wellfounded recursion, part 1.

Johan G. F. Belinfante
revised 2004 August 3

```
In[1]:= SetDirectory["i:"]; << goedel60.02a; << tools.m;

        :Package Title: goedel60.02a            2004 August 2 at 11:00 p.m.

        It is now:  2004 Aug 3 at 15:41

        Loading Simplification Rules

        TOOLS.M                      Revised 2004 June 22

        weightlimit = 40
```

---

summary

This notebook is concerned with the beginnings of the theory of recursive definitions with respect to a well-founded relation. Basic information is derived about two constructors **partrec[x,y]** and **rec[x,y]** that have been introduced for this prurpose. The theory of well-founded recursion is due to A. Tarski and R. Montague. The derivation presented here follows an exposition of this on pages 65-66 of a book by Levy.

```
In[2]:= "Azriel Levy, Basic Set Theory, reprinted by Dover Publications, 2002."
Out[2]= Azriel Levy, Basic Set Theory, reprinted by Dover Publications, 2002.
```

Many of the results derived are quite time-consuming, and various tricks are used to reduce execution time. The results derived in this notebook take almost five and half minutes on an HP Pavilion a574n computer.

```
In[3]:= starttime = Date[];
```

One should think of the class **partrec[x,y]** as the collection of all partial solutions of a certain recursion condition, and the class **rec[x,y]** as the union of this class of partial solutions. The membership rules for these contructors are wrapped in **class** to avoid having the definitions from being expanded each time they are used.

```
In[4]:= Begin["Goedel`Private`"];
```

```
In[5]:= FirstMatch[class[u_, member[v_, HoldPattern[partrec[x_, y_]]]]]

Out[5]= class[u_, member[v_, partrec[x_, y_]]] := Module[{w = Unique[]},
          class[u, and[invariant[y, domain[v]], exists[w, and[equal[w,
          composite[id[v], inverse[FIRST]]], subclass[v, composite[x,
          id[composite[IMAGE[w], VERTSECT[y]]], inverse[FIRST]]]]]]]]
```

The class **partrec[x,y]** is the class of partial solutions for the generator **x** with respect to the relation **y**. In practice **x** will be a binary function with domain **cart[V,V]** and **y** a thin relation whose inverse is well-founded, although the definitions strictly speaking do not require this. For some of the first theorems these conditions are not needed, and it is convenient to state them in full generality. The total solution of the recursion is the class **rec[x,y]**, which generally is expected to be a proper class, not a set. Whether or not this is actually the case depends on the hypotheses placed on **x** and **y**. The definition itself places no restriction whatever on **x** and **y**, but the more interesting theorems about **rec[x,y]** will add various conditions.

```
In[6]:= FirstMatch[class[u_, member[v_, HoldPattern[rec[x_, y_]]]]]

Out[6]= class[u_, member[v_, rec[x_, y_]]] := Module[{w = Unique[]},
          class[u, exists[w, and[member[v, w], member[w, partrec[x, y]]]]]]
```

Note that both definitions contain explicit quantifiers. In the case of **partrec**, this is done to produce cleaner normalization formulas. For **rec**, the quantifier is unavoidable.

---

relation between partrec[x,y] and rec[x,y]

The relation between **partrec[x,y]** and **rec[x,y]** is an immediate corollary of the membership rules. The **simplify** and **cond** flags are turned off in this section to speed things up.

```
In[7]:= simplify = False; cond = False;
```

The first result obtained is the class **rec[x,y]** is the union of all partial solutions of the recursion relation.

```
In[8]:= Map[equal[0, #] &, symdif[U[partrec[x, y]], rec[x, y]] // Normality]

Out[8]= equal[rec[x, y], U[partrec[x, y]]] == True
```

```
In[9]:= U[partrec[x_, y_]] := rec[x, y]
```

The partial solutions are contained in the **rec[x,y]**.

```
In[10]:= SubstTest[implies, member[w, z], subclass[w, U[z]], z → partrec[x, y]]

Out[10]= or[not[member[w, partrec[x, y]]], subclass[w, rec[x, y]]] == True
```

```
In[11]:= or[not[member[w_, partrec[x_, y_]]], subclass[w_, rec[x_, y_]]] := True
```

The following corollary will be needed later.

```
In[12]:= Map[not, SubstTest[and, implies[p2, p3], implies[and[p1, p3], p4],
           implies[and[p1, p4], p5], not[implies[and[p1, p2], p5]],
           {p1 → member[z, domain[w]], p2 → member[w, partrec[x, y]],
            p3 → subclass[w, rec[x, y]], p4 → subclass[domain[w], domain[rec[x, y]]],
            p5 → member[z, domain[rec[x, y]]]}]]

Out[12]= or[member[z, domain[rec[x, y]]],
           not[member[w, partrec[x, y]]], not[member[z, domain[w]]]] == True

In[13]:= or[member[z_, domain[rec[x_, y_]]],
           not[member[w_, partrec[x_, y_]]], not[member[z_, domain[w_]]]] := True
```

---

rec[x,y] is a relation

Each partial solution is a relation:

```
In[14]:= or[not[member[w, partrec[x, y]]], subclass[w, cart[V, V]]] // AssertTest

Out[14]= or[not[member[w, partrec[x, y]]], subclass[w, cart[V, V]]] == True

In[15]:= or[not[member[w_, partrec[x_, y_]]], subclass[w_, cart[V, V]]] := True
```

Eliminating the variable **w** yields:

```
In[16]:= Map[equal[V, #] &,
           SubstTest[class, w, or[not[member[w, z]], subclass[w, cart[V, V]]],
            z -> partrec[x, y]]] // Reverse

Out[16]= subclass[rec[x, y], cart[V, V]] == True

In[17]:= subclass[rec[x_, y_], cart[V, V]] := True
```

