Well−founded recursion, part 3. Restrictions of partial solutions.

Johan G. F. Belinfante
2004 August 26

```
In[1]:= << goedel60.23b; << tools.m;

        :Package Title: goedel60.23b          2004 August 23 at 9:05 a.m.

        It is now:  2004 Aug 26 at 12:1

        Loading Simplification Rules

        TOOLS.M                      Revised 2004 August 11

        weightlimit = 40
```

summary

This third notebook on well−founded recursion is concerned with restrictions of partial solutions. It may be recalled that to avoid automatic expansion of its (fairly complicated) definition, the class **partrec[x,y]** of partial solutions had been defined by a membership rule wrapped with **class**. It was later shown in the notebook **partrec.nb** that an unwrapped version of the membership rule for the class **partrec[x,y]** of partial solutions can be made available via the following characterization:

```
In[2]:= equiv[member[w, partrec[x, y]], and[member[w, V], invariant[y, domain[w]],
            subclass[w,
             composite[x, id[composite[IMAGE[composite[id[w], inverse[FIRST]]], VERTSECT[y]]],
              inverse[FIRST]]]]] // not // not

Out[2]= True
```

Because it is convenient in this notebook to omit the set−hood condition **member[w,V]**, the following temporary abbreviation is introduced here to help make it easier to explain the results obtained:

```
In[3]:= partialsolution[w_, x_, y_] := and[invariant[y, domain[w]],
          subclass[w,
           composite[x, id[composite[IMAGE[composite[id[w], inverse[FIRST]]], VERTSECT[y]]],
            inverse[FIRST]]]]
```

It will be shown in this notebook that the restriction of any partial solution to any class **z** that is invariant under **y** is another partial solution. A technical feature of interest is that the methods used in this notebook allow this result to be derived without any restrictions on the classes **x** and **y**. Since it is not assumed that **x** is a function, the partial solutions **w** need not be functions, and therefore function application can not play any role in the derivations. The derivations presented in this notebook do not use **APPLY**, but instead work directly with the recursion relation as formulated above. A series of technical lemmas is derived to make it possible to do so.

In the special case that **x** is a function, and **y** is a thin relation whose inverse is well−founded, the class **partrec[x,y]** of partial solutions and the total solution **rec[x,y]** of the recursion problem each determines the other. Not only is the the total solution equal to the union of the class of partial solutions, but in this case, the class of partial solutions is the class of all restrictions of the total solution to sets which are invariant under **y**.

the invariance condition

The first lemma is a consequence of the associative law for composition and the fact that any subclass of the function **VERTSECT[y]** is a restriction:

```
In[4]:= Map[subclass[#, composite[IMAGE[id[z]], VERTSECT[y]]] &,
          Assoc[IMAGE[id[z]], id[P[z]], VERTSECT[y]]] // Reverse

Out[4]= subclass[composite[VERTSECT[y], id[complement[image[inverse[y], complement[z]]]]],
          composite[IMAGE[id[z]], VERTSECT[y]]] == True
```

This result is made a temporary rewrite rule.

```
In[5]:= (% /. {y → y_, z → z_}) /. Equal → SetDelayed
```

From this one obtains:

```
In[6]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
          {u -> composite[VERTSECT[y], id[z]],
           v -> composite[id[P[z]], VERTSECT[y]], w -> composite[IMAGE[id[z]], VERTSECT[y]]}]

Out[6]= or[not[subclass[image[y, intersection[z, domain[VERTSECT[y]]]], z]],
          subclass[composite[VERTSECT[y], id[z]], composite[IMAGE[id[z]], VERTSECT[y]]] == True
```

```
In[7]:= (% /. {y → y_, z → z_}) /. Equal → SetDelayed
```

