the function EQUIV = lambda[x, eqv[x]]

Johan G. F. Belinfante
2004 May 12

```
In[1]:= << goedel57.12y; << tools.m

        :Package Title: goedel57.12y      2004 May 12 at 9:50 p.m.

        It is now:  2004 May 13 at 9:50

        Loading Simplification Rules

        TOOLS.M                       Revised 2004 May 11

        weightlimit = 40
```

summary

The function **EQUIV = lambda[x, eqv[x]]** is introduced, and some of its properties are derived in this notebook. It is anticipated that this function will be useful for deriving variable−free statements about the theory of equivalence relations.

definition of EQUIV

The definition of **EQUIV = lambda[x, eqv[x]]** is:

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

sethood result

In this section it is shown that if **x** is a set, then so is **eqv[x]**. The converse need not hold.

```
In[3]:= (SubstTest[implies, member[u, V], member[intersection[u, v], V],
            {u -> trv[x], v -> inverse[trv[x]]}] /. x -> x_) /. Equal -> SetDelayed

In[4]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
          {p1 -> member[x, y], p2 -> member[trv[x], V], p3 -> member[eqv[x], V]}]]

Out[4]= or[member[eqv[x], V], not[member[x, y]]] == True

In[5]:= or[member[eqv[x_], V], not[member[x_, y_]]] := True
```

The proper class **PS** provides a counterexample for the converse statement:

```
In[6]:= eqv[PS]

Out[6]= 0
```

The sethood hypothesis can be eliminated by using the **setpart** wrapper. This formula will be used shortly to derive a formula for the domain of **EQUIV**.

```
In[7]:= SubstTest[implies, member[y, V], member[eqv[y], V], y -> setpart[x]]

Out[7]= member[eqv[setpart[x]], V] == True

In[8]:= member[eqv[setpart[x_]], V] := True
```

vertical section rule

A set−hood lemma is needed.

```
In[9]:= Map[implies[member[x, V], #] &, SubstTest[implies, equal[x, y],
          equal[eqv[x], eqv[y]], y -> union[x, complement[image[V, singleton[x]]]]]]

Out[9]= or[equal[eqv[x], eqv[union[x, complement[image[V, singleton[x]]]]]],
          not[member[x, V]]] == True

In[10]:= (% /. x -> x_) /. Equal -> SetDelayed
```

Lemma.

```
In[11]:= trv[union[x, complement[image[V, y]]]] // Normality

Out[11]= trv[union[x, complement[image[V, y]]]] ==
          union[cart[V, complement[image[V, y]]], trv[x]]

In[12]:= trv[union[x_, complement[image[V, y_]]]] :=
          union[cart[V, complement[image[V, y]]], trv[x]]

In[13]:= (SubstTest[fix, intersection[u, inverse[u]], u -> trv[v]]) /.
          v -> union[x, complement[image[V, y]]]

Out[13]= fix[eqv[union[x, complement[image[V, y]]]]] ==
          union[complement[image[V, y]], fix[trv[x]]]

In[14]:= fix[eqv[union[x_, complement[image[V, y_]]]]] :=
          union[complement[image[V, y]], fix[trv[x]]]
```

Technical lemma:

```
In[15]:= equal[intersection[image[V, singleton[x]],
            singleton[eqv[union[x, complement[image[V, singleton[x]]]]]]],
             intersection[image[V, singleton[x]], singleton[eqv[x]]]] // AssertTest

Out[15]= or[and[not[member[eqv[x], V]],
             not[member[eqv[union[x, complement[image[V, singleton[x]]]]], V]]],
           and[not[member[eqv[x], V]], not[member[fix[trv[x]], V]]],
           equal[singleton[eqv[x]], singleton[
             eqv[union[x, complement[image[V, singleton[x]]]]], not[member[x, V]]] == True

In[16]:= (% /. x -> x_) /. Equal -> SetDelayed
```

Restatement:

```
In[17]:= equal[intersection[image[V, singleton[x]],
            singleton[eqv[union[x, complement[image[V, singleton[x]]]]]]],
              intersection[image[V, singleton[x]], singleton[eqv[x]]]]

Out[17]= True
```

Make this a rewrite rule:

```
In[18]:= intersection[image[V, singleton[x_]],
            singleton[eqv[union[x_, complement[image[V, singleton[x_]]]]]]] :=
           intersection[image[V, singleton[x]], singleton[eqv[x]]]
```

Main result:

```
In[19]:= image[EQUIV, singleton[x]] // Normality

Out[19]= image[EQUIV, singleton[x]] == intersection[image[V, singleton[x]], singleton[eqv[x]]]

In[20]:= image[EQUIV, singleton[x_]] := intersection[image[V, singleton[x]], singleton[eqv[x]]]
```

