well-founded recursion, part 6.

Johan G. F. Belinfante
revised 2004 September 15

```
In[1]:= SetDirectory["i:"]; << goedel61.13c; << tools.m;

        :Package Title: goedel61.13c          2004 September 13 at 10:10 a.m.

        It is now:  2004 Sep 15 at 19:20

        Loading Simplification Rules

        TOOLS.M                     Revised 2004 September 13

        weightlimit = 40
```

summary

In this sixth notebook on well-founded recursion, a uniqueness theorem is derived, and the result is applied to establish a connection between **rec[x,y]** and the function **RANK**.

a fact about history

The concepts of **HISTORY** and **history** are in fact equal, but deliberately, fewer rewrite rules have been made available for **history**. It is easy, however, to transfer facts about **HISTORY** to obtain corresponding facts about **history** whenever the need arises. To illustrate this, consider the following fact which is known to hold for the function **HISTORY[funpart[x],thinpart[y]]**.

```
In[2]:= implies[equal[cart[V, V], domain[x]],
          equal[V, domain[composite[x, HISTORY[funpart[w], thinpart[y]]]]]]

Out[2]= True
```

The transfer from **HISTORY** to **history** can be done using equality substitution

```
In[3]:= SubstTest[implies, and[equal[cart[V, V], domain[x]], equal[t,
            composite[x, HISTORY[funpart[w], thinpart[y]]]]], equal[V, domain[t]],
          t → composite[x, history[funpart[w], thinpart[y]]]]

Out[3]= or[equal[V, image[inverse[history[funpart[w], thinpart[y]]], domain[x]]],
          not[equal[cart[V, V], domain[x]]]] == True

In[4]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

temporary abbreviations

The following two temporary predicates are introduced here to provide a convenient language to describe various results that will be derived in this notebook. Note that partial solutions are not required to be sets. In the definition of a total solution, no condition is placed on the domain.

```
In[5]:= partialsolution[w_, x_, y_] :=
          and[invariant[y, domain[w]], subclass[w, composite[x, history[w, y]]]]

In[6]:= totalsolution[w_, x_, y_] := equal[w, composite[x, history[w, y]]]
```

If **x** is a function, any partial or total solution is also a function:

```
In[7]:= implies[
          and[FUNCTION[x], subclass[w, composite[x, history[w, y]]]], FUNCTION[w]]

Out[7]= True

In[8]:= implies[and[FUNCTION[x], equal[w, composite[x, history[w, y]]]], FUNCTION[w]]

Out[8]= True
```

Recall that the class **partrec[x,y]** consists of partial solutions that are sets, and **rec[x,y]** is defined to be the union of this class.

```
In[9]:= equiv[member[w, partrec[x, y]],
            and[member[w, V], partialsolution[w, x, y]]] // not // not

Out[9]= True

In[10]:= U[partrec[x, y]]

Out[10]= rec[x, y]
```

on the totality of total solutions

A **total function** is a function whose domain is **V**. A **total binary function** is one whose domain is **cart[V,V]**. Under suitable hypotheses, total solutions are also total functions.

```
In[11]:= SubstTest[implies, and[equal[r, funpart[w]],
           equal[s, thinpart[y]], equal[t, composite[x, HISTORY[r, s]]],
           equal[cart[V, V], domain[x]]], equal[V, domain[t]],
          {r -> w, s → composite[Id, y], t → composite[x, history[w, y]]}]

Out[11]= or[equal[V, image[inverse[history[w, y]], domain[x]]],
           not[equal[V, domain[VERTSECT[y]]]],
           not[equal[cart[V, V], domain[x]]], not[FUNCTION[w]]] == True

In[12]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

The condition **w = t** for **w** to be a total solution requires special attention:

```
In[13]:= SubstTest[implies, and[FUNCTION[w], equal[cart[V, V], domain[x]], thin[y],
           equal[t, composite[x, history[w, y]]]], equal[V, domain[t]], t → w]

Out[13]= or[equal[V, domain[w]], not[equal[V, domain[VERTSECT[y]]]],
           not[equal[w, composite[x, history[w, y]]]],
           not[equal[cart[V, V], domain[x]]], not[FUNCTION[w]]] == True

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

In practice, it is convenient to replace the condition that **w** be a function by the condition that **x** be a function.

