WFPART = lambda[x,wf[x]]

Johan G. F. Belinfante
2004 September 25

```
In[1]:= SetDirectory["i:"]; << goedel61.24a; << tools.m

        :Package Title: goedel61.24a          2004 September 24 at 5:50 p.m.

        It is now:  2004 Sep 25 at 8:58

        Loading Simplification Rules

        TOOLS.M                      Revised 2004 September 18

        weightlimit = 40
```

summary

The function **WFPART = lambda[x, wf[x]]** is studied in this notebook, and a reification formula for **wf[x]** is derived.

definition of WFPART

The following membership rule will serve to define **WFPART**.

```
In[2]:= member[x_, WFPART] := and[member[first[x], V], equal[second[x], wf[first[x]]]]
```

It is immediately evident from the definition that **WFPART** is a relation.

```
In[3]:= Map[equal[0, #] &, dif[WFPART, cart[V, V]] // Normality]

Out[3]= subclass[WFPART, cart[V, V]] == True

In[4]:= subclass[WFPART, cart[V, V]] := True

In[5]:= equal[composite[Id, WFPART], WFPART]

Out[5]= True

In[6]:= composite[Id, WFPART] := WFPART
```

Another immediate consequence is a formula for its fixed point class.

```
In[7]:=  fix[WFPART] // Normality

Out[7]=  fix[WFPART] == WF

In[8]:=  fix[WFPART] := WF
```

vertical section rule

In this section, a vertical section rule for **WFPART** is derived. To produce a clean formula, some lemmas are derived. The first one is needed to get rid of an **image[V,x]** expression.

```
In[9]:=  wf[union[x, complement[image[V, y]]]] // Normality

Out[9]=  wf[union[x, complement[image[V, y]]]] == composite[id[image[V, y]], wf[x]]

In[10]:=  wf[union[x_, complement[image[V, y_]]]] := composite[id[image[V, y]], wf[x]]
```

The second lemma is not specific to **wf[x]**. It reduces a double occurrence of an **image[-V,x]** expression to a single one.

```
In[11]:=  equal[intersection[image[V, x], singleton[composite[id[image[V, x]], y]]],
            intersection[image[V, x], singleton[composite[Id, y]]]] // assert

Out[11]=  True

In[12]:=  intersection[image[V, x_], singleton[composite[id[image[V, x_]], y_]]] :=
           intersection[image[V, x], singleton[composite[Id, y]]]
```

The vertical section rule is now clean:

```
In[13]:=  image[WFPART, singleton[x]] // Normality

Out[13]=  image[WFPART, singleton[x]] ==
           intersection[image[V, singleton[x]], singleton[wf[x]]]

In[14]:=  image[WFPART, singleton[x_]] :=
           intersection[image[V, singleton[x]], singleton[wf[x]]]
```

WFPART is a function

The derivation in this section can be done without turning off flags, but doing so would increase the execution time from 6.5 seconds to 16 seconds.

```
In[15]:= simplify = False; cond = False;
```

With the vertical section rule in place, one can derive the fact that **WFPART** is a function as follows:

```
In[16]:= Map[equal[0, #] &, dif[WFPART, funpart[WFPART]] // RelnNormality]

Out[16]= FUNCTION[WFPART] == True

In[17]:= FUNCTION[WFPART] := True
```

It is worth noting that the test used for this is **RelnNormality**, and not **VSNormality**, as one might have expected because the presence of the vertical section rule is crucial for this derivation. The **VSNormality** test would not work here.

normalization

To make progress, one needs to normalize **WFPART**, which requires turning the **cond** flag back on. The **simplify** flag can be left off for now.

```
In[18]:= cond = True;

In[19]:= WFPART // VSNormality // Reverse

Out[19]= intersection[composite[inverse[S], IMAGE[id[cart[V, V]]]],
           composite[inverse[IMAGE[SECOND]], DISJOINT, BIGCUP, SUBVAR],
           fix[composite[inverse[DIF], inverse[IMAGE[SECOND]],
             inverse[S], BIGCUP, SUBVAR, FIRST]]] == WFPART

In[20]:= % /. Equal → SetDelayed
```

range

The function **WFPART** is idempotent.

```
In[21]:= composite[WFPART, WFPART] // VSNormality

Out[21]= composite[WFPART, WFPART] == WFPART

In[22]:= composite[WFPART, WFPART] := WFPART
```

```
In[23]:= SubstTest[implies, and[FUNCTION[x], idempotent[x]],
          equal[range[x], fix[x]], x → WFPART]

Out[23]= equal[WF, range[WFPART]] == True

In[24]:= range[WFPART] := WF
```

The **simplify** flag needs to be turned back on now.

```
In[25]:= simplify = True;
```

Corollary.

```
In[26]:= Assoc[id[P[cart[V, V]]], id[WF], WFPART]

Out[26]= composite[id[P[cart[V, V]]], WFPART] == WFPART

In[27]:= composite[id[P[cart[V, V]]], WFPART] := WFPART
```

