intadd[x,y]

Johan G. F. Belinfante
2003 August 9

```
In[1]:= << goedel52.s71; << tools.m

        :Package Title: goedel52.s71       2003 August 9 at 9:40 a.m.

        It is now:  2003 Aug 15 at 10:55

        Loading Simplification Rules

        TOOLS.M                       Revised 2003 August 9

        weightlimit = 40
```

summary

A wrapped membership rule for **intadd[x,y]** was added to the current version of the **GOEDEL** program. Some basic consequences of this rule are derived here.

time−consuming basics

It is shown that the expression **APPLY[INTADD,PAIR[x,y]]** is equal to **intadd[x,y]**.

```
In[2]:= simplify = False; cond = False;

In[3]:= subclass[composite[x, y], intadd[x, y]] // AssertTest

Out[3]= subclass[composite[x, y], intadd[x, y]] == True

In[4]:= subclass[composite[x_, y_], intadd[x_, y_]] := True
```

This takes a long time:

```
In[5]:= intadd[x, y] // Normality // Reverse

Out[5]= union[A[intersection[Z, image[S, singleton[composite[x, y]]]]],
          complement[image[V, intersection[omega,
             