```
In[15]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
           implies[and[p1, p2, p3], p4], not[implies[and[p1, p2], p4]],
           {p1 → and[FUNCTION[x], equal[cart[V, V], domain[x]], thin[y]],
            p2 → equal[w, composite[x, history[w, y]]],
            p3 → FUNCTION[w], p4 → equal[V, domain[w]]}]]

Out[15]= or[equal[V, domain[w]], not[equal[V, domain[VERTSECT[y]]]],
           not[equal[w, composite[x, history[w, y]]]],
           not[equal[cart[V, V], domain[x]]], not[FUNCTION[x]]] == True

In[16]:= or[equal[V, domain[w_]], not[equal[V, domain[VERTSECT[y_]]]],
           not[equal[w_, composite[x_, history[w_, y_]]]],
           not[equal[cart[V, V], domain[x_]]], not[FUNCTION[x_]]] := True
```

Restatement: if **x** is a total binary function and **y** is thin, then any total solution **w** is a total function.

---

total solutions are partial solutions

It is an easy corollary of the result derived in the preceding section that under the same hypotheses, the domain of a total solution **w** is invariant under **y**.

```
In[17]:= 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], totalsolution[w, x, y]],
            p2 → equal[V, domain[w]], p3 → invariant[y, domain[w]]}]]

Out[17]= or[not[equal[V, domain[VERTSECT[y]]]],
           not[equal[w, composite[x, history[w, y]]]],
           not[equal[cart[V, V], domain[x]]], not[FUNCTION[x]],
           subclass[image[y, domain[w]], domain[w]]] == True

In[18]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

It follows that if **x** is a total binary function, and if **y** is thin, then any total solution is also a partial solution.

```
In[19]:= implies[and[FUNCTION[x], equal[cart[V, V], domain[x]], thin[y],
            totalsolution[w, x, y]], partialsolution[w, x, y]] // NotNotTest

Out[19]= or[and[subclass[w, composite[x, history[w, y]]],
            subclass[image[y, domain[w]], domain[w]]],
           not[equal[V, domain[VERTSECT[y]]]],
           not[equal[w, composite[x, history[w, y]]]],
           not[equal[cart[V, V], domain[x]]], not[FUNCTION[x]]] == True

In[20]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Restatement:

```
In[21]:= implies[and[FUNCTION[x], equal[cart[V, V], domain[x]],
           thin[y], totalsolution[w, x, y]], partialsolution[w, x, y]]

Out[21]= True
```

a condition for rec[x,y] to be a total solution

In this section it is shown that if **x** is a total bunary function, and if **y** is a thin relation whose inverse is well-founded, then **rec[x,y]** is a total solution. It is already known that under these conditions **rec[x,y]** is a partial solution, and that it is a total function. To complete the proof, the idea is to show that **composite[x, history[rec[x,y],y]]** is also a function, and therefore **rec[x,y]** is a restriction of it. Since **rec[x,y]** is a total function, this restriction reduces to an equality.

```
In[22]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p1, p3], implies[and[p2, p3], p4], implies[p1, p5],
           implies[and[p4, p5], p6], not[implies[p1, p6]], {p1 -> and[FUNCTION[x],
              equal[cart[V, V], domain[x]], thin[y], WELLFOUNDED[inverse[y]]],
            p2 -> subclass[rec[x, y], composite[x, history[rec[x, y], y]]],
            p3 → FUNCTION[composite[x, history[rec[x, y], y]]],
            p4 → equal[rec[x, y], composite[x, history[rec[x, y], y],
               id[domain[rec[x, y]]]]], p5 → equal[domain[rec[x, y]], V],
            p6 -> equal[rec[x, y], composite[x, history[rec[x, y], y]]]}]]

Out[22]= or[equal[composite[x, history[rec[x, y], y]], 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[23]:= or[equal[composite[x_, history[rec[x_, y_], y_]], rec[x_, y_]],
           not[equal[V, domain[VERTSECT[y_]]]], not[equal[cart[V, V], domain[x_]]],
           not[FUNCTION[x_]], not[WELLFOUNDED[inverse[y_]]]] := True
```

Restatement:

```
In[24]:= implies[and[FUNCTION[x], equal[cart[V, V], domain[x]],
           thin[y], WELLFOUNDED[inverse[y]]], totalsolution[rec[x, y], x, y]]

Out[24]= True
```

compatibility of partial solutions

Functions are **compatible** if their union is a function. Recall the following basic theorem about compatibility of partial solutions:

```
In[25]:= implies[
           and[WELLFOUNDED[inverse[y]], partialsolution[u, funpart[x], thinpart[y]],
            partialsolution[v, funpart[x], thinpart[y]]], FUNCTION[union[u, v]]]

Out[25]= True
```

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

```
In[26]:= SubstTest[implies,
           and[equal[s, thinpart[y]], equal[t, funpart[x]], WELLFOUNDED[inverse[y]],
            partialsolution[u, t, s], partialsolution[v, t, s]],
           FUNCTION[union[u, v]], {s → composite[Id, y], t → x}]

Out[26]= or[FUNCTION[union[u, v]], not[equal[V, domain[VERTSECT[y]]]],
            not[FUNCTION[x]], not[subclass[u, composite[x, history[u, y]]]],
            not[subclass[v, composite[x, history[v, y]]]],
            not[subclass[image[y, domain[u]], domain[u]]],
            not[subclass[image[y, domain[v]], domain[v]]],
            not[WELLFOUNDED[inverse[y]]]] == True

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

Restatement of the compatibility theorem.

```
In[28]:= implies[and[FUNCTION[x], thin[y], WELLFOUNDED[inverse[y]],
            partialsolution[u, x, y], partialsolution[v, x, y]], FUNCTION[union[u, v]]]

Out[28]= True
```

---

a condition for partial solutions to be contained in rec[x,y]

In this section it is shown that under suitable hypotheses, any partial solution is contained in **rec[x,y]**. When **w** is a set, this is obvious, because **rec[x,y]** is the union of all small partial solutions:

```
In[29]:= implies[member[w, partrec[x, y]], subclass[w, rec[x, y]]]

Out[29]= True
```

The compatibility theorem for partial solutions can be applied to the case that one of the partial solutions is **rec[x,y]:**

```
In[30]:= SubstTest[implies,
          and[FUNCTION[x], thin[y], WELLFOUNDED[inverse[y]], partialsolution[v, x, y],
           partialsolution[w, x, y]], FUNCTION[union[v, w]], v → rec[x, y]]

Out[30]= or[FUNCTION[union[w, rec[x, y]]], not[equal[V, domain[VERTSECT[y]]]],
           not[FUNCTION[x]], not[subclass[w, composite[x, history[w, y]]]],
           not[subclass[image[y, domain[w]], domain[w]]],
           not[subclass[rec[x, y], composite[x, history[rec[x, y], y]]]],
           not[WELLFOUNDED[inverse[y]]]] == True

In[31]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

This statement can be simplified by using the fact that **rec[x,y]** is a partial solution under the assumed hypotheses:

```
In[32]:= Map[not, SubstTest[and, implies[and[p1, p2, p3], p5],
           implies[and[p1, p2, p3, p4, p5], p6],
           not[implies[and[p1, p2, p3, p4], p6]], {p1 → FUNCTION[x], p2 → thin[y],
            p3 → WELLFOUNDED[inverse[y]], p4 → partialsolution[w, x, y],
            p5 → subclass[rec[x, y], composite[x, history[rec[x, y], y]]],
            p6 → FUNCTION[union[w, rec[x, y]]]}]]

Out[32]= or[FUNCTION[union[w, rec[x, y]]], not[equal[V, domain[VERTSECT[y]]]],
           not[FUNCTION[x]], not[subclass[w, composite[x, history[w, y]]]],
           not[subclass[image[y, domain[w]], domain[w]]],
           not[WELLFOUNDED[inverse[y]]]] == True