This result can be cleaned up:

```
In[8]:= Map[not, SubstTest[and, implies[p1, p2],
          implies[p2, p3], not[implies[p1, p3]], {p1 → invariant[y, z],
           p2 -> subclass[image[y, intersection[z, domain[VERTSECT[y]]]], z],
           p3 -> subclass[composite[VERTSECT[y], id[z]],
             composite[IMAGE[id[z]], VERTSECT[y]]]}]]

Out[8]= or[not[subclass[image[y, z], z]],
          subclass[composite[VERTSECT[y], id[z]], composite[IMAGE[id[z]], VERTSECT[y]]] == True
```

```
In[9]:= or[not[subclass[image[y_, z_], z_]], subclass[
           composite[VERTSECT[y_], id[z_]], composite[IMAGE[id[z_]], VERTSECT[y_]]]] := True
```

restrictions of restrictions

The function that produces restrictions of a given relation **w** is

```
In[10]:= lambda[t, composite[w, id[t]]]

Out[10]= IMAGE[composite[id[w], inverse[FIRST]]]
```

The function that produces restrictions of a given restriction of **w** can be expressed in terms of this:

```
In[11]:= composite[IMAGE[composite[id[w], inverse[FIRST]]], IMAGE[id[z]]] // VSNormality

Out[11]= composite[IMAGE[composite[id[w], inverse[FIRST]]], IMAGE[id[z]]] ==
          IMAGE[composite[id[composite[w, id[z]]], inverse[FIRST]]]
```

```
In[12]:= composite[IMAGE[composite[id[w_], inverse[FIRST]]], IMAGE[id[z_]]] :=
          IMAGE[composite[id[composite[w, id[z]]], inverse[FIRST]]]
```

combining the results of the preceding sections

Each of the formulas derived in the preceding two sections involved **IMAGE[id[z]]**. This function can be eliminated by combining the results as follows:

```
In[13]:= Map[not, SubstTest[and, implies[p1, p2],
            implies[p2, p3], not[implies[p1, p3]], {p1 → invariant[y, z], p2 ->
              subclass[composite[VERTSECT[y], id[z]], composite[IMAGE[id[z]], VERTSECT[y]]],
             p3 -> subclass[composite[x, VERTSECT[y], id[z]],
               composite[x, IMAGE[id[z]], VERTSECT[y]]]}]] /.
          x → IMAGE[composite[id[w], inverse[FIRST]]]

Out[13]= or[not[subclass[image[y, z], z]], subclass[
            composite[IMAGE[composite[id[w], inverse[FIRST]]], VERTSECT[y], id[z]], composite[
             IMAGE[composite[id[composite[w, id[z]]], inverse[FIRST]]], VERTSECT[y]]]] == True

In[14]:= (% /. {w → w_, y → y_, z → z_}) /. Equal → SetDelayed
```

The next step is to wrap **composite[x, id[ ], inverse[FIRST]]** around the inclusion in the conclusion of the above result, which is possible because i**d[ ]** and **composite** are monotone constructors:

```
In[15]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
           not[implies[p1, p3]], {p1 → invariant[y, z], p2 -> subclass[composite[
               IMAGE[composite[id[w], inverse[FIRST]]], VERTSECT[y], id[z]], composite[
               IMAGE[composite[id[composite[w, id[z]]], inverse[FIRST]]], VERTSECT[y]]],
            p3 → subclass[composite[x, id[composite[IMAGE[composite[id[w], inverse[FIRST]]],
                  VERTSECT[y], id[z]]], inverse[FIRST]],
              composite[x, id[composite[IMAGE[composite[id[composite[w, id[z]]],
                   inverse[FIRST]]], VERTSECT[y]]], inverse[FIRST]]]}]]

Out[15]= or[not[subclass[image[y, z], z]],
           subclass[composite[x, id[composite[IMAGE[composite[id[w], inverse[FIRST]]],
               VERTSECT[y], id[z]]], inverse[FIRST]], composite[x,
             id[composite[IMAGE[composite[id[composite[w, id[z]]], inverse[FIRST]]],
               VERTSECT[y]]], inverse[FIRST]]]] == True

In[16]:= (% /. {w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

The recursion condition has the form **subclass[w, t]**, from which one can deduce that a restriction of **w** is contained in the corresponding restriction of **t**. The following rewrite rule applies to the latter statement:

```
In[17]:= subclass[composite[w, id[z]], composite[t, id[z]]]

