# trv[K]

Johan G. F. Belinfante
2004 March 23

```
In[1]:= << goedel55.21b; << tools.m

        :Package Title: goedel55.21b           2004 March 21 at 10:20 p.m.

        It is now:  2004 Mar 24 at 10:11

        Loading Simplification Rules

        TOOLS.M                       Revised 2004 March 14

        weightlimit = 40
```

## summary

An ordered pair **pair[x,y]** belongs to the transitive closure of the cover relation **K** if **y** is the union of **x** and some finite set. This yields a formula for the transitive closure of **K**. It is shown that the transitive closure of **union[Id,K]** is a partial order.

## a corollary of the associative law for CUP

Recall these formulas:

```
In[2]:= composite[id[cart[V, singleton[x]]], inverse[FIRST]]

Out[2]= RIGHT[x]

In[3]:= composite[ASSOC, RIGHT[x]]

Out[3]= cross[Id, RIGHT[x]]
```

The following lemma generalizes the above formula:

```
In[4]:= composite[ASSOC, id[cart[x, y]], inverse[FIRST]] // VSNormality

Out[4]= composite[ASSOC, id[cart[x, y]], inverse[FIRST]] ==
         composite[cross[Id, composite[id[cart[V, y]], inverse[FIRST]]], id[x]]

In[5]:= composite[ASSOC, id[cart[x_, y_]], inverse[FIRST]] :=
         composite[cross[Id, composite[id[cart[V, y]], inverse[FIRST]]], id[x]]
```

There is a companion formula with **RIGHT** replaced by **LEFT**, and **ASSOC** replaced with its inverse:

```
In[6]:= composite[inverse[ASSOC], LEFT[x]]

Out[6]= cross[LEFT[x], Id]
```

The companion formula can also be generalized, although this result will not be needed further in this notebook:

```
In[7]:= composite[inverse[ASSOC], id[cart[x, y]], inverse[SECOND]] // VSNormality

Out[7]= composite[inverse[ASSOC], id[cart[x, y]], inverse[SECOND]] ==
         composite[cross[composite[id[cart[x, V]], inverse[SECOND]], Id], id[y]]

In[8]:= composite[inverse[ASSOC], id[cart[x_, y_]], inverse[SECOND]] :=
         composite[cross[composite[id[cart[x, V]], inverse[SECOND]], Id], id[y]]
```

Application: The associative law for **CUP** implies the following corollary:

```
In[9]:= Assoc[composite[CUP, cross[Id, CUP]], ASSOC,
          composite[id[cart[V, x]], inverse[FIRST]]] // Reverse

Out[9]= composite[CUP, id[cart[V, x]], inverse[FIRST], CUP] ==
         composite[CUP, cross[Id, composite[CUP, id[cart[V, x]], inverse[FIRST]]]]

In[10]:= composite[CUP, id[cart[V, x_]], inverse[FIRST], CUP] :=
          composite[CUP, cross[Id, composite[CUP, id[cart[V, x]], inverse[FIRST]]]]
```

## normalization

The following temporary abbreviation may help to clarify the structure of some consequences of the formula derived in the preceding section:

```
In[11]:= c[x_] := composite[CUP, id[cart[V, x]], inverse[FIRST]]
```

To prevent this expression from being transformed by normality tests, it is useful to introduce the following rewrite rule:

```
In[12]:= c[x] // VSNormality // Reverse

Out[12]= intersection[S,
           inverse[fix[composite[inverse[DIF], inverse[S], id[x], inverse[S], FIRST]]]] ==
         composite[CUP, id[cart[V, x]], inverse[FIRST]]

In[13]:= intersection[S,
           inverse[fix[composite[inverse[DIF], inverse[S], id[x_], inverse[S], FIRST]]]] :=
         composite[CUP, id[cart[V, x]], inverse[FIRST]]
```

The dependence of relation **c[x]** on the parameter **x** can be exhibited explicitly. Another temporary abbreviation is introduced:

```
In[14]:= cee := composite[SWAP, inverse[rotate[CUP]]]
```

There is a rewrite rules for **rotate[CUP]** which makes a mess of this, but all one needs to know is that **c[x]** is an image of **cee**:

```
In[15]:= image[cee, x] == c[x]

Out[15]= True
```

It is a consequence of the associative law studied in the preceding section that **composite[c[x],c[y]]** can be written in the form **c[z]**, with **z = image[c[x],y]**:

```
In[16]:= composite[c[x], c[y]] == c[image[c[x], y]]

Out[16]= True
```

This is an important observation since since it implies that all powers of **c[x]** can be written as **c[something]**. Writing **z** for **cee**, one can analyze the structure of this formula as follows:

```
In[17]:= composite[image[z, x], image[z, y]] == image[z, image[image[z, x], y]] /. z -> cee

Out[17]= True
```

Abstraction can be used to remove the variable **y**:

```
In[18]:= Map[abstract[y, #] &,
           composite[image[z, x], image[z, y]] == image[z, image[image[z, x], y]]]

Out[18]= composite[cross[Id, image[z, x]], z] == composite[z, image[z, x]]
```