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

One can now deduce that if **x** is a total bnary function, and if **y** is a thin relation whose inverse is well-founded, then any partial solution **w** is contained in **rec[x,y]**. Here it is not assumed that **w** is a set.

```
In[34]:= Map[not, SubstTest[and, implies[and[p1, p3, p4, p5], p6],
           implies[and[p1, p2, p3, p4], p7], implies[and[p6, p7], p8],
           not[implies[and[p1, p2, p3, p4, p5], p8]],
           {p1 → FUNCTION[x], p2 → equal[cart[V, V], domain[x]],
            p3 → thin[y], p4 → WELLFOUNDED[inverse[y]],
            p5 → partialsolution[w, x, y], p6 → FUNCTION[union[w, rec[x, y]]],
            p7 → equal[V, domain[rec[x, y]]], p8 → subclass[w, rec[x, y]]}]]

Out[34]= or[not[equal[V, domain[VERTSECT[y]]]], not[equal[cart[V, V], domain[x]]],
           not[FUNCTION[x]], not[subclass[w, composite[x, history[w, y]]]],
           not[subclass[image[y, domain[w]], domain[w]]],
           not[WELLFOUNDED[inverse[y]]], subclass[w, rec[x, y]]] == True
```

```
In[35]:= or[not[equal[V, domain[VERTSECT[y_]]]], not[equal[cart[V, V], domain[x_]]],
           not[FUNCTION[x_]], not[subclass[image[y_, domain[w_]], domain[w_]]],
           not[subclass[w_, composite[x_, history[w_, y_]]]],
           not[WELLFOUNDED[inverse[y_]]], subclass[w_, rec[x_, y_]]] := True
```

Restatement:

```
In[36]:= implies[and[FUNCTION[x], equal[cart[V, V], domain[x]], thin[y],
           WELLFOUNDED[inverse[y]], partialsolution[w, x, y]], subclass[w, rec[x, y]]]

Out[36]= True
```

uniqueness theorem

It is an easy corollary of the result derived in the preceding section that, under the same hypotheses, any total solution is contained in **rec[x,y]**.

```
In[37]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
           implies[and[p1, p3], p4], not[implies[and[p1, p2], p4]],
           {p1 -> and[FUNCTION[x], equal[cart[V, V], domain[x]], thin[y],
              WELLFOUNDED[inverse[y]]], p2 → totalsolution[w, x, y],
            p3 → partialsolution[w, x, y], p4 → subclass[w, rec[x, y]]}]]

Out[37]= or[not[equal[V, domain[VERTSECT[y]]]],
           not[equal[w, composite[x, history[w, y]]]],
           not[equal[cart[V, V], domain[x]]], not[FUNCTION[x]],
           not[WELLFOUNDED[inverse[y]]], subclass[w, rec[x, y]]] == True

In[38]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

This result can be sharpened to obtain the following uniqueness theorem for **rec[x,y]**:

```
In[39]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[and[p1, p2], p4],
           implies[and[p1, p4], p5] , implies[and[p1, p2], p6], implies[and[p3, p5], p7],
           implies[and[p6, p7], p8], not[implies[and[p1, p2], p8]],
           {p1 -> and[FUNCTION[x], equal[cart[V, V], domain[x]], thin[y],
              WELLFOUNDED[inverse[y]]], p2 → totalsolution[w, x, y],
            p3 → equal[V, domain[w]], p4 → partialsolution[w, x, y],
            p5 → FUNCTION[union[w, rec[x, y]]], p6 → subclass[w, rec[x, y]],
            p7 → subclass[rec[x, y], w], p8 → equal[w, rec[x, y]]}]]

Out[39]= or[equal[w, rec[x, y]], not[equal[V, domain[VERTSECT[y]]]],
           not[equal[w, composite[x, history[w, y]]]],
           not[equal[cart[V, V], domain[x]]], not[FUNCTION[x]],
           not[WELLFOUNDED[inverse[y]]]] == True
```

```
In[40]:= or[equal[w_, rec[x_, y_]], not[equal[V, domain[VERTSECT[y_]]]],
           not[equal[w_, composite[x_, history[w_, y_]]]],
           not[equal[cart[V, V], domain[x_]]], not[FUNCTION[x_]],
           not[WELLFOUNDED[inverse[y_]]]] := True
```

Restatement: if **x** is a total binary function, and if **y** is a thin relation whose inverse is well-founded, then **rec[x,y]** is the only total solution.