Out[17]= subclass[composite[w, id[z]], t]
```

To avoid losing the factor **id[z]** on the right hand side via the above rewrite rule, one needs the following lemma:

```
In[18]:= Map[implies[subclass[w,
           composite[x, id[composite[IMAGE[composite[id[w], inverse[FIRST]]], VERTSECT[y]]],
            inverse[FIRST]]], #] &,
         SubstTest[subclass, composite[w, id[z]], composite[t, id[z]],
          t -> composite[x, id[composite[
            IMAGE[composite[id[w], inverse[FIRST]]], VERTSECT[y]]], inverse[FIRST]]]]

Out[18]= or[not[subclass[w,
           composite[x, id[composite[IMAGE[composite[id[w], inverse[FIRST]]], VERTSECT[y]]],
            inverse[FIRST]]]], subclass[composite[w, id[z]], composite[x,
           id[composite[IMAGE[composite[id[w], inverse[FIRST]]], VERTSECT[y], id[z]]],
           inverse[FIRST]]]] == True

In[19]:= (% /. {w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

the main result

It follows that a restriction of a partial solution to a class **z** that is invariant under **y** satisfies the recursion condition:

```
In[20]:= Map[not, SubstTest[and, implies[p1, p4], implies[p2, p3],
           implies[and[p3, p4], p5], not[implies[and[p1, p2], p5]],
           {p1 → invariant[y, z], p2 -> subclass[w, composite[x, id[composite[
                 IMAGE[composite[id[w], inverse[FIRST]]], VERTSECT[y]]], inverse[FIRST]]],
            p3 -> subclass[composite[w, id[z]], composite[x, id[composite[IMAGE[
                  composite[id[w], inverse[FIRST]]], VERTSECT[y], id[z]]], inverse[FIRST]]],
            p4 -> subclass[composite[x, id[composite[IMAGE[composite[id[w], inverse[FIRST]]],
                 VERTSECT[y], id[z]]], inverse[FIRST]], composite[x,
              id[composite[IMAGE[composite[id[composite[w, id[z]]], inverse[FIRST]]],
                 VERTSECT[y]]], inverse[FIRST]]],
            p5 -> subclass[composite[w, id[z]], composite[x,
              id[composite[IMAGE[composite[id[composite[w, id[z]]], inverse[FIRST]]],
                 VERTSECT[y]]], inverse[FIRST]]]}]]

Out[20]= or[not[subclass[w,
           composite[x, id[composite[IMAGE[composite[id[w], inverse[FIRST]]], VERTSECT[y]]],
            inverse[FIRST]]]],
         not[subclass[image[y, z], z]], subclass[composite[w, id[z]], composite[x,
           id[composite[IMAGE[composite[id[composite[w, id[z]]], inverse[FIRST]]],
             VERTSECT[y]]], inverse[FIRST]]]] == True

In[21]:= or[not[subclass[image[y_, z_], z_]], not[subclass[w_, composite[x_,
           id[composite[IMAGE[composite[id[w_], inverse[FIRST]]], VERTSECT[y_]]],
           inverse[FIRST]]]], subclass[composite[w_, id[z_]], composite[x_,
          id[composite[IMAGE[composite[id[composite[w_, id[z_]]], inverse[FIRST]]],
            VERTSECT[y_]]], inverse[FIRST]]]] := True
```

Restatement:

```
In[22]:= implies[and[partialsolution[w, x, y], invariant[y, z]], subclass[composite[w, id[z]],
           composite[x, id[composite[IMAGE[composite[id[composite[w, id[z]]], inverse[FIRST]]],
             VERTSECT[y]]], inverse[FIRST]]]]

Out[22]= True
```