EQUIV is a function

The following lemma is key:

```
In[21]:= member[x, composite[Di, y]] // AssertTest

Out[21]= member[x, composite[Di, y]] ==
           not[subclass[image[y, singleton[first[x]]], singleton[second[x]]]]

In[22]:= member[x_, composite[Di, y_]] :=
           not[subclass[image[y, singleton[first[x]]], singleton[second[x]]]]
```

From this lemma one quickly derives the fact that **EQUIV** is a function:

```
In[23]:= Map[equal[0, #] &, dif[EQUIV, funpart[EQUIV]] // Normality]

Out[23]= FUNCTION[EQUIV] == True

In[24]:= FUNCTION[EQUIV] := True
```

APPLY rule

Because **EQUIV** is a function, it is useful to have a formula for its application. The following lemma is needed to obtain a simple result.

```
In[25]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
                  {p1 -> member[x, V], p2 -> member[eqv[x], V], p3 -> member[fix[eqv[x]], V]}]]

Out[25]= or[member[fix[eqv[x]], V], not[member[x, V]]] == True

In[26]:= (% /. x -> x_) /. Equal -> SetDelayed
```

```
In[27]:= Map[complement,
           Map[complement, SubstTest[A, image[z, singleton[x]], z -> EQUIV]]] // Reverse

Out[27]= APPLY[EQUIV, x] == union[complement[image[V, singleton[x]]], eqv[x]]

In[28]:= APPLY[EQUIV, x_] := union[complement[image[V, singleton[x]]], eqv[x]]
```

membership rule for pairs

It is useful to remove the general membership rule prior to adding a special one for ordered pairs. It will be restored at the end of this section.

```
In[29]:= member[x_, EQUIV] =.
```

Lemma.

```
In[30]:= SubstTest[member, pair[x, y], composite[Id, z], z -> EQUIV] // Reverse

Out[30]= and[member[x, V], member[y, V], member[pair[x, y], EQUIV]] == member[pair[x, y], EQUIV]

In[31]:= and[member[x_, V], member[y_, V], member[pair[x_, y_], EQUIV]] :=
          member[pair[x, y], EQUIV]
```

The vertical section rule is used to derive the new membership rule for pairs:

```
In[32]:= SubstTest[member, y, image[z, singleton[x]], z -> EQUIV] // Reverse

Out[32]= member[pair[x, y], EQUIV] == and[equal[y, eqv[x]], member[x, V], member[y, V]]

In[33]:= member[pair[x_, y_], EQUIV] := and[equal[y, eqv[x]], member[x, V], member[y, V]]
```

The original membership rule can now be restored:

```
In[34]:= AssertTest[member[x, composite[Id, z]]] /. z -> EQUIV

Out[34]= member[x, EQUIV] == and[equal[eqv[first[x]], second[x]], member[first[x], V]]

In[35]:= member[x_, EQUIV] := and[equal[eqv[first[x]], second[x]], member[first[x], V]]
```

domain, range and fix

The sethood lemma for eqv with the sethood wrapper yields the fact that the function **EQUIV** is total:

```
In[36]:= SubstTest[class, x,
           member[image[z, singleton[setpart[x]]], range[SINGLETON]], z -> EQUIV] // Reverse

Out[36]= domain[EQUIV] == V

In[37]:= domain[EQUIV] := V
```

The fixed point class of **EQUIV** is easy to deduce:

```
In[38]:= fix[EQUIV] // Normality

Out[38]= fix[EQUIV] == EQV

In[39]:= fix[EQUIV] := EQV
```

The idempotence of **EQUIV** is also easy:

```
In[40]:= (symdif[composite[EQUIV, EQUIV], EQUIV] // VSNormality) /. Equal -> SetDelayed

In[41]:= SubstTest[equal, 0, symdif[u, v], {u -> composite[EQUIV, EQUIV], v -> EQUIV}]

Out[41]= True == equal[EQUIV, composite[EQUIV, EQUIV]]

In[42]:= composite[EQUIV, EQUIV] := EQUIV
```

As a corollary, one obtains a formula for the range:

```
In[43]:= SubstTest[implies, and[FUNCTION[x], equal[composite[x, x], x]],
           equal[fix[x], range[x]], x -> EQUIV]

Out[43]= equal[EQV, range[EQUIV]] == True

In[44]:= range[EQUIV] := EQV
```

variable−free formulation of properties of eqv[x]