```
In[41]:= implies[and[FUNCTION[x], equal[cart[V, V], domain[x]], thin[y],
           WELLFOUNDED[inverse[y]], equal[w, composite[x, history[w, y]]]],
          equal[w, rec[x, y]]]

Out[41]= True
```

The uniqueness theorem can also be restated using **HISTORY** in place of **history**. This is admitedly less easy to read, but it is easier to use because more rewrite rules are available when one uses **HISTORY**.

```
In[42]:= SubstTest[implies, and[equal[t, composite[x, history[w, y]]],
           FUNCTION[x], equal[cart[V, V], domain[x]],
           thin[y], WELLFOUNDED[inverse[y]], equal[w, t]],
          equal[w, rec[x, y]], t → composite[x, HISTORY[w, y]]]

Out[42]= or[equal[w, rec[x, y]],
           not[equal[V, domain[VERTSECT[y]]]], not[equal[w, composite[x,
              id[composite[IMAGE[composite[id[w], inverse[FIRST]]], VERTSECT[y]]],
              inverse[FIRST]]]], not[equal[cart[V, V], domain[x]]],
           not[FUNCTION[x]], not[WELLFOUNDED[inverse[y]]]] == True

In[43]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

---

comment about an alternative formulation of uniqueness

Under the usual hypotheses, the conditions **equal[V,domain[w]]** and **subclass[w,composite[x,history[w,y]]]** imply that **w = rec[x,y]**.

```
In[44]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
           implies[p1, p4], implies[and[p3, p4], p5], implies[and[p2, p5], p6],
           not[implies[and[p1, p2], p6]], {p1 → and[FUNCTION[x],
              equal[cart[V, V], domain[x]], thin[y], WELLFOUNDED[inverse[y]]],
            p2 → and[equal[V, domain[w]], subclass[w, composite[x, history[w, y]]]],
            p3 → subclass[w, rec[x, y]], p4 → FUNCTION[rec[x, y]], p5 →
             equal[w, composite[rec[x, y], id[domain[w]]]], p6 → equal[w, rec[x, y]]}]]

Out[44]= or[equal[w, rec[x, y]], not[equal[V, domain[w]]],
           not[equal[V, domain[VERTSECT[y]]]], not[equal[cart[V, V], domain[x]]],
           not[FUNCTION[x]], not[subclass[w, composite[x, history[w, y]]]],
           not[subclass[w, rec[x, y]]], not[WELLFOUNDED[inverse[y]]]] == True

In[45]:= or[equal[w_, rec[x_, y_]], not[equal[V, domain[w_]]],
           not[equal[V, domain[VERTSECT[y_]]]], not[equal[cart[V, V], domain[x_]]],
           not[FUNCTION[x_]], not[subclass[w_, composite[x_, history[w_, y_]]]],
           not[WELLFOUNDED[inverse[y_]]]] := True
```

In practice this version of the uniqueness theorem appears to be harder to use. For the application considered below, the **subclass** version of the recursion condition on **w** led to complicated rewrites that are avoided when one uses the **equal** form.

RANK

The theory of rank and the Zermelo-von Neumann cumulative hierarchy were developed a few years ago without using recursive definitions. In this section, some of the facts about **rank** and the corresponding function **RANK** are reviewed. The **rank** of a regular set **x** is the lowest level of the cumulative hierarchy that holds it. The class of levels that hold a given set **x** is:

```
In[46]:= class[w, and[member[w, OMEGA], member[x, image[z, singleton[w]]]]] /. z → ZN

Out[46]= intersection[OMEGA, image[inverse[ZN], singleton[x]]]
```

The intersection of this is class is

```
In[47]:= A[intersection[OMEGA, image[inverse[ZN], singleton[x]]]]

Out[47]= rank[x]
```

Note that if **x** is regular, it does belong to the level of its rank.

```
In[48]:= member[x, image[ZN, singleton[rank[x]]]]
Out[48]= member[x, REGULAR]
```