To conclude that **composite[w,id[z]]** is a partial solution, one only needs to show that its domain is invariant under **y**, and this can be done simply by using a double negation:

```
In[23]:= implies[and[partialsolution[w, x, y], invariant[y, z]],
            partialsolution[composite[w, id[z]], x, y]] // not // not

Out[23]= True
```

a corollary

The main result obtained above does not require that the partial solution **w** be a set. In this section, a corollary is derived by adding the condition that **w** be a set:

```
In[24]:= Map[not, SubstTest[and, implies[p2, p3], implies[and[p1, p3], p4],
           not[implies[and[p1, p2], p4]], {p1 → invariant[y, z],
            p2 → member[w, partrec[x, y]], p3 -> subclass[w, composite[x, id[composite[
                 IMAGE[composite[id[w], inverse[FIRST]]], VERTSECT[y]]], inverse[FIRST]]],
            p4 -> subclass[composite[w, id[z]], composite[x,
               id[composite[IMAGE[composite[id[composite[w, id[z]]], inverse[FIRST]]],
                 VERTSECT[y]]], inverse[FIRST]]]}]]

Out[24]= or[not[member[w, partrec[x, y]]],
           not[subclass[image[y, z], z]], subclass[composite[w, id[z]], composite[x,
             id[composite[IMAGE[composite[id[composite[w, id[z]]], inverse[FIRST]]],
               VERTSECT[y]]], inverse[FIRST]]]] == True

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

The following result spells out what one needs in order to conclude that **composite[w, id[z]]** belong to **partrec[x,y]**.

```
In[26]:= SubstTest[implies,
          and[member[t, V], invariant[y, domain[t]], subclass[t, composite[x, id[composite[
               IMAGE[composite[id[t], inverse[FIRST]]], VERTSECT[y]]], inverse[FIRST]]]],
          member[t, partrec[x, y]], t → composite[w, id[z]]]

Out[26]= or[member[composite[w, id[z]], partrec[x, y]],
           not[member[image[w, z], V]], not[member[intersection[z, domain[w]], V]],
           not[subclass[composite[w, id[z]], composite[x, id[composite[
                IMAGE[composite[id[composite[w, id[z]]], inverse[FIRST]]], VERTSECT[y]]],
              inverse[FIRST]]]], not[subclass[image[y, intersection[z, domain[w]]], z]],
           not[subclass[image[y, intersection[z, domain[w]]], domain[w]]]] == True

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

All these conditions can be deduced:

```
In[28]:= Map[not, SubstTest[and, implies[p1, p3], implies[p3, p4],
           implies[p2, p5], implies[and[p1, p2], p6], implies[p1, p7], implies[p1, p8],
           implies[and[p4, p5, p6, p7, p8], p9], not[implies[and[p1, p2], p9]],
           {p1 → member[w, partrec[x, y]], p2 → invariant[y, z], p3 → invariant[y, domain[w]],
            p4 -> subclass[image[y, intersection[z, domain[w]]], domain[w]],
            p5 -> subclass[image[y, intersection[z, domain[w]]], z],
            p6 -> subclass[composite[w, id[z]], composite[x, id[composite[IMAGE[composite[
                   id[composite[w, id[z]]], inverse[FIRST]]], VERTSECT[y]]], inverse[FIRST]]],
            p7 -> member[image[w, z], V], p8 → member[intersection[z, domain[w]], V],
            p9 -> member[composite[w, id[z]], partrec[x, y]]}]]

Out[28]= or[member[composite[w, id[z]], partrec[x, y]],
           not[member[w, partrec[x, y]]], not[subclass[image[y, z], z]]] == True

In[29]:= or[member[composite[w_, id[z_]], partrec[x_, y_]],
           not[member[w_, partrec[x_, y_]]], not[subclass[image[y_, z_], z_]]] := True
```

The variable **w** can be eliminated:

```
In[30]:= Map[equal[V, #] &, SubstTest[class, w, or[member[composite[w, id[z]], p],
           not[member[w, p]], not[subclass[image[y, z], z]]], p → partrec[x, y]]] // Reverse

Out[30]= or[not[subclass[image[y, z], z]],
          subclass[image[IMAGE[id[cart[z, V]]], partrec[x, y]], partrec[x, y]]] == True

In[31]:= or[not[subclass[image[y_, z_], z_]],
          subclass[image[IMAGE[id[cart[z_, V]]], partrec[x_, y_]], partrec[x_, y_]]] := True
```

The result of this section does not appear to provide any new facts about the total solution **rec[x,y] = U[partrec[x,y]]**, but only says something about the class of partial solutions.