Some related results:

```
In[28]:= Assoc[WFPART, WFPART, inverse[WFPART]]

Out[28]= composite[WFPART, id[WF]] == id[WF]

In[29]:= composite[WFPART, id[WF]] := id[WF]

In[30]:= ImageComp[WFPART, WFPART, V] // Reverse

Out[30]= image[WFPART, WF] == WF

In[31]:= image[WFPART, WF] := WF
```

---

domain

The function **WFPART** is total:

```
In[32]:= Map[domain,
          class[pair[x, y], member[y, wf[x]]] // VERTSECT // RelnNormality // Reverse]

Out[32]= domain[WFPART] == V

In[33]:= domain[WFPART] := V
```

A similar computation yields a formula for **composite[inverse[E], WFPART]**.

```
In[34]:= Map[composite[inverse[E], #] &,
          class[pair[x, y], member[y, wf[x]]] // VERTSECT // RelnNormality]

Out[34]= intersection[
            composite[inverse[SECOND], complement[inverse[E]], BIGCUP, SUBVAR],
            inverse[E]] == composite[inverse[E], WFPART]

In[35]:= intersection[composite[inverse[SECOND], complement[inverse[E]],
            BIGCUP, SUBVAR], inverse[E]] := composite[inverse[E], WFPART]
```

The following simple formula for **WFPART** follows as a corollary:

```
In[36]:= VERTSECT[intersection[composite[inverse[SECOND],
            complement[inverse[E]], BIGCUP, SUBVAR], inverse[E]]]

Out[36]= WFPART
```

APPLY formula

Lemma.

```
In[37]:= SubstTest[implies, and[subclass[z, x], member[x, V]],
           member[z, V], z → composite[Id, w]] /. w → wf[x]

Out[37]= or[and[member[domain[wf[x]], V], member[range[wf[x]], V]],
           not[member[x, V]]] == True

In[38]:= (% /. x → x_) /. Equal → SetDelayed
```

Corollary of the lemma.

```
In[39]:= equal[union[complement[image[V, singleton[x]]],
           complement[image[V, singleton[domain[wf[x]]]]],
           complement[image[V, singleton[range[wf[x]]]]], wf[x]],
          union[complement[image[V, singleton[x]]], wf[x]]]

Out[39]= True

In[40]:= union[complement[image[V, singleton[x_]]],
           complement[image[V, singleton[domain[wf[x_]]]]],
           complement[image[V, singleton[range[wf[x_]]]]], wf[x_]] :=
          union[complement[image[V, singleton[x]]], wf[x]]

In[41]:= SubstTest[A, image[w, singleton[x]], w → WFPART] // Reverse

Out[41]= APPLY[WFPART, x] == union[complement[image[V, singleton[x]]], wf[x]]

In[42]:= APPLY[WFPART, x_] := union[complement[image[V, singleton[x]]], wf[x]]
```

composites with IMAGE[id[cart[V,V]]]

The following result can also be derived using **RelnRenormality**.

```
In[43]:= Assoc[IMAGE[id[cart[V, V]]], id[P[cart[V, V]]], WFPART]

Out[43]= composite[IMAGE[id[cart[V, V]]], WFPART] == WFPART

In[44]:= composite[IMAGE[id[cart[V, V]]], WFPART] := WFPART
```

The composite in the reverse order can be derived most simply using **RelnNormality**.

```
In[45]:= composite[WFPART, IMAGE[id[cart[V, V]]]] // RelnNormality

Out[45]= composite[WFPART, IMAGE[id[cart[V, V]]]] == WFPART

In[46]:= composite[WFPART, IMAGE[id[cart[V, V]]]] := WFPART
```

other composites

Theorem This formula is a variable-free version of the equation **wf[id[x]] = 0**.

```
In[47]:= composite[WFPART, IDP] // VSNormality

Out[47]= composite[WFPART, IMAGE[DUP]] == cart[V, singleton[0]]

In[48]:= composite[WFPART, IMAGE[DUP]] := cart[V, singleton[0]]
```

Deriving a variable-free counterpart of the equation **fix[wf[x]] = 0** is somewhat trickier:

```
In[49]:= Map[VERTSECT,
          composite[inverse[E], IMAGE[inverse[DUP]], WFPART] // VSRenormality]

Out[49]= composite[IMAGE[inverse[DUP]], WFPART] == cart[V, singleton[0]]

In[50]:= composite[IMAGE[inverse[DUP]], WFPART] := cart[V, singleton[0]]
```

Corollary.

```
In[51]:= ImageComp[IMAGE[inverse[DUP]], WFPART, V] // Reverse

Out[51]= image[IMAGE[inverse[DUP]], WF] == singleton[0]

In[52]:= image[IMAGE[inverse[DUP]], WF] := singleton[0]
```

Such results can more easily obtained using reification. The needed formulas are derived in the next section.