The corresponding function **RANK** is **lambda[x, rank[x]]**, or equivalently:

```
In[49]:= VERTSECT[reify[x, rank[x]]]
Out[49]= RANK
```

The connection between **rank[x]** and **RANK** can also be exhibited more simply as follows:

```
In[50]:= member[pair[x, y], RANK]
Out[50]= and[equal[y, rank[x]], member[y, V]]
```

The condition that **y** be a set forces **x** to be regular:

```
In[51]:= member[pair[x, rank[x]], RANK]
Out[51]= member[x, REGULAR]
```

In other words, the domain of **RANK** is the class of regular sets:

```
In[52]:= domain[RANK]
Out[52]= REGULAR
```

---

a recursion relation for RANK

The rank of a regular set can be recursively computed using the following recursion relation:

```
In[53]:= implies[member[x, REGULAR], equal[rank[x], tc[image[RANK, x]]]]
Out[53]= True
```

The corresponding recursion relation holds for the function **RANK**.

```
In[54]:= composite[TC, IMAGE[RANK]]
Out[54]= composite[RANK, IMAGE[id[REGULAR]]]
```

The goal in this section is to provide a connection between these facts and the theory of well-founded recursion, using the following well-founded relation:

```
In[55]:= WELLFOUNDED[composite[id[REGULAR], E]]

Out[55]= True
```

One needs of course to extend **RANK** to a total function. The following result summarizes the connection between **RANK** and **rec[x,y]**.

```
In[56]:= equal[w, composite[x, HISTORY[w, y]]] /.
          {w → union[RANK, cart[complement[REGULAR], singleton[0]]],
           x → composite[TC, IMAGE[SECOND], SECOND],
           y → composite[inverse[E], id[REGULAR]]}

Out[56]= True
```

Using the **HISTORY** form of the uniqueness theorem, one finds:

```
In[57]:= SubstTest[implies, and[FUNCTION[x], equal[cart[V, V], domain[x]], thin[y],
           WELLFOUNDED[inverse[y]], equal[w, composite[x, HISTORY[w, y]]]],
          equal[w, rec[x, y]],
          {w → union[RANK, cart[complement[REGULAR], singleton[0]]],
           x → composite[TC, IMAGE[SECOND], SECOND],
           y → composite[inverse[E], id[REGULAR]]}]

Out[57]= equal[rec[composite[TC, IMAGE[SECOND], SECOND],
            composite[inverse[E], id[REGULAR]]],
           union[RANK, cart[complement[REGULAR], singleton[0]]]] == True

In[58]:= rec[composite[TC, IMAGE[SECOND], SECOND],
           composite[inverse[E], id[REGULAR]]] :=
          union[RANK, cart[complement[REGULAR], singleton[0]]]
```

---

some corollaries

```
In[59]:= SubstTest[implies,
          and[FUNCTION[x], thin[y], WELLFOUNDED[inverse[y]]], FUNCTION[rec[x, y]],
          {x -> composite[TC, IMAGE[SECOND], SECOND],
           y → composite[inverse[E], id[REGULAR]]}]

Out[59]= FUNCTION[union[RANK, cart[complement[REGULAR], singleton[0]]]] == True

In[60]:= FUNCTION[union[RANK, cart[complement[REGULAR], singleton[0]]]] := True
```

Lemma needed to derive a **history** form of the recursion relation:

```
In[61]:= SubstTest[implies, equal[t, HISTORY[x, y]],
          equal[composite[IMAGE[SECOND], SECOND, t],
           composite[IMAGE[x], VERTSECT[y]]], t → history[x, y]]

Out[61]= equal[composite[IMAGE[x], VERTSECT[y]],
           composite[IMAGE[SECOND], SECOND, history[x, y]]] == True

In[62]:= composite[IMAGE[SECOND], SECOND, history[x_, y_]] :=
          composite[IMAGE[x], VERTSECT[y]]
```

The **history** form of the recursion relation holds:

```
In[63]:= composite[TC, IMAGE[SECOND], SECOND,
          history[union[RANK, cart[complement[REGULAR], singleton[0]]],
           composite[inverse[E], id[REGULAR]]]]

Out[63]= union[RANK, cart[complement[REGULAR], singleton[0]]]
```