When one replaces **z** with **cee**, this result is not immediately recognized to be true because various rewrite rule interfere:

```
In[19]:= composite[cross[Id, image[z, x]], z] == composite[z, image[z, x]] /. z -> cee

Out[19]= composite[SWAP,
           cross[composite[CUP, id[cart[V, x]], inverse[FIRST]], Id], id[inverse[S]],
           intersection[composite[inverse[DIF], inverse[S]], composite[inverse[FIRST], S]]] ==
          composite[SWAP, id[inverse[S]], intersection[composite[inverse[DIF], inverse[S]],
            composite[inverse[FIRST], S]], CUP, id[cart[V, x]], inverse[FIRST]]
```

This result is nonetheless true, and one can derive it easily by removing the factor **SWAP** that appears on both sides, and then applying **TripleRotate** to the inverse of the expression on the left side:

```
In[20]:= Map[inverse, flip[inverse[composite[cross[Id, c[x]], cee]]] // TripleRotate]

Out[20]= composite[cross[composite[CUP, id[cart[V, x]], inverse[FIRST]], Id], id[inverse[S]],
           intersection[composite[inverse[DIF], inverse[S]], composite[inverse[FIRST], S]]] ==
          composite[id[inverse[S]], intersection[composite[inverse[DIF], inverse[S]],
            composite[inverse[FIRST], S]], CUP, id[cart[V, x]], inverse[FIRST]]

In[21]:= composite[cross[composite[CUP, id[cart[V, x_]], inverse[FIRST]], Id], id[inverse[S]],
           intersection[composite[inverse[DIF], inverse[S]], composite[inverse[FIRST], S]]] :=
          composite[id[inverse[S]], intersection[composite[inverse[DIF], inverse[S]],
            composite[inverse[FIRST], S]], CUP, id[cart[V, x]], inverse[FIRST]]
```

This verifies the expected formula:

```
In[22]:= composite[cross[Id, c[x]], cee] == composite[cee, c[x]]

Out[22]= True
```

## the main idea

The main idea is that iterated applications of the cover relation **K** amounts to repeatedly adding singletons to a set. If the singleton being added is already in the set, then nothing happens, so one gets something that is just short of being **union[Id,K]**:

```
In[23]:= composite[CUP, id[cart[V, range[SINGLETON]]], inverse[FIRST]]

Out[23]= union[K, id[complement[singleton[0]]]]
```

The only exception is that one cannot get the empty set from itself by adding a singleton. By combining **range[SINGLETON]** with **singleton[0]**, one obtains simpler formulas.

```
In[24]:= composite[CUP, id[cart[V, union[range[SINGLETON], singleton[0]]]], inverse[FIRST]] //
          RelnNormality

Out[24]= composite[CUP, id[cart[V, union[range[SINGLETON], singleton[0]]]], inverse[FIRST]] ==
          union[Id, K]

In[25]:= composite[CUP, id[cart[V, union[range[SINGLETON], singleton[0]]]], inverse[FIRST]] :=
          union[Id, K]
```

Replacing the iteration of **K** with that of **union[Id, K]** amounts to forming cumulative generations:

```
In[26]:= power[union[Id, x]]

Out[26]= composite[power[x], inverse[S], id[omega]]
```

Note that the range is the same, whether one uses **x** or **union[Id,x]**:

```
In[27]:= SubstTest[range, power[y], y → union[Id, x]]

Out[27]= image[power[x], image[inverse[S], omega]] == union[Id, trv[x]]

In[28]:= image[power[x_], image[inverse[S], omega]] := union[Id, trv[x]]
```

## iteration

The uniqueness of iteration is applied to get a formula for **power[union[Id,K]]**.

```
In[29]:= SubstTest[implies, and[equal[image[w, singleton[0]], v],
                    equal[composite[w, SUCC], composite[u, w]]],
                  equal[composite[w, id[omega]], iterate[u, v]],
                  {u → cross[Id, c[x]], v -> Id, w → composite[cee, iterate[c[x], singleton[0]]]}] /.
           x -> union[range[SINGLETON], singleton[0]]

Out[29]= equal[composite[power[K], inverse[S], id[omega]], composite[SWAP, id[inverse[S]],
           intersection[composite[inverse[DIF], inverse[S]], composite[inverse[FIRST], S]],
           inverse[CARD], id[omega], inverse[S], id[omega]]] == True
```

This is made into a temporary rewrite rule:

```
In[30]:= composite[power[K], inverse[S], id[omega]] := composite[SWAP, id[inverse[S]],
           intersection[composite[inverse[DIF], inverse[S]], composite[inverse[FIRST], S]],
           inverse[CARD], id[omega], inverse[S], id[omega]]
```

The above result looks a bit better when it is expressed as follows:

```
In[31]:= composite[power[K], inverse[S], id[omega]] ==
          composite[cee, inverse[CARD], id[omega], inverse[S], id[omega]]

Out[31]= True
```