reify

The following lemmas are used to eliminate **RIF**.

```
In[53]:= fix[composite[complement[composite[
             intersection[composite[inverse[FIRST], inverse[SECOND], inverse[x]],
              composite[inverse[SECOND], E, FIRST]], inverse[SECOND]]],
           inverse[E], SECOND]] // InvertFixTest

Out[53]= fix[composite[complement[composite[
             intersection[composite[inverse[FIRST], inverse[SECOND], inverse[x]],
              composite[inverse[SECOND], E, FIRST]], inverse[SECOND]]],
           inverse[E], SECOND]] == fix[composite[inverse[SECOND], E,
           complement[composite[SECOND, intersection[composite[x, SECOND, FIRST],
              composite[inverse[FIRST], inverse[E], SECOND]]]]]]

In[54]:= fix[composite[complement[composite[
             intersection[composite[inverse[FIRST], inverse[SECOND], inverse[x_]],
              composite[inverse[SECOND], E, FIRST]], inverse[SECOND]]],
           inverse[E], SECOND]] := fix[composite[inverse[SECOND], E,
           complement[composite[SECOND, intersection[composite[x, SECOND, FIRST],
              composite[inverse[FIRST], inverse[E], SECOND]]]]]]

In[55]:= Map[inverse,
          composite[intersection[composite[inverse[x], SECOND], composite[complement[
              composite[complement[composite[complement[composite[inverse[x],
                   cross[inverse[E], Id]]]], id[inverse[E]], inverse[FIRST]]], E]],
             inverse[DUP], FIRST]], inverse[RIF], SWAP] // VSTriNormality]

Out[55]= composite[SWAP, RIF,
          intersection[composite[inverse[SECOND], x], composite[inverse[FIRST],
            DUP, complement[composite[inverse[E], complement[composite[FIRST,
                id[inverse[E]], complement[composite[cross[E, Id], x]]]]]]]]] ==
         composite[id[cart[V, V]], intersection[x, complement[
            inverse[fix[composite[complement[inverse[
                 fix[composite[inverse[SECOND], E, complement[composite[SECOND,
                     intersection[composite[x, SECOND, FIRST], composite[inverse[
                        FIRST], inverse[E], SECOND]]]]]]]], E, SECOND, FIRST]]]]]]

In[56]:= ((First[%] /. x → x_) == Last[%]) /. Equal → SetDelayed
```

The reification formula for **wf[x]** is derived as follows:

```
In[57]:= SubstTest[reify, x, composite[id[complement[U[f[g[x]]]]], g[x]], f → subvar]

Out[57]= reify[x, wf[g[x]]] ==
          composite[id[cart[V, V]], intersection[complement[inverse[
              fix[composite[complement[inverse[fix[composite[inverse[SECOND], E,
                    complement[composite[SECOND, intersection[composite[inverse[
                        FIRST], inverse[E], SECOND], composite[reify[x, g[x]], SECOND,
                      FIRST]]]]]]]]], E, SECOND, FIRST]]]], reify[x, g[x]]]]
```

This is the reification rule for **wf[x]**.

```
In[58]:= reify[x_, wf[y_]] :=
          composite[id[cart[V, V]], intersection[complement[inverse[
              fix[composite[complement[inverse[fix[composite[inverse[SECOND], E,
                    complement[composite[SECOND, intersection[composite[inverse[
                        FIRST], inverse[E], SECOND], composite[reify[x, y],
                      SECOND, FIRST]]]]]]]]], E, SECOND, FIRST]]]], reify[x, y]]]
```

variable-free formulation of fix[trv[wf[x]]] = 0

Here is a simple application of reification that appears to be hard to derive by other means.

```
In[59]:= Map[VERTSECT, SubstTest[reify, x, fix[trv[f[x]]], f → wf]] // Reverse

Out[59]= composite[IMAGE[inverse[DUP]], HULL[TRV], WFPART] == cart[V, singleton[0]]

In[60]:= composite[IMAGE[inverse[DUP]], HULL[TRV], WFPART] := cart[V, singleton[0]]
```

inclusions and intersections

```
In[61]:= Map[equal[0, #] &, dif[WFPART, inverse[S]] // VSNormality]

Out[61]= subclass[WFPART, inverse[S]] == True

In[62]:= subclass[WFPART, inverse[S]] := True
```

Corollary.

```
In[63]:= equal[intersection[WFPART, inverse[S]], WFPART]

Out[63]= True

In[64]:= intersection[WFPART, inverse[S]] := WFPART
```

Corollary.

```
In[65]:= AssInt[S, WFPART, inverse[S]]

Out[65]= intersection[S, WFPART] == id[WF]

In[66]:= intersection[S, WFPART] := id[WF]
```

fix formulas

```
In[67]:= IminInt[S, WFPART, V] // Reverse

Out[67]= fix[composite[inverse[S], WFPART]] == WF

In[68]:= fix[composite[inverse[S], WFPART]] := WF

In[69]:= SubstTest[range, intersection[w, inverse[S]], w → WFPART] // Reverse

Out[69]= fix[composite[WFPART, S]] == WF

In[70]:= fix[composite[WFPART, S]] := WF

In[71]:= ImageInt[S, WFPART, V] // Reverse // InvertFix

Out[71]= fix[composite[WFPART, inverse[S]]] == WF

In[72]:= fix[composite[WFPART, inverse[S]]] := WF
```