a more significant corollary

The remainder of this notebook is devoted to a special consequence that holds when **x** is a function, and **y** is a thin relation whose inverse is well−founded. It will be shown that for this special case, the class of partial solutions and the total solution of the recursion problem each determines the other. In this case, it turns out that not only can the the total solution be obtained as the union of the class of partial solutions, but also the class of partial solutions can be recovered as the class of all restrictions of the total solution to sets which are invariant under **y**. The entire derivation is rather long, and will be broken up into two inclusions. At the end these two inclusions are combined into an equation. In this initial section, the main result is derived with an additional variable **z**, representing any set that is invariant under **y**. The variable **z** will be eliminated in yhe next section.

Lemma.

```
In[32]:= SubstTest[implies, invariant[y, t],
          subclass[image[y, intersection[z, t]], t], t → domain[rec[x, y]]]

Out[32]= subclass[image[y, intersection[z, domain[rec[x, y]]]], domain[rec[x, y]]] == True

In[33]:= subclass[image[y_, intersection[z_, domain[rec[x_, y_]]]], domain[rec[x_, y_]]] := True
```

Lemma.

```
In[34]:= SubstTest[implies,
          and[member[t, V], invariant[y, domain[t]], subclass[t, composite[x, id[composite[
               IMAGE[composite[id[t], inverse[FIRST]]], VERTSECT[y]]], inverse[FIRST]]]],
          member[t, partrec[x, y]], t → composite[rec[x, y], id[z]]]

Out[34]= or[member[composite[rec[x, y], id[z]], partrec[x, y]],
           not[member[image[rec[x, y], z], V]],
           not[member[intersection[z, domain[rec[x, y]]], V]],
           not[subclass[composite[rec[x, y], id[z]], composite[x,
              id[composite[IMAGE[composite[id[composite[rec[x, y], id[z]]], inverse[FIRST]]],
                VERTSECT[y]]], inverse[FIRST]]]],
           not[subclass[image[y, intersection[z, domain[rec[x, y]]]], z]]] == True

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

The following takes a rather long time:

```
In[36]:= Map[not, SubstTest[and, implies[and[p1, p3], p5], implies[p2, p3],
           implies[p2, p4], implies[and[p1, p4], p6], implies[p1, p7], implies[p1, p8],
           implies[and[p5, p6, p7, p8], p9], not[implies[and[p1, p2], p9]],
           {p1 → member[z, invar[y]],
            p2 → and[FUNCTION[x], thin[y], WELLFOUNDED[inverse[y]]],
            p3 → subclass[rec[x, y], composite[x,
               id[composite[IMAGE[composite[id[rec[x, y]], inverse[FIRST]]], VERTSECT[y]]],
               inverse[FIRST]]],
            p4 → FUNCTION[rec[x, y]],
            p5 –> subclass[composite[rec[x, y], id[z]], composite[x,
               id[composite[IMAGE[composite[id[composite[rec[x, y], id[z]]], inverse[FIRST]]],
                 VERTSECT[y]]], inverse[FIRST]]],
            p6 –> member[image[rec[x, y], z], V],
            p7 –> member[intersection[z, domain[rec[x, y]]], V],
            p8 –> subclass[image[y, intersection[z, domain[rec[x, y]]]], z],
            p9 –> member[composite[rec[x, y], id[z]], partrec[x, y]]}]]

Out[36]= or[member[composite[rec[x, y], id[z]], partrec[x, y]],
           not[equal[V, domain[VERTSECT[y]]]], not[FUNCTION[x]], not[member[z, V]],
           not[subclass[image[y, z], z]], not[WELLFOUNDED[inverse[y]]]] == True

In[37]:= or[member[composite[rec[x_, y_], id[z_]], partrec[x_, y_]],
           not[equal[V, domain[VERTSECT[y_]]]], not[FUNCTION[x_]], not[member[z_, V]],
           not[subclass[image[y_, z_], z_]], not[WELLFOUNDED[inverse[y_]]]] := True
```

---

eliminating z

```
In[38]:= image[inverse[IMAGE[composite[id[x], inverse[FIRST]]]], y] // Normality // Reverse