---

the domain condition on partial solutions

The domain condition in the membership rule for **partrec[x,y]** is made more explicit as follows:

```
In[18]:= implies[member[w, partrec[x, y]], invariant[y, domain[w]]] // AssertTest

Out[18]= or[not[member[w, partrec[x, y]]],
           subclass[image[y, domain[w]], domain[w]]] == True
```

```
In[19]:= or[not[member[w_, partrec[x_, y_]]],
            subclass[image[y_, domain[w_]], domain[w_]]] := True
```

Eliminating the variable **w** yields:

```
In[20]:= Map[equal[V, #] &, SubstTest[class, w, implies[member[w, u], member[w, v]],
             {u → partrec[x, y], v → image[inverse[IMAGE[FIRST]], invar[y]]}]] // Reverse
Out[20]= subclass[image[IMAGE[FIRST], partrec[x, y]], invar[y]] == True

In[21]:= subclass[image[IMAGE[FIRST], partrec[x_, y_]], invar[y_]] := True
```

---

an upper bound on the domain

Applying **U** to both sides yields:

```
In[22]:= SubstTest[implies, subclass[u, v], subclass[U[u], U[v]],
           {u -> image[IMAGE[FIRST], partrec[x, y]], v -> invar[y]}]
Out[22]= subclass[domain[rec[x, y]], domain[VERTSECT[trv[y]]]] == True

In[23]:= subclass[domain[rec[x_, y_]], domain[VERTSECT[trv[y_]]]] := True
```

In practice it will be assumed that **y** is thin, in which case this imposes no limitation on **domain[rec[x,y]]**.

```
In[24]:= equal[domain[VERTSECT[trv[thinpart[y]]]], V]
Out[24]= True
```

Since the union of invariant subsets is invariant, one deduces:

```
In[25]:= SubstTest[implies, subclass[w, invar[y]],
           invariant[y, U[w]], w → image[IMAGE[FIRST], partrec[x, y]]]
Out[25]= subclass[image[y, domain[rec[x, y]]], domain[rec[x, y]]] == True

In[26]:= subclass[image[y_, domain[rec[x_, y_]]], domain[rec[x_, y_]]] := True
```

In practice this condition is not especially interesting because conditions on **x** and **y** will usually be imposed to guarantee that **domain[rec[x,y]]** is **V**.

---

a lemma that will be needed later

Lemma.

```
In[27]:= Map[not,
          SubstTest[and, implies[p1, p3], implies[p2, p4], implies[and[p3, p4], p5],
           not[implies[and[p1, p2], p5]],
           {p1 → member[z, domain[w]], p2 → member[w, partrec[x, y]],
            p3 → subclass[image[y, singleton[z]], image[y, domain[w]]],
            p4 → invariant[y, domain[w]],
            p5 -> subclass[image[y, singleton[z]], domain[w]]}]]

Out[27]= or[not[member[w, partrec[x, y]]], not[member[z, domain[w]]],
           subclass[image[y, singleton[z]], domain[w]]] == True

In[28]:= or[not[member[w_, partrec[x_, y_]]], not[member[z_, domain[w_]]],
           subclass[image[y_, singleton[z_]], domain[w_]]] := True
```

---

some trivial special cases

When the generator is the empty set, the results are trivial:

```
In[29]:= partrec[0, x] // Normality

Out[29]= partrec[0, x] == singleton[0]

In[30]:= partrec[0, x_] := singleton[0]

In[31]:= SubstTest[U, partrec[z, x], z → 0] // Reverse

Out[31]= rec[0, x] == 0

In[32]:= rec[0, x_] := 0
```

Note that the (inverse of the) empty relation is thin and well founded.

```
In[33]:= WELLFOUNDED[0]

Out[33]= True
```

When the relation **y** is empty, one has:

```
In[34]:= partrec[x, 0] // Normality

Out[34]= partrec[x, 0] == P[composite[x, RIGHT[0]]]

In[35]:= partrec[x_, 0] := P[composite[x, RIGHT[0]]]

In[36]:= SubstTest[U, partrec[x, z], z → 0] // Reverse

Out[36]= rec[x, 0] == composite[x, RIGHT[0]]

In[37]:= rec[x_, 0] := composite[x, RIGHT[0]]
```

When **x** is a binary function, then **rec[x,0]** is the function obtained by setting the second argument equal to **0**.

```
In[38]:= equal[APPLY[rec[funpart[x], 0], y], APPLY[funpart[x], PAIR[y, 0]]] // assert

Out[38]= True
```

normalization

The class **partrec[x,y]** is normalied in this section. The **cond** flag needs to be on for this section. With the **simplify** flag off, this still takes over a minute.

```
In[39]:= cond = True;

In[40]:= symdif[partrec[x, y],
            intersection[image[inverse[IMAGE[FIRST]], invar[y]], fix[composite[
               inverse[IMAGE[composite[id[inverse[FIRST]], inverse[SECOND]]]],
               UB[composite[FIRST, intersection[composite[inverse[IMG], SECOND],
                  composite[inverse[SECOND], VERTSECT[y], FIRST]],
                 intersection[composite[inverse[FIRST], FIRST],
                  composite[inverse[x], SECOND]]]]]]]] // Normality

Out[40]= union[intersection[complement[fix[composite[
              inverse[IMAGE[composite[id[inverse[FIRST]], inverse[SECOND]]]],
              UB[composite[FIRST, intersection[composite[inverse[IMG], SECOND],
                 composite[inverse[SECOND], VERTSECT[y], FIRST]],
                intersection[composite[inverse[FIRST], FIRST],
                 composite[inverse[x], SECOND]]]]]], partrec[x, y]],
           intersection[complement[image[inverse[IMAGE[FIRST]], invar[y]]],
            partrec[x, y]],
           intersection[complement[partrec[x, y]],
            fix[composite[
              inverse[IMAGE[composite[id[inverse[FIRST]], inverse[SECOND]]]],
              UB[composite[FIRST, intersection[composite[inverse[IMG], SECOND],
                 composite[inverse[SECOND], VERTSECT[y], FIRST]], intersection[
                 composite[inverse[FIRST], FIRST], composite[inverse[x], SECOND]]]]]],
            image[inverse[IMAGE[FIRST]], invar[y]]]] == 0

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