The intended application of **EQUIV** to eliminating variables from the theory of equivalence relations will be illustrated in this section with a few examples. The variable−free formulation of the symmetric property of **eqv[x]** can be derived as follows:

```
In[45]:= (symdif[EQUIV, composite[IMAGE[SWAP], EQUIV]] // VSNormality) /. Equal -> SetDelayed

In[46]:= SubstTest[equal, 0, symdif[u, v], {u -> EQUIV, v -> composite[IMAGE[SWAP], EQUIV]}]

Out[46]= True == equal[EQUIV, composite[IMAGE[SWAP], EQUIV]]

In[47]:= composite[IMAGE[SWAP], EQUIV] := EQUIV
```

Corollaries:

```
In[48]:= Assoc[IMAGE[SWAP], IMAGE[SWAP], EQUIV] // Reverse

Out[48]= composite[IMAGE[id[cart[V, V]]], EQUIV] == EQUIV

In[49]:= composite[IMAGE[id[cart[V, V]]], EQUIV] := EQUIV

In[50]:= Assoc[id[P[cart[V, V]]], IMAGE[id[cart[V, V]]], EQUIV]

Out[50]= composite[id[P[cart[V, V]]], EQUIV] == EQUIV

In[51]:= composite[id[P[cart[V, V]]], EQUIV] := EQUIV
```

```
In[52]:= Assoc[IMAGE[SWAP], id[P[cart[V, V]]], EQUIV] // Reverse

Out[52]= composite[INVERSE, EQUIV] == EQUIV

In[53]:= composite[INVERSE, EQUIV] := EQUIV
```

A similar precedure can be used to eliminate the variable in the statement that **eqv[x]** is idempotent.

```
In[54]:= (symdif[EQUIV, composite[COMPOSE, DUP, EQUIV]] // VSNormality) /. Equal -> SetDelayed

In[55]:= SubstTest[equal, 0, symdif[u, v], {u -> EQUIV, v -> composite[COMPOSE, DUP, EQUIV]}]

Out[55]= True == equal[EQUIV, composite[COMPOSE, DUP, EQUIV]]

In[56]:= composite[COMPOSE, DUP, EQUIV] := EQUIV
```

The fact that the domain of **eqv[x]** is identical to its fixed−point set has the following formulation:

```
In[57]:= (symdif[composite[IMAGE[FIRST], EQUIV], composite[FIX, EQUIV]] // VSNormality) /.
          Equal -> SetDelayed

In[58]:= SubstTest[equal, 0, symdif[u, v],
          {u -> composite[IMAGE[FIRST], EQUIV], v -> composite[FIX, EQUIV]}]

Out[58]= True == equal[composite[IMAGE[FIRST], EQUIV], composite[IMAGE[inverse[DUP]], EQUIV]]

In[59]:= Equal[composite[IMAGE[FIRST], EQUIV], composite[IMAGE[inverse[DUP]], EQUIV]]

Out[59]= composite[IMAGE[FIRST], EQUIV] == composite[IMAGE[inverse[DUP]], EQUIV]

In[60]:= composite[IMAGE[FIRST], EQUIV] := composite[IMAGE[inverse[DUP]], EQUIV]
```

Corolllary:

```
In[61]:= Assoc[IMAGE[SECOND], IMAGE[SWAP], EQUIV]

Out[61]= composite[IMAGE[SECOND], EQUIV] == composite[IMAGE[inverse[DUP]], EQUIV]

In[62]:= composite[IMAGE[SECOND], EQUIV] := composite[IMAGE[inverse[DUP]], EQUIV]
```

These formulas, of course, contain less information than the statements from which they are derived in that the function **EQUIV** only reflects the properties of small equivalence relations, whereas the constructor **eqv[x]** can be applied to any class, not just to a set. It may be possible to overcome this limitation to some extent by approximating an arbitrary equivalence relation with its restrictions to sets.

```
In[63]:= SubstTest[implies, and[EQUIVALENCE[u], EQUIVALENCE[v]],
          EQUIVALENCE[intersection[u, v]], {u -> eqv[x], v -> cart[y, y]}]

Out[63]= EQUIVALENCE[composite[id[y], eqv[x], id[y]]] == True

In[64]:= EQUIVALENCE[composite[id[y_], eqv[x_], id[y_]]] := True
```