Out[38]= fix[composite[S, IMAGE[FIRST], id[intersection[y, P[composite[Id, x]]]],
            S, IMAGE[composite[id[x], inverse[FIRST]]]]] ==
          image[inverse[IMAGE[composite[id[x], inverse[FIRST]]]], y]

In[39]:= fix[composite[S, IMAGE[FIRST], id[intersection[y_, P[composite[Id, x_]]]],
            S, IMAGE[composite[id[x_], inverse[FIRST]]]]] :=
          image[inverse[IMAGE[composite[id[x], inverse[FIRST]]]], y]

In[40]:= Map[equal[V, #] &,
           SubstTest[class, z, or[member[composite[r, id[z]], p], not[equal[V, v]],
             not[subclass[P[x], u]], not[member[z, V]], not[subclass[image[y, z], z]],
             not[subclass[P[inverse[y]], w]]], {r → rec[x, y], p → partrec[x, y],
             u → FUNS, v → domain[VERTSECT[y]], w → WF}]] // Reverse

Out[40]= or[not[equal[V, domain[VERTSECT[y]]]], not[FUNCTION[x]],
           not[WELLFOUNDED[inverse[y]]], subclass[invar[y], image[
             inverse[IMAGE[composite[id[rec[x, y]], inverse[FIRST]]]], partrec[x, y]]]] == True

In[41]:= (% /. {x → x_, y → y}) /. Equal → SetDelayed

In[42]:= SubstTest[implies, equal[t, funpart[v]],
          implies[subclass[u, image[inverse[IMAGE[composite[id[t], inverse[FIRST]]]], w]],
           subclass[image[IMAGE[composite[id[t], inverse[FIRST]]], u], w]], t → v]

Out[42]= or[not[FUNCTION[v]],
           not[subclass[u, image[inverse[IMAGE[composite[id[v], inverse[FIRST]]]], w]]],
           subclass[image[IMAGE[composite[id[v], inverse[FIRST]]], u], w]] == True

In[43]:= (% /. {u → u_, v → v_, w → w_}) /. Equal → SetDelayed
```

```
In[44]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p2, p3], p4],
           not[implies[p1, p4]], {p1 → and[FUNCTION[x], thin[y], WELLFOUNDED[inverse[y]]],
            p2 → FUNCTION[rec[x, y]], p3 -> subclass[invar[y],
              image[inverse[IMAGE[composite[id[rec[x, y]], inverse[FIRST]]]], partrec[x, y]]],
            p4 → subclass[image[IMAGE[composite[id[rec[x, y]], inverse[FIRST]]], invar[y]],
              partrec[x, y]]}]]

Out[44]= or[not[equal[V, domain[VERTSECT[y]]]], not[FUNCTION[x]], not[WELLFOUNDED[inverse[y]]],
           subclass[image[IMAGE[composite[id[rec[x, y]], inverse[FIRST]]], invar[y]],
            partrec[x, y]]] == True

In[45]:= (% /. {u → u_, v → v_, w → w_}) /. Equal → SetDelayed
```

the reverse inclusion

In this section it is shown that the reverse inclusion also holds; under these conditions, every partial solution is a restriction of the total solution.

```
In[47]:= SubstTest[implies, and[member[pair[u, v], composite[Id, z]], member[u, x]],
          member[v, image[z, x]], {u → domain[v], z → IMAGE[composite[id[w], inverse[FIRST]]]}]

Out[47]= or[member[v, image[IMAGE[composite[id[w], inverse[FIRST]]], x]],
           not[equal[v, composite[w, id[domain[v]]]]],
           not[member[v, V]], not[member[domain[v], x]]] == True

In[48]:= (% /. {v → v_, w → w_, x → x_}) /. Equal → SetDelayed

In[49]:= SubstTest[implies, and[member[z, x], subclass[x, t]],
          member[z, t], t → image[inverse[IMAGE[FIRST]], y]]

Out[49]= or[member[domain[z], y], not[member[z, x]],
           not[subclass[image[IMAGE[FIRST], x], y]]] == True