```
In[42]:= SubstTest[equal, 0, symdif[u, v], {u → partrec[x, y],
           v -> intersection[image[inverse[IMAGE[FIRST]], invar[y]], fix[composite[
               inverse[IMAGE[composite[id[inverse[FIRST]], inverse[SECOND]]]],
               UB[composite[FIRST, intersection[composite[inverse[IMG], SECOND],
                  composite[inverse[SECOND], VERTSECT[y], FIRST]],
                 intersection[composite[inverse[FIRST], FIRST],
                  composite[inverse[x], SECOND]]]]]]]}]

Out[42]= True == equal[intersection[fix[composite[
              inverse[IMAGE[composite[id[inverse[FIRST]], inverse[SECOND]]]],
              UB[composite[FIRST, intersection[composite[inverse[IMG], SECOND],
                 composite[inverse[SECOND], VERTSECT[y], FIRST]], intersection[
                 composite[inverse[FIRST], FIRST], composite[inverse[x], SECOND]]]]]],
            image[inverse[IMAGE[FIRST]], invar[y]]], partrec[x, y]]
```

This yields an explicit albeit somewhat complicated formula for **partrec[x,y]**.

```
In[43]:= intersection[fix[
            composite[inverse[IMAGE[composite[id[inverse[FIRST]], inverse[SECOND]]]],
             UB[composite[FIRST, intersection[composite[inverse[IMG], SECOND],
                composite[inverse[SECOND], VERTSECT[y_], FIRST]], intersection[
                composite[inverse[FIRST], FIRST], composite[inverse[x_], SECOND]]]]],
          image[inverse[IMAGE[FIRST]], invar[y_]]] := partrec[x, y]
```

---

corollaries of the normalization formula

Corollary:

```
In[44]:= SubstTest[subclass, intersection[u, v], u,
          {u -> fix[composite[
              inverse[IMAGE[composite[id[inverse[FIRST]], inverse[SECOND]]]],
              UB[composite[FIRST, intersection[composite[inverse[IMG], SECOND],
                 composite[inverse[SECOND], VERTSECT[y], FIRST]], intersection[
                 composite[inverse[FIRST], FIRST], composite[inverse[x], SECOND]]]]],
           v -> image[inverse[IMAGE[FIRST]], invar[y]]}]

Out[44]= subclass[partrec[x, y], fix[
            composite[inverse[IMAGE[composite[id[inverse[FIRST]], inverse[SECOND]]]],
             UB[composite[FIRST, intersection[composite[inverse[IMG], SECOND],
                composite[inverse[SECOND], VERTSECT[y], FIRST]],
               intersection[composite[inverse[FIRST], FIRST],
                composite[inverse[x], SECOND]]]]]]] == True

In[45]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Since **fix[w]** is contained in **domain[w]**, the result derived in the preceding section allows one to obtain an upper bound for **partrec[x,y]**:

```
In[46]:= SubstTest[implies, subclass[u, fix[w]],
          subclass[u, domain[w]], {u → partrec[x, y], w ->
            composite[inverse[IMAGE[composite[id[inverse[FIRST]], inverse[SECOND]]]],
             UB[composite[FIRST, intersection[composite[inverse[IMG], SECOND],
                composite[inverse[SECOND], VERTSECT[y], FIRST]], intersection[
                composite[inverse[FIRST], FIRST], composite[inverse[x], SECOND]]]]}]

Out[46]= subclass[partrec[x, y], image[
           inverse[UB[composite[FIRST, intersection[composite[inverse[IMG], SECOND],
               composite[inverse[SECOND], VERTSECT[y], FIRST]],
              intersection[composite[inverse[FIRST], FIRST],
               composite[inverse[x], SECOND]]]]], P[inverse[FIRST]]]] == True

In[47]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Applying **U** yields an upper bound for **rec[x,y]**.

```
In[48]:= SubstTest[implies, subclass[u, v], subclass[U[u], U[v]],
          {u -> partrec[x, y],
           v -> image[inverse[UB[composite[FIRST, intersection[composite[inverse[IMG],
                  SECOND], composite[inverse[SECOND], VERTSECT[y], FIRST]],
                intersection[composite[inverse[FIRST], FIRST],
                 composite[inverse[x], SECOND]]]]], P[inverse[FIRST]]]}]

Out[48]= subclass[rec[x, y],
           composite[x, id[composite[IMG, id[cart[P[inverse[FIRST]], V]],
              inverse[SECOND], VERTSECT[y]]], inverse[FIRST]]] == True

In[49]:= subclass[rec[x_, y_],
           composite[x_, id[composite[IMG, id[cart[P[inverse[FIRST]], V]],
              inverse[SECOND], VERTSECT[y_]]], inverse[FIRST]]] := True
```

In particular, this implies:

```
In[50]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
          {u → rec[x, y], v -> composite[x,
             id[composite[IMG, id[cart[P[inverse[FIRST]], V]], inverse[SECOND],
               VERTSECT[y]]], inverse[FIRST]], w → composite[x, inverse[FIRST]]}]

Out[50]= subclass[rec[x, y], composite[x, inverse[FIRST]]] == True

In[51]:= subclass[rec[x_, y_], composite[x_, inverse[FIRST]]] := True
```

It follows that the range of **rec[x,y]** is contained in the range of the generator **x**.