a formula for EQUIV

In this section it is shown that the function **EQUIV** can be expressed in terms of the functions **CORE[SYM]** and **HULL[TRV]**.

```
In[65]:= SubstTest[equal, trv[y], hull[TRV, y],     y -> composite[Id, setpart[x]]]

Out[65]= equal[hull[TRV, composite[Id, setpart[x]]], trv[setpart[x]]] == True

In[66]:= hull[TRV, composite[Id, setpart[x_]]] := trv[setpart[x]]

In[67]:= SubstTest[reify, x, image[z, singleton[setpart[x]]],
           z -> symdif[EQUIV, composite[CORE[SYM], HULL[TRV], IMAGE[id[cart[V, V]]]]]] //
         Reverse

Out[67]= union[intersection[EQUIV,
           composite[complement[CORE[SYM]], HULL[TRV], IMAGE[id[cart[V, V]]]]], intersection[
           complement[EQUIV], composite[CORE[SYM], HULL[TRV], IMAGE[id[cart[V, V]]]]]] == 0

In[68]:= (% /. Equal -> SetDelayed)

In[69]:= SubstTest[equal, 0, symdif[u, v],
           {u -> EQUIV, v -> composite[CORE[SYM], HULL[TRV], IMAGE[id[cart[V, V]]]]}]

Out[69]= True == equal[EQUIV, composite[CORE[SYM], HULL[TRV], IMAGE[id[cart[V, V]]]]]
```

This yields the following formula for **EQUIV**.

```
In[70]:= composite[CORE[SYM], HULL[TRV], IMAGE[id[cart[V, V]]]] := EQUIV
```

This formula will not be made permanent because it can be replaced with a simpler one:

```
In[71]:= Assoc[composite[CORE[SYM], HULL[TRV]], IMAGE[id[cart[V, V]]], id[P[cart[V, V]]]]

Out[71]= composite[CORE[SYM], HULL[TRV]] == composite[EQUIV, id[P[cart[V, V]]]]

In[72]:= composite[CORE[SYM], HULL[TRV]] := composite[EQUIV, id[P[cart[V, V]]]]
```

In order to make really effective use of this formula, one would need to know rather more about the two functions **CORE[SYM]** and **HULL[TRV]** that appear in it than is currently available in the **GOEDEL** program. Some trivial consequences are easy to derive:

```
In[73]:= Assoc[CORE[SYM], CORE[SYM], composite[HULL[TRV], IMAGE[id[cart[V, V]]]]]

Out[73]= composite[CORE[SYM], EQUIV] == EQUIV

In[74]:= composite[CORE[SYM], EQUIV] := EQUIV

In[75]:= Assoc[CORE[SYM], composite[HULL[TRV], IMAGE[id[cart[V, V]]]],
           IMAGE[id[cart[V, V]]]] // Reverse

Out[75]= composite[EQUIV, IMAGE[id[cart[V, V]]]] == EQUIV

In[76]:= composite[EQUIV, IMAGE[id[cart[V, V]]]] := EQUIV
```

```
In[77]:= Map[composite[#, id[P[cart[V, V]]]] &,
           Assoc[CORE[SYM], composite[HULL[TRV], IMAGE[id[cart[V, V]]]],
            composite[HULL[TRV], IMAGE[id[cart[V, V]]]]]] // Reverse

Out[77]= composite[EQUIV, HULL[TRV]] == composite[EQUIV, id[P[cart[V, V]]]]

In[78]:= composite[EQUIV, HULL[TRV]] := composite[EQUIV, id[P[cart[V, V]]]]
```

reify rule for eqv[x]

The reify rule for **trv** can be used to derive one for **eqv**.

```
In[79]:= SubstTest[reify, x, intersection[f[g[x]], inverse[f[g[x]]]], f -> trv]

Out[79]= reify[x, eqv[g[x]]] == composite[inverse[E], CAP,
           intersection[composite[inverse[FIRST], HULL[TRV], inverse[LB[reify[x, g[x]]]]],
            composite[inverse[SECOND], INVERSE, HULL[TRV], inverse[LB[reify[x, g[x]]]]]]]

In[80]:= reify[x_, eqv[y_]] := composite[inverse[E], CAP,
           intersection[composite[inverse[FIRST], HULL[TRV], inverse[LB[reify[x, y]]]],
            composite[inverse[SECOND], INVERSE, HULL[TRV], inverse[LB[reify[x, y]]]]]]
```

In practice this rule yields eqxpressions involving **CAP** that may sometimes be simplified further. Only the simplest case will be dealt with here.

```
In[81]:= SubstTest[reify, x, APPLY[z, x], z -> EQUIV]

Out[81]= composite[inverse[E], CAP,
           intersection[composite[inverse[FIRST], HULL[TRV], inverse[S]],
            composite[inverse[SECOND], INVERSE, HULL[TRV], inverse[S]]]] ==
          composite[inverse[E], EQUIV]

In[82]:= composite[inverse[E], CAP,
           intersection[composite[inverse[FIRST], HULL[TRV], inverse[S]], composite[
             inverse[SECOND], INVERSE, HULL[TRV], inverse[S]]]] := composite[inverse[E], EQUIV]
```

This yields a formula that can be interpreted as **lambda[x, eqv[x]] = EQUIV**.

```
In[83]:= VERTSECT[reify[x, eqv[x]]]

Out[83]= EQUIV
```