In[50]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed

In[51]:= Map[not, SubstTest[and, implies[and[p1, p4], p5],
           implies[and[p2, p3], p6], implies[p3, p4], implies[p3, p7],
           implies[and[p5, p6, p7], p8], not[implies[and[p1, p2, p3], p8]],
           {p1 → FUNCTION[U[x]], p2 → subclass[image[IMAGE[FIRST], x], y], p3 → member[z, x],
            p4 → subclass[z, U[x]], p5 → equal[z, composite[U[x], id[domain[z]]]],
            p6 → member[domain[z], y], p7 → member[z, V],
            p8 → member[z, image[IMAGE[composite[id[U[x]], inverse[FIRST]]], y]]}]]

Out[51]= or[member[z, image[IMAGE[composite[id[U[x]], inverse[FIRST]]], y]],
           not[FUNCTION[U[x]]], not[member[z, x]],
           not[subclass[image[IMAGE[FIRST], x], y]]] == True

In[52]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed

In[53]:= Map[equal[V, #] &, SubstTest[class, z,
            or[member[z, u], not[subclass[P[U[x]], w]], not[member[z, x]], not[subclass[v, y]]],
            {u -> image[IMAGE[composite[id[U[x]], inverse[FIRST]]], y],
             v -> image[IMAGE[FIRST], x], w → FUNS}]] // Reverse

Out[53]= or[not[FUNCTION[U[x]]], not[subclass[image[IMAGE[FIRST], x], y]],
           subclass[x, image[IMAGE[composite[id[U[x]], inverse[FIRST]]], y]]] == True

In[54]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

```
In[55]:= SubstTest[or, not[FUNCTION[U[u]]], not[subclass[image[IMAGE[FIRST], u], v]],
          subclass[u, image[IMAGE[composite[id[U[u]], inverse[FIRST]]], v]],
          {u → partrec[x, y], v -> invar[y]}]

Out[55]= or[not[FUNCTION[rec[x, y]]], subclass[partrec[x, y],
           image[IMAGE[composite[id[rec[x, y]], inverse[FIRST]]], invar[y]]]] == True

In[56]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed

In[57]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
            not[implies[p1, p3]], {p1 → and[FUNCTION[x], thin[y], WELLFOUNDED[inverse[y]]],
             p2 → FUNCTION[rec[x, y]], p3 -> subclass[partrec[x, y],
               image[IMAGE[composite[id[rec[x, y]], inverse[FIRST]]], invar[y]]]}]]

Out[57]= or[not[equal[V, domain[VERTSECT[y]]]], not[FUNCTION[x]],
           not[WELLFOUNDED[inverse[y]]], subclass[partrec[x, y],
            image[IMAGE[composite[id[rec[x, y]], inverse[FIRST]]], invar[y]]]] == True

In[58]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

This results can be used to sharpen the inclusion in the preceding section to an equation:

```
In[60]:= SubstTest[and, implies[p, subclass[u, v]], implies[p, subclass[v, u]],
           {p -> and[FUNCTION[x], thin[y], WELLFOUNDED[inverse[y]]], u → partrec[x, y],
            v -> image[IMAGE[composite[id[rec[x, y]], inverse[FIRST]]], invar[y]]}] // Reverse

Out[60]= or[equal[image[IMAGE[composite[id[rec[x, y]], inverse[FIRST]]], invar[y]],
            partrec[x, y]], not[equal[V, domain[VERTSECT[y]]]],
           not[FUNCTION[x]], not[WELLFOUNDED[inverse[y]]]] == True

In[61]:= or[equal[image[IMAGE[composite[id[rec[x_, y_]], inverse[FIRST]]], invar[y_]],
            partrec[x_, y_]], not[equal[V, domain[VERTSECT[y_]]]],
           not[FUNCTION[x_]], not[WELLFOUNDED[inverse[y_]]]] := True
```

This result says that when **x** is a function, and **y** is a thin relation whose inverse is well founded, the class **partrec[x,y]** of partial solutions of the recursion problem can be recovered from the total solution **rec[x,y]** as the class of all restrictions to sets which are invariant under **y**.