```
In[52]:= Map[implies[#, subclass[range[rec[x, y]], range[x]]] &,
           SubstTest[implies, subclass[u, v], subclass[range[u], range[v]],
            {u → rec[x, y], v -> composite[x, inverse[FIRST]]}]] // Reverse

Out[52]= subclass[range[rec[x, y]], range[x]] == True

In[53]:= subclass[range[rec[x_, y_]], range[x_]] := True
```

For the domain, one obtains this upper bound:

```
In[54]:= SubstTest[implies, subclass[u, v], subclass[domain[u], domain[v]],
          {u → rec[x, y], v -> composite[x, inverse[FIRST]]}]

Out[54]= subclass[domain[rec[x, y]], domain[domain[x]]] == True

In[55]:= subclass[domain[rec[x_, y_]], domain[domain[x_]]] := True
```

For the domain, one also obtains another upper bound, which follows from one derived previously, since **domain[VERTSECT[trv[y]]]** is contained in **domain[VERTSECT[y]]**.

```
In[56]:= Map[implies[#, subclass[domain[rec[x, y]], domain[VERTSECT[y]]]] &, SubstTest[
            implies, subclass[u, v], subclass[domain[u], domain[v]], {u → rec[x, y],
             v -> composite[x, id[composite[IMG, id[cart[P[inverse[FIRST]], V]],
                 inverse[SECOND], VERTSECT[y]]], inverse[FIRST]]}]] // Reverse

Out[56]= subclass[domain[rec[x, y]], domain[VERTSECT[y]]] == True

In[57]:= subclass[domain[rec[x_, y_]], domain[VERTSECT[y_]]] := True
```

---

key idea: every function is a VERTSECT

A key idea needed to derive a useful form of the recursion condition is exploited in this section. Note that the definition of **partrec** involves **VERTSECT**, but one could have replaced it with an arbitrary function. Any function can be written as **VERTSECT[y]**:

```
In[58]:= VERTSECT[union[composite[inverse[E], funpart[y]],
           cart[complement[domain[funpart[y]]], V]]]

Out[58]= funpart[y]
```

The **simplify** flag can be turned on.

```
In[59]:= simplify = True;
```

Now both flags are on:

```
In[60]:= {simplify, cond}

Out[60]= {True, True}
```

The trick to deriving the recursion equation is to get the **VERTSECT** in the definition of **partrec** to be temporarily replaced by **funpart**, and then to go back again. This is done as follows:

```
In[61]:= (member[setpart[u], partrec[funpart[v], composite[inverse[E], funpart[w]]]] //
           AssertTest) /.
         {u → setpart[x], v → funpart[y], w → VERTSECT[thinpart[z]]}

Out[61]= member[setpart[x], partrec[funpart[y], thinpart[z]]] ==
          and[subclass[image[thinpart[z], domain[setpart[x]]], domain[setpart[x]]],
           subclass[setpart[x], composite[funpart[y],
             id[composite[IMAGE[composite[id[setpart[x]], inverse[FIRST]]],
               VERTSECT[thinpart[z]]]], inverse[FIRST]]]]
```

This is simplified and a negated form is obtained to facilitate removal of the **setpart** wrapper:

```
In[62]:= Map[not[implies[#, subclass[setpart[x], composite[funpart[y],
              id[composite[IMAGE[composite[id[setpart[x]], inverse[FIRST]]],
                VERTSECT[thinpart[z]]]], inverse[FIRST]]]]] &, %]

Out[62]= and[member[setpart[x], partrec[funpart[y], thinpart[z]]],
           not[subclass[setpart[x], composite[funpart[y],
              id[composite[IMAGE[composite[id[setpart[x]], inverse[FIRST]]],
                VERTSECT[thinpart[z]]]], inverse[FIRST]]]] == False

In[63]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Equality substitution is used to remove the **setpart** wrapper:

```
In[64]:= SubstTest[implies,
          and[equal[x, w], or[not[member[w, partrec[funpart[y], thinpart[z]]]],
            subclass[w, composite[funpart[y], id[composite[IMAGE[composite[id[w],
                 inverse[FIRST]]], VERTSECT[thinpart[z]]]], inverse[FIRST]]]]],
          or[not[member[x, partrec[funpart[y], thinpart[z]]]], subclass[x, composite[
             funpart[y], id[composite[IMAGE[composite[id[x], inverse[FIRST]]],
               VERTSECT[thinpart[z]]]], inverse[FIRST]]]], w → setpart[x]]

Out[64]= or[not[member[x, partrec[funpart[y], thinpart[z]]]], subclass[x, composite[
             funpart[y], id[composite[IMAGE[composite[id[x], inverse[FIRST]]],
               VERTSECT[thinpart[z]]]], inverse[FIRST]]]] == True

In[65]:= or[not[member[x_, partrec[funpart[y_], thinpart[z_]]]],
           subclass[x_, composite[funpart[y_],
             id[composite[IMAGE[composite[id[x_], inverse[FIRST]]],
               VERTSECT[thinpart[z_]]]], inverse[FIRST]]]] := True
```

partial solutions are functions

Lemma.

```
In[66]:= SubstTest[implies, and[subclass[w, z], FUNCTION[z]], FUNCTION[w],
          z -> composite[funpart[x],
            id[composite[IMAGE[t], VERTSECT[y]]], inverse[FIRST]]]

Out[66]= or[FUNCTION[w], not[subclass[w, composite[funpart[x],
              id[composite[IMAGE[t], VERTSECT[y]]], inverse[FIRST]]]]] == True

In[67]:= (% /. {t → t_, w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

An immediate corollary is that when the generator **x** is a function any partial solution is a function:

```
In[68]:= Map[not,
          SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 → member[w, partrec[funpart[x], thinpart[y]]],
            p2 -> subclass[w, composite[funpart[x],
               id[composite[IMAGE[composite[id[w], inverse[FIRST]]],
                 VERTSECT[thinpart[y]]]], inverse[FIRST]]], p3 → FUNCTION[w]}]]

Out[68]= or[FUNCTION[w], not[member[w, partrec[funpart[x], thinpart[y]]]]] == True

In[69]:= or[FUNCTION[w_], not[member[w_, partrec[funpart[x_], thinpart[y_]]]]] := True
```

Removing the set variable **w** yields:

```
In[70]:= Map[equal[V, #] &, SubstTest[class, w, or[FUNCTION[w], not[member[w, z]]],
            z -> partrec[funpart[x], thinpart[y]]]] // Reverse

Out[70]= subclass[partrec[funpart[x], thinpart[y]], FUNS] == True

In[71]:= subclass[partrec[funpart[x_], thinpart[y_]], FUNS] := True
```

a recursion formula involving APPLY

In this section a recursion equation is obtained for the result of applying a partial solution of the recursion condition to an argument in its domain. We begin by looking at vertical sections:

```
In[72]:= SubstTest[implies, subclass[w, t], subclass[image[w, u], image[t, u]],
          {t -> composite[funpart[x],
             id[composite[IMAGE[composite[id[w], inverse[FIRST]]],
               VERTSECT[thinpart[y]]]], inverse[FIRST]], u → singleton[z]}]

Out[72]= or[not[subclass[w, composite[funpart[x],
              id[composite[IMAGE[composite[id[w], inverse[FIRST]]],
                VERTSECT[thinpart[y]]]], inverse[FIRST]]]],
           subclass[image[w, singleton[z]], singleton[APPLY[funpart[x],
              PAIR[z, composite[w, id[image[thinpart[y], singleton[z]]]]]]]]] == True

In[73]:= (% /. {w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed

In[74]:= Map[not, SubstTest[and, implies[p1, p3], implies[p1, p4],
           implies[and[p2, p3, p4], p5], not[implies[and[p1, p2], p5]],
           {p1 -> subclass[w, composite[funpart[x], id[composite[
                 IMAGE[composite[id[w], inverse[FIRST]]], VERTSECT[thinpart[y]]]],
              inverse[FIRST]]], p2 → member[z, domain[w]], p3 → FUNCTION[w],
            p4 → subclass[image[w, singleton[z]], singleton[APPLY[funpart[x],
               PAIR[z, composite[w, id[image[thinpart[y], singleton[z]]]]]]]],
            p5 → equal[APPLY[w, z], APPLY[funpart[x],
              PAIR[z, composite[w, id[image[thinpart[y], singleton[z]]]]]]}]]

Out[74]= or[equal[APPLY[w, z], APPLY[funpart[x],
             PAIR[z, composite[w, id[image[thinpart[y], singleton[z]]]]]]],
           not[member[z, domain[w]]], not[subclass[w, composite[funpart[x],
              id[composite[IMAGE[composite[id[w], inverse[FIRST]]],
                VERTSECT[thinpart[y]]]], inverse[FIRST]]]]] == True

In[75]:= or[equal[APPLY[funpart[x_],
             PAIR[z_, composite[w_, id[image[thinpart[y_], singleton[z_]]]]]],
            APPLY[w_, z_]], not[member[z_, domain[w_]]], not[subclass[w_, composite[
             funpart[x_], id[composite[IMAGE[composite[id[w_], inverse[FIRST]]],
               VERTSECT[thinpart[y_]]]], inverse[FIRST]]]]] := True
```

Corollary:

```
In[76]:= Map[not, SubstTest[and, implies[p1, p3],
           implies[and[p2, p3], p4], not[implies[and[p1, p2], p4]],
           {p1 → member[w, partrec[funpart[x], thinpart[y]]], p2 → member[z, domain[w]],
            p3 -> subclass[w, composite[funpart[x], id[composite[IMAGE[composite[id[w],
                  inverse[FIRST]]], VERTSECT[thinpart[y]]]], inverse[FIRST]]],
            p4 -> equal[APPLY[w, z], APPLY[funpart[x], PAIR[z,
                composite[w, id[image[thinpart[y], singleton[z]]]]]]}]]

Out[76]= or[equal[APPLY[w, z], APPLY[funpart[x],
             PAIR[z, composite[w, id[image[thinpart[y], singleton[z]]]]]]],
           not[member[w, partrec[funpart[x], thinpart[y]]]],
           not[member[z, domain[w]]]] == True
```

```
In[77]:= or[equal[APPLY[funpart[x_],
             PAIR[z_, composite[w_, id[image[thinpart[y_], singleton[z_]]]]]],
           APPLY[w_, z_]], not[member[w_, partrec[funpart[x_], thinpart[y_]]]],
         not[member[z_, domain[w_]]]] := True
```

---

compatibility

Two functions are said to be compatible if they agree on the intersections of thir domains. This is equivalent to the statement that their union is a function. It will be shown that any two partial solutions are compatible. The basic argument is this:

```
In[78]:= {implies[p1, p3], implies[p2, p4], implies[and[p1, p5], p9],
          implies[and[p2, p6], p10], implies[and[p3, p4, p7], p8],
          implies[and[p8, p9, p10], p11], implies[and[p3, p4, p5, p6, p11], p12],
          not[implies[and[p1, p2, p5, p6, p7], p12]]} /.
         {p1 → member[u, partrec[funpart[x], thinpart[y]]],
          p2 → member[v, partrec[funpart[x], thinpart[y]]],
          p3 → FUNCTION[u], p4 → FUNCTION[v], p5 → member[z, domain[u]],
          p6 → member[z, domain[v]],
          p7 → subclass[image[thinpart[y], singleton[z]],
            fix[composite[inverse[u], v]]],
          p8 → equal[composite[u, id[image[thinpart[y], singleton[z]]]],
            composite[v, id[image[thinpart[y], singleton[z]]]]],
          p9 → equal[APPLY[u, z], APPLY[funpart[x],
             PAIR[z, composite[u, id[image[thinpart[y], singleton[z]]]]]]],
          p10 -> equal[APPLY[v, z], APPLY[funpart[x],
             PAIR[z, composite[v, id[image[thinpart[y], singleton[z]]]]]]],
          p11 → equal[APPLY[u, z], APPLY[v, z]],
          p12 → member[z, fix[composite[inverse[u], v]]]}

