composite[eval[x], APPLY[CURRY, y]]

Johan G. F. Belinfante
2006 November 22

```
In[1]:= SetDirectory["l:"]; << goedel87.21a; << tools.m

        :Package Title: goedel87.21a          2006 November 21 at 1:45 a.m.

        It is now:  2006 Nov 22 at 8:27

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 November 5

        weightlimit = 40
```

summary

Rewrite rules are derived for composites of the evaluation function **eval[x]** and curried functions. Some examples in natural arithmetic are provided to illustrate the general results.

a basic CURRY formula

The following formula connects the functions **eval[x]** and **CURRY**.

```
In[2]:= Assoc[IMAGE[cross[Id, eval[x]]], VS,
          composite[IMAGE[ASSOC], id[P[cart[cart[V, V], V]]]]]

Out[2]= composite[IMAGE[cross[Id, eval[x]]], CURRY] == composite[FUNPART,
          IMAGE[cross[Id, inverse[LEFT[x]]]], IMAGE[ASSOC], id[P[cart[cart[V, V], V]]]]

In[3]:= composite[IMAGE[cross[Id, eval[x_]]], CURRY] := composite[FUNPART,
          IMAGE[cross[Id, inverse[LEFT[x]]]], IMAGE[ASSOC], id[P[cart[cart[V, V], V]]]]
```

new ApComp test

The following test is often a convenient replacement for **ImageComp** when one is dealing with the application of composites of two functions.

```
In[4]:= ApComp[x_, y_, z_] := Module[{u = Unique[], v = Unique[]},
          SubstTest[APPLY, composite[funpart[u], funpart[v]], z, {u → x, v → y}]]
```

composites of eval[x]

An existing rewrite rule for the composite of **eval[x]** and **VERTSECT[y]** has this counterpart:

```
In[5]:= composite[eval[x], IMAGE[SWAP], VERTSECT[y]] // RelnNormality

Out[5]= composite[eval[x], IMAGE[SWAP], VERTSECT[y]] ==
         composite[funpart[composite[inverse[RIGHT[x]], y]], id[domain[VERTSECT[y]]]]

In[6]:= composite[eval[x_], IMAGE[SWAP], VERTSECT[y_]] :=
         composite[funpart[composite[inverse[RIGHT[x]], y]], id[domain[VERTSECT[y]]]]
```

Using **funpart** and **setpart** wrappers, one derives:

```
In[7]:= ApComp[IMAGE[cross[Id, eval[x]]], CURRY, composite[funpart[setpart[y]], id[cart[V, V]]]]

Out[7]= composite[eval[x], APPLY[CURRY, composite[funpart[setpart[y]], id[cart[V, V]]]]] ==
         composite[funpart[setpart[y]], RIGHT[x]]

In[8]:= composite[eval[x_], APPLY[CURRY, composite[funpart[setpart[y_]], id[cart[V, V]]]]] :=
         composite[funpart[setpart[y]], RIGHT[x]]
```

Removing the wrappers, one obtains:

```
In[9]:= Map[implies[member[y, z], #] &,
         SubstTest[implies, equal[y, composite[funpart[setpart[t]], id[cart[V, V]]]],
           equal[composite[eval[x], APPLY[CURRY, y]], composite[y, RIGHT[x]]], t → y] // Reverse]

Out[9]= or[equal[composite[y, RIGHT[x]], composite[eval[x], APPLY[CURRY, y]]],
          not[FUNCTION[y]], not[member[y, z]], not[subclass[domain[y], cart[V, V]]]] == True

In[10]:= (% /. {x → x_, y → y_, z :> z_}) /. Equal → SetDelayed
```

A similar result holds for flipped functions:

```
In[11]:= SubstTest[or, equal[composite[z, RIGHT[x]], composite[eval[x], APPLY[CURRY, z]]],
           not[FUNCTION[z]], not[member[z, V]], not[subclass[domain[z], cart[V, V]]],
           z → flip[funpart[setpart[y]]]] // Reverse

Out[11]= equal[composite[eval[x], APPLY[CURRY, composite[funpart[setpart[y]], SWAP]]],
           composite[funpart[setpart[y]], LEFT[x]]] == True

In[12]:= composite[eval[x_], APPLY[CURRY, composite[funpart[setpart[y_]], SWAP]]] :=
          composite[funpart[setpart[y]], LEFT[x]]
```

Again, one can derive a similar result without wrappers:

```
In[13]:= Map[implies[member[y, z], #] &,
          SubstTest[implies, equal[y, composite[funpart[setpart[t]]]],
           equal[composite[eval[x], APPLY[CURRY, flip[y]]], composite[y, LEFT[x]]], t → y] //
          Reverse]

Out[13]= or[equal[composite[y, LEFT[x]], composite[eval[x], APPLY[CURRY, composite[y, SWAP]]]],
           not[FUNCTION[y]], not[member[y, z]]] == True

In[14]:= or[equal[composite[y_, LEFT[x_]],
            composite[eval[x_], APPLY[CURRY, composite[y_, SWAP]]]],
           not[FUNCTION[y_]], not[member[y_, z_]]] := True
```

---

some examples

A simple example of the above results is the following formula:

```
In[15]:= SubstTest[composite, eval[x],
           APPLY[CURRY, flip[funpart[setpart[z]]]], z → NATMUL] // Reverse

Out[15]= composite[eval[x], TIMES] == times[x]
```

The same result can also be derived another way without explicit use of **CURRY** as follows:

```
In[16]:= SubstTest[composite, eval[x],
           VERTSECT[inverse[rotate[y]]], y → rotate[NATMUL]] // Reverse

Out[16]= composite[eval[x], TIMES] == times[x]

In[17]:= composite[eval[x_], TIMES] := times[x]
```

Another application is to the function **NATMOD**.

```
In[18]:= SubstTest[composite, eval[x],
           APPLY[CURRY, composite[funpart[setpart[y]], id[cart[V, V]]]], y → NATMOD] // Reverse

Out[18]= composite[eval[x], APPLY[CURRY, NATMOD]] == modulo[x]

       composite[eval[x_], APPLY[CURRY, NATMOD]] := modulo[x]

In[19]:= SubstTest[composite, eval[x],
           APPLY[CURRY, composite[funpart[setpart[y]], SWAP]], y → NATMOD] // Reverse

Out[19]= composite[eval[x], APPLY[CURRY, composite[NATMOD, SWAP]]] == composite[NATMOD, LEFT[x]]

In[20]:= composite[eval[x_], APPLY[CURRY, composite[NATMOD, SWAP]]] :=
          composite[NATMOD, LEFT[x]]
```

For the function **NATEXP** one has:

```
In[21]:= SubstTest[composite, eval[x],
           APPLY[CURRY, composite[funpart[setpart[y]], id[cart[V, V]]]], y → NATEXP] // Reverse

Out[21]= composite[eval[x], APPLY[CURRY, NATEXP]] == composite[NATEXP, RIGHT[x]]

In[22]:= composite[eval[x_], APPLY[CURRY, NATEXP]] := composite[NATEXP, RIGHT[x]]

In[23]:= SubstTest[composite, eval[x],
           APPLY[CURRY, flip[funpart[setpart[y]]]], y → NATEXP] // Reverse

Out[23]= composite[eval[x], APPLY[CURRY, composite[NATEXP, SWAP]]] == composite[NATEXP, LEFT[x]]

In[24]:= composite[eval[x_], APPLY[CURRY, composite[NATEXP, SWAP]]] :=
          composite[NATEXP, LEFT[x]]
```