Taking the range yields the main result in this notebook:

```
In[32]:= SubstTest[range, composite[power[x], inverse[S], id[omega]], x -> K]

Out[32]= composite[CUP, id[cart[V, FINITE]], inverse[FIRST]] == union[Id, trv[K]]

In[33]:= composite[CUP, id[cart[V, FINITE]], inverse[FIRST]] := union[Id, trv[K]]
```

## FINITE and trv[K]

The connection between **trv[K]** and **FINITE** is as follows:

```
In[34]:= SubstTest[range, iterate[x, singleton[0]], x -> K] // Reverse

Out[34]= union[image[trv[K], singleton[0]], singleton[0]] == FINITE

In[35]:= % /. Equal -> SetDelayed
```

It would be preferable to have a rewrite rule for **image[trv[K],singleton[0]]**, which requires knowing that **0** does not belong to it. This can be established as follows.

```
In[36]:= SubstTest[implies, subclass[u, v], subclass[trv[u], trv[v]], {u -> K, v -> S}]

Out[36]= subclass[trv[K], S] == True

In[37]:= subclass[trv[K], S] := True

In[38]:= SubstTest[implies, subclass[u, v], subclass[trv[u], trv[v]], {u -> K, v -> PS}]

Out[38]= equal[0, fix[trv[K]]] == True

In[39]:= fix[trv[K]] := 0

In[40]:= (AssertTest[member[x, fix[y]]] /. {x -> 0, y -> trv[K]}) // Reverse

Out[40]= member[pair[0, 0], trv[K]] == False

In[41]:= % /. Equal -> SetDelayed
```

Note that:

```
In[42]:= equal[intersection[complement[singleton[0]], image[trv[K], singleton[0]]],
           image[trv[K], singleton[0]]]

Out[42]= True
```

A corresponding rewrite rule is introduced:

```
In[43]:= intersection[complement[singleton[0]], image[trv[K], singleton[0]]] :=
           image[trv[K], singleton[0]]
```

This permits one to solve for **image[trv[K],singleton[0]]** in terms of the class **FINITE**.

```
In[44]:= SubstTest[dif, union[x, y], y,
           {x -> image[trv[K], singleton[0]], y -> singleton[0]}] // Reverse

Out[44]= image[trv[K], singleton[0]] == intersection[FINITE, complement[singleton[0]]]

In[45]:= image[trv[K], singleton[0]] := intersection[FINITE, complement[singleton[0]]]
```

A new conditional rewrite rule helps avoid problems with expressions like **union[FINITE,singleton[0]]**.

```
In[46]:= implies[member[y, x], equal[union[x, singleton[y]], x]]

Out[46]= True

In[47]:= union[x_, singleton[y_]] := x /; member[y, x]
```

## union[Id, trv[K]] is a partial order

In this section a derivation is given of the (rather obvious) fact that this is a partial order. The reflexive and transitive properties are easy:

```
In[48]:= SubstTest[subclass, z, cart[fix[z], fix[z]], z -> union[Id, x]] // Reverse

Out[48]= REFLEXIVE[union[Id, x]] == subclass[x, cart[V, V]]

In[49]:= REFLEXIVE[union[Id, x_]] := subclass[x, cart[V, V]]

In[50]:= SubstTest[TRANSITIVE, trv[y], y -> union[Id, x]]

Out[50]= TRANSITIVE[union[Id, trv[x]]] == True

In[51]:= TRANSITIVE[union[Id, trv[x_]]] := True
```

The proof of antisymmetry is a bit tedious, but straightforward. Begin by noticing:

```
In[52]:= equal[intersection[trv[K], PS], trv[K]]

Out[52]= True
```

The corresponding rewrite rule is introduced.

```
In[53]:= intersection[PS, trv[K]] := trv[K]
```

The inverse rule is also needed:

```
In[54]:= intersection[inverse[PS], inverse[trv[K]]] // DoubleInverse

Out[54]= intersection[inverse[PS], inverse[trv[K]]] == inverse[trv[K]]

In[55]:= % /. Equal → SetDelayed
```

The rest of the derivation uses the associativity of intersections:

```
In[56]:= AssInt[inverse[trv[K]], inverse[PS], PS] // Reverse

Out[56]= intersection[PS, inverse[trv[K]]] == 0

In[57]:= % /. Equal → SetDelayed

In[58]:= AssInt[inverse[trv[K]], PS, trv[K]]

Out[58]= intersection[inverse[trv[K]], trv[K]] == 0

In[59]:= intersection[inverse[trv[K]], trv[K]] := 0
```

One can assemble these results as follows:

```
In[60]:= SubstTest[and, REFLEXIVE[x], ANTISYMMETRIC[x],
           TRANSITIVE[x], x -> union[Id, trv[K]]] // Reverse

Out[60]= PARTIALORDER[union[Id, trv[K]]] == True

In[61]:= PARTIALORDER[union[Id, trv[K]]] := True
```