Out[78]= {True, True, True, True, True, True, True,
          and[member[u, partrec[funpart[x], thinpart[y]]],
           member[v, partrec[funpart[x], thinpart[y]]], member[z, domain[u]],
           member[z, domain[v]], not[member[z, fix[composite[inverse[u], v]]]],
           subclass[image[thinpart[y], singleton[z]], fix[composite[inverse[u], v]]]]}
```

Verifying the validity of such a long argument is a challenge for the **GOEDEL** program. Bundling the hypotheses helps:

```
In[79]:= Map[not, SubstTest[and, implies[p1, p3],
           implies[p1, p4], implies[p1, p9], implies[p1, p10],
           implies[and[p1, p3, p4], p8], implies[and[p8, p9, p10], p11],
           implies[and[p1, p3, p4, p11], p12], not[implies[p1, p12]],
           {p1 → and[member[u, partrec[funpart[x], thinpart[y]]],
              member[v, partrec[funpart[x], thinpart[y]]], member[z, domain[u]],
              member[z, domain[v]], subclass[image[thinpart[y], singleton[z]],
               fix[composite[inverse[u], v]]]], p3 → FUNCTION[u], p4 → FUNCTION[v],
            p8 → equal[composite[u, id[image[thinpart[y], singleton[z]]]],
              composite[v, id[image[thinpart[y], singleton[z]]]]],
            p9 → equal[APPLY[u, z], APPLY[funpart[x],
               PAIR[z, composite[u, id[image[thinpart[y], singleton[z]]]]]]],
            p10 -> equal[APPLY[v, z], APPLY[funpart[x],
               PAIR[z, composite[v, id[image[thinpart[y], singleton[z]]]]]]],
            p11 → equal[APPLY[u, z], APPLY[v, z]],
            p12 → member[z, fix[composite[inverse[u], v]]]}]]

Out[79]= or[member[z, fix[composite[inverse[u], v]]],
           not[member[u, partrec[funpart[x], thinpart[y]]]],
           not[member[v, partrec[funpart[x], thinpart[y]]]],
           not[member[z, domain[u]]], not[member[z, domain[v]]], not[subclass[
             image[thinpart[y], singleton[z]], fix[composite[inverse[u], v]]]]] == True

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

The next step is to eliminate the variable **z**.

```
In[81]:= Map[equal[V, #] &, SubstTest[class, z, or[member[z, t], not[member[u, w]],
             not[member[v, w]], not[member[z, domain[u]]], not[member[z, domain[v]]],
             not[subclass[image[thinpart[y], singleton[z]], t]]],
            {t -> fix[composite[inverse[u], v]],
             w -> partrec[funpart[x], thinpart[y]]}]] // Reverse

Out[81]= or[not[member[u, partrec[funpart[x], thinpart[y]]]],
           not[member[v, partrec[funpart[x], thinpart[y]]]],
           subclass[intersection[domain[u], domain[v]],
            union[fix[composite[inverse[u], v]], image[inverse[thinpart[y]],
              complement[fix[composite[inverse[u], v]]]]]] == True

In[82]:= (% /. {u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

This is really a statement of subvariance, but the proof requires overcoming a sticky point that is relegated to the next section.

sticky point lemma

The derivation in this section took some doing, although **Otter** was able to derive this result without any help (independently, using a different line of reasoning) in a few seconds. In this section, the variable **x** refers to the class **fix[composite[inverse[u],v]]** in the preceding section, **y** refers to **intersection[domain[u], domain[v]]**, and **z** is what was called **thinpart[y]**.

```
In[83]:= Map[equal[#, union[image[inverse[z], complement[x]],
              image[inverse[z], complement[y]]]] &, SubstTest[image, inverse[z],
            union[u, v], {u → complement[y], v → dif[y, x]}]] // Reverse

Out[83]= equal[union[image[inverse[z], complement[x]],
            image[inverse[z], complement[y]]], union[image[inverse[z], complement[y]],
            image[inverse[z], intersection[y, complement[x]]]]] == True

In[84]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

This can be simplified.

```
In[85]:= SubstTest[implies, equal[u, v],
          equal[intersection[u, w], intersection[v, w]], {u -> union[
             image[inverse[z], complement[y]], image[inverse[z], complement[x]]],
           v -> union[image[inverse[z], complement[y]],
             image[inverse[z], intersection[y, complement[x]]]], w → y}]

Out[85]= equal[union[intersection[y, image[inverse[z], complement[x]]],
            intersection[y, image[inverse[z], complement[y]]]],
           union[intersection[y, image[inverse[z], complement[y]]],
            intersection[y, image[inverse[z], intersection[y, complement[x]]]]]] == True

In[86]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed

In[87]:= Map[implies[invariant[z, y], #] &,
           SubstTest[and, equal[0, u], equal[union[u, v], union[u, w]],
            {u -> intersection[y, image[inverse[z], complement[y]]],
             v -> intersection[y, image[inverse[z], complement[x]]], w -> intersection[
               y, image[inverse[z], intersection[y, complement[x]]]]}]] // Reverse

Out[87]= or[equal[intersection[y, image[inverse[z], complement[x]]],
            intersection[y, image[inverse[z], intersection[y, complement[x]]]]],
           not[subclass[image[z, y], y]]] == True

In[88]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Corollary

```
In[89]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p2, p3], not[implies[p1, p3]], {p1 → invariant[z, y],
            p2 → equal[intersection[y, image[inverse[z], complement[x]]],
              intersection[y, image[inverse[z], intersection[y, complement[x]]]]],
            p3 -> subclass[intersection[y, image[inverse[z], complement[x]]],
              image[inverse[z], intersection[y, complement[x]]]]}]]

Out[89]= or[not[subclass[image[z, y], y]],
           subclass[intersection[y, image[inverse[z], complement[x]]],
            image[inverse[z], intersection[y, complement[x]]]] == True

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

In[91]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
          {u → dif[y, x], v -> intersection[y, image[inverse[z], complement[x]]],
           w → image[inverse[z], dif[y, x]]}]

