In[91]:= SubstTest[implies, thin[t], member[image[t, singleton[z]], V], t → trv[y]]

Out[91]= or[member[image[trv[y], singleton[z]], V],
           not[equal[V, domain[VERTSECT[y]]]]] == True

In[92]:= or[member[image[trv[y_], singleton[z_]], V],
           not[equal[V, domain[VERTSECT[y_]]]]] := True
```

When **x** is a function and **y** is a thin relation whose inverse is well-founded, any restriction of **rec[x,y]** to a set that is invariant under **y** is a partial solution.

```
In[93]:= SubstTest[implies,
          and[FUNCTION[x], thin[y], WELLFOUNDED[inverse[y]], member[w, invar[y]]],
          member[composite[rec[x, y], id[w]], partrec[x, y]],
          w → image[trv[y], singleton[z]]]

Out[93]= or[member[composite[rec[x, y], id[image[trv[y], singleton[z]]]],
             partrec[x, y]], not[equal[V, domain[VERTSECT[y]]]],
           not[FUNCTION[x]], not[member[image[trv[y], singleton[z]], V]],
           not[WELLFOUNDED[inverse[y]]]] == True
```

```
In[94]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

This can be simplified:

```
In[95]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[and[p1, p2], p3], not[implies[p1, p3]],
           {p1 -> and[FUNCTION[x], thin[y], WELLFOUNDED[inverse[y]]],
            p2 -> member[image[trv[y], singleton[z]], V], p3 -> member[
              composite[rec[x, y], id[image[trv[y], singleton[z]]]], partrec[x, y]]}]]

Out[95]= or[member[composite[rec[x, y], id[image[trv[y], singleton[z]]]],
            partrec[x, y]], not[equal[V, domain[VERTSECT[y]]]],
           not[FUNCTION[x]], not[WELLFOUNDED[inverse[y]]]] == True

In[96]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

One can introduce the abbreviation **w** for this partial solution.

```
In[97]:= and[equal[V, domain[VERTSECT[y]]],
           equal[w, composite[rec[x, y], id[image[trv[y], singleton[z]]]]],
           FUNCTION[x], not[member[w, partrec[x, y]]],
           WELLFOUNDED[inverse[y]]] // NotNotTest

Out[97]= and[equal[V, domain[VERTSECT[y]]],
           equal[w, composite[rec[x, y], id[image[trv[y], singleton[z]]]]],
           FUNCTION[x], not[member[w, partrec[x, y]]], WELLFOUNDED[inverse[y]]] == False

In[98]:= (% /. {w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed

In[99]:= equality = False;

In[100]:=
       implies[and[FUNCTION[x], thin[y], WELLFOUNDED[inverse[y]],
          equal[w, composite[rec[x, y], id[image[trv[y], singleton[z]]]]],
          member[w, partrec[x, y]]] // NotNotTest

Out[100]=
       or[member[w, partrec[x, y]], not[equal[V, domain[VERTSECT[y]]]],
         not[equal[w, composite[rec[x, y], id[image[trv[y], singleton[z]]]]],
         not[FUNCTION[x]], not[WELLFOUNDED[inverse[y]]]] == True

In[101]:=
       (% /. {w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Note that the condition **subclass[image[y,singleton[z]], domain[w]]** holds for this partial solution.

```
In[102]:=
       equality = True;
```

In[103]:=
```
and[equal[w, composite[rec[x, y], id[image[trv[y], singleton[z]]]]],
  not[subclass[image[y, singleton[z]], domain[w]]],
  subclass[image[y, singleton[z]], domain[rec[x, y]]]] // NotNotTest
```

Out[103]=
```
and[equal[w, composite[rec[x, y], id[image[trv[y], singleton[z]]]]],
  not[subclass[image[y, singleton[z]], domain[w]]],
  subclass[image[y, singleton[z]], domain[rec[x, y]]]] == False
```

In[104]:=
```
(% /. {w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

In[105]:=
```
equality = False;
```

In[106]:=
```
implies[and[subclass[image[y, singleton[z]], domain[rec[x, y]]],
   equal[w, composite[rec[x, y], id[image[trv[y], singleton[z]]]]]],
  subclass[image[y, singleton[z]], domain[w]]] // NotNotTest
```

Out[106]=
```
or[not[equal[w, composite[rec[x, y], id[image[trv[y], singleton[z]]]]]],
  not[subclass[image[y, singleton[z]], domain[rec[x, y]]]],
  subclass[image[y, singleton[z]], domain[w]]] == True
```

In[107]:=
```
(% /. {w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

It is also convenient to dispose of the condition **fix[y] = 0** now.

In[108]:=
```
SubstTest[implies, WELLFOUNDED[t], equal[0, fix[t]], t → inverse[y]]
```

Out[108]=
```
or[equal[0, fix[y]], not[WELLFOUNDED[inverse[y]]]] == True
```

In[109]:=
```
or[equal[0, fix[y_]], not[WELLFOUNDED[inverse[y_]]]] := True
```

At this point one can eliminate the variable **w**.

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 → and[FUNCTION[x],
      thin[y], WELLFOUNDED[inverse[y]], equal[cart[V, V], domain[x]]],
    p2 → equal[w, composite[rec[x, y], id[image[trv[y], singleton[z]]]]],
    p3 → member[z, V],
    p4 -> subclass[image[y, singleton[z]], domain[rec[x, y]]],
    p5 → member[w, partrec[x, y]],
    p6 -> subclass[image[y, singleton[z]], domain[w]],
    p7 → equal[0, fix[y]], p8 -> member[z, domain[rec[x, y]]]}]] /.
  w -> 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]:=
```
(% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

---

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 **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[VERTSECT[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[VERTSECT[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 **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[VERTSECT[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[VERTSECT[y_]]]], not[equal[cart[V, V], domain[x_]]],
  not[FUNCTION[x_]], not[WELLFOUNDED[inverse[y_]]]] := True
```