Out[91]= or[not[subclass[y, union[x, image[inverse[z], complement[x]]]]],
           not[subclass[intersection[y, image[inverse[z], complement[x]]],
             image[inverse[z], intersection[y, complement[x]]]], subclass[y,
            union[x, image[inverse[z], intersection[y, complement[x]]]]]] == True

In[92]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Corollary.

```
In[93]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p2, p3], p4],
           not[implies[and[p1, p3], p4]], {p1 → invariant[z, y],
            p2 -> subclass[intersection[y, image[inverse[z], complement[x]]],
              image[inverse[z], intersection[y, complement[x]]]],
            p3 -> subclass[y, union[x, image[inverse[z], complement[x]]]],
            p4 -> subclass[y,
              union[x, image[inverse[z], intersection[y, complement[x]]]]]}]]

Out[93]= or[not[subclass[y, union[x, image[inverse[z], complement[x]]]]],
           not[subclass[image[z, y], y]], subclass[y,
            union[x, image[inverse[z], intersection[y, complement[x]]]]] == True

In[94]:= or[not[subclass[image[z_, y_], y_]], not[
            subclass[y_, union[image[inverse[z_], complement[x_]], x_]]], subclass[y_,
            union[image[inverse[z_], intersection[complement[x_], y_]], x_]]] := True
```

Restatement of the sticky-point lemma.

```
In[95]:= implies[
          and[invariant[z, y], subclass[dif[y, x], image[inverse[z], complement[x]]]],
          subvariant[inverse[z], dif[y, x]]]

Out[95]= True
```

In practice, **y** is an intersection, so one needs one further wrinkle:

```
In[96]:= SubstTest[implies,
          and[invariant[z, y], subclass[dif[y, x], image[inverse[z], complement[x]]]],
          subvariant[inverse[z], dif[y, x]],
          y → intersection[y1, y2]]

Out[96]= or[not[subclass[image[z, intersection[y1, y2]], y1]],
           not[subclass[image[z, intersection[y1, y2]], y2]],
           not[subclass[intersection[y1, y2], union[x,
              image[inverse[z], complement[x]]]]], subclass[intersection[y1, y2],
            union[x, image[inverse[z], intersection[y1, y2, complement[x]]]]]] == True

In[97]:= (% /. {x → x_, y1 → y1_, y2 → y2_, z → z_}) /. Equal → SetDelayed

In[98]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[and[p1, p2], p4],
           implies[and[p3, p4, p5], p6], not[implies[and[p1, p2, p5], p6]],
           {p1 → invariant[z, y1], p2 → invariant[z, y2],
            p3 -> subclass[image[z, intersection[y1, y2]], y1],
            p4 -> subclass[image[z, intersection[y1, y2]], y2],
            p5 -> subclass[intersection[y1, y2],
              union[x, image[inverse[z], complement[x]]]],
            p6 → subvariant[inverse[z], intersection[y1, y2, complement[x]]]}]]

Out[98]= or[not[subclass[image[z, y1], y1]], not[subclass[image[z, y2], y2]],
           not[subclass[intersection[y1, y2], union[x,
              image[inverse[z], complement[x]]]]], subclass[intersection[y1, y2],
            union[x, image[inverse[z], intersection[y1, y2, complement[x]]]]]] == True

In[99]:= (% /. {x → x_, y1 → y1_, y2 → y2_, z → z_}) /. Equal → SetDelayed
```

---

thin relations whose inverses are wellfounded

Lemma 1.

```
In[100]:=
SubstTest[implies, and[subclass[u, v], WELLFOUNDED[v]],
 WELLFOUNDED[u], {u → inverse[thinpart[y]], v → inverse[y]}]

Out[100]=
or[not[WELLFOUNDED[inverse[y]]], WELLFOUNDED[inverse[thinpart[y]]]] == True

In[101]:=
or[not[WELLFOUNDED[inverse[y_]]], WELLFOUNDED[inverse[thinpart[y_]]]] := True
```

Lemma 2.

In[102]:=
```
SubstTest[implies, and[WELLFOUNDED[w], thin[inverse[w]], subvariant[w, x]],
 equal[0, x], w → inverse[thinpart[y]]]
```

Out[102]=
```
or[equal[0, x], not[subclass[x, image[inverse[thinpart[y]], x]]],
  not[WELLFOUNDED[inverse[thinpart[y]]]]] == True
```

In[103]:=
```
(% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

In[104]:=
```
Map[not, SubstTest[and, implies[p1, p2],
  implies[and[p2, p3], p4], not[implies[and[p1, p3], p4]],
  {p1 → WELLFOUNDED[inverse[y]], p2 -> WELLFOUNDED[inverse[thinpart[y]]],
   p3 -> subclass[x, image[inverse[thinpart[y]], x]], p4 → equal[0, x]}]]
```

Out[104]=
```
or[equal[0, x], not[subclass[x, image[inverse[thinpart[y]], x]]],
  not[WELLFOUNDED[inverse[y]]]] == True
```

In[105]:=
```
or[equal[0, x_], not[subclass[x_, image[inverse[thinpart[y_]], x_]]],
  not[WELLFOUNDED[inverse[y_]]]] := True
```

---

application to the compatibility condition

The results of the preceding sections are combined to complete the proof that partial solutions are compatible under the hypothesis that **y** is a thin relation whose inverse is wellfounded.

In[106]:=
```
SubstTest[implies, and[WELLFOUNDED[inverse[y]],
  subvariant[inverse[thinpart[y]], x]], equal[0, x],
 x → intersection[domain[u], domain[v],
   complement[fix[composite[inverse[u], v]]]]]
```

Out[106]=
```
or[not[subclass[intersection[domain[u], domain[v]], union[
     fix[composite[inverse[u], v]], image[inverse[thinpart[y]], intersection[
       complement[fix[composite[inverse[u], v]]], domain[u], domain[v]]]]]],
  not[WELLFOUNDED[inverse[y]]], subclass[intersection[domain[u], domain[v]],
   fix[composite[inverse[u], v]]]] == True
```

In[107]:=
```
(% /. {u → u_, v → v_, y → y_}) /. Equal → SetDelayed
```

The rest of the compatibility proof goes like this:

```
In[108]:=
        {implies[and[p2, p3], p4], implies[p2, p5], implies[p3, p6],
          implies[and[p4, p5, p6], p7], implies[and[p1, p7], p8],
          implies[p2, p9], implies[p3, p10], implies[and[p8, p9, p10], p11],
          not[implies[and[p1, p2, p3], p11]]} /.
         {p1 → WELLFOUNDED[inverse[y]],
          p2 -> member[u, partrec[funpart[x], thinpart[y]]],
          p3 -> member[v, partrec[funpart[x], thinpart[y]]],
          p4 -> subclass[intersection[domain[u], domain[v]],
            union[fix[composite[inverse[u], v]], image[inverse[thinpart[y]],
              complement[fix[composite[inverse[u], v]]]]]],
          p5 → invariant[thinpart[y], domain[u]],
          p6 → invariant[thinpart[y], domain[v]],
          p7 → subvariant[inverse[thinpart[y]], intersection[domain[u],
             domain[v], complement[fix[composite[inverse[u], v]]]]],
          p8 → subclass[intersection[domain[u], domain[v]],
            fix[composite[inverse[u], v]]], p9 → FUNCTION[u], p10 -> FUNCTION[v],
          p11 → FUNCTION[union[u, v]]}

Out[108]=
        {True, True, True, True, True, True, True,
         True, and[member[u, partrec[funpart[x], thinpart[y]]],
          member[v, partrec[funpart[x], thinpart[y]]],
          not[FUNCTION[union[u, v]]], WELLFOUNDED[inverse[y]]]}
```

Verifying the validity of this argument is again too much for the **GOEDEL** program. Bundling the hypotheses helps a little, but it is still a good idea to break the argument into two pieces:

In[109]:=

```
Map[not, SubstTest[and, implies[p1, p4], implies[p1, p5],
  implies[p1, p6], implies[and[p4, p5, p6], p7], not[implies[p1, p7]],
  {p1 → and[member[u, partrec[funpart[x], thinpart[y]]],
     member[v, partrec[funpart[x], thinpart[y]]]],
   p4 -> subclass[intersection[domain[u], domain[v]],
     union[fix[composite[inverse[u], v]], image[inverse[thinpart[y]],
       complement[fix[composite[inverse[u], v]]]]]],
   p5 → invariant[thinpart[y], domain[u]],
   p6 → invariant[thinpart[y], domain[v]],
   p7 → subvariant[inverse[thinpart[y]], intersection[domain[u],
      domain[v], complement[fix[composite[inverse[u], v]]]]]}]]
```

Out[109]=

```
or[not[member[u, partrec[funpart[x], thinpart[y]]]],
  not[member[v, partrec[funpart[x], thinpart[y]]]],
  subclass[intersection[domain[u], domain[v]],
   union[fix[composite[inverse[u], v]], image[inverse[thinpart[y]],
     intersection[complement[fix[composite[inverse[u], v]]],
      domain[u], domain[v]]]]]] == True
```

In[110]:=

```
(% /. {u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

In[111]:=

```
Map[not, SubstTest[and, implies[p1, p7], implies[and[p1, p7], p8],
  implies[p1, p9], implies[p1, p10],
  implies[and[p8, p9, p10], p11], not[implies[p1, p11]],
  {p1 → and[WELLFOUNDED[inverse[y]], member[u, partrec[funpart[x],
       thinpart[y]]], member[v, partrec[funpart[x], thinpart[y]]]],
   p7 → subvariant[inverse[thinpart[y]], intersection[domain[u],
      domain[v], complement[fix[composite[inverse[u], v]]]]],
   p8 → subclass[intersection[domain[u], domain[v]],
     fix[composite[inverse[u], v]]], p9 → FUNCTION[u], p10 -> FUNCTION[v],
   p11 → FUNCTION[union[u, v]]}]]
```

Out[111]=

```
or[FUNCTION[union[u, v]], not[member[u, partrec[funpart[x], thinpart[y]]]],
  not[member[v, partrec[funpart[x], thinpart[y]]]],
  not[WELLFOUNDED[inverse[y]]] == True
```

In[112]:=

```
or[FUNCTION[union[u_, v_]],
  not[member[u_, partrec[funpart[x_], thinpart[y_]]]],
  not[member[v_, partrec[funpart[x_], thinpart[y_]]]],
  not[WELLFOUNDED[inverse[y_]]]] := True
```

The final step is to remove the variables **u** and **v**. The **GOEDEL** program automatically uses the fact that a family of functions is pairwise compatible if and only if the union of the entire collection is a function.

In[113]:=
```
Map[equal[0, composite[Id, complement[#]]] &, SubstTest[class, pair[u, v],
   implies[and[subclass[P[inverse[y]], w], member[u, t], member[v, t]],
    member[pair[u, v], z]], {w → WF, t → partrec[funpart[x], thinpart[y]],
    z -> image[inverse[CUP], FUNS]}]] // Reverse
```

Out[113]=
```
or[FUNCTION[rec[funpart[x], thinpart[y]]], not[WELLFOUNDED[inverse[y]]]] == True
```

In[114]:=
```
or[FUNCTION[rec[funpart[x_], thinpart[y_]]],
  not[WELLFOUNDED[inverse[y_]]]] := True
```

The investigation of the properties of this function will be taken up in a separate notebook. In particular, it will be of interest to determine its domain, and to show that **rec[x,y]** is the unique solution of a recursion equation similar to that satisfied by the partial solutions.

In[115]:=
```
Date[] - starttime
```

Out[115]=
```
{0, 0, 0, 0, 5, 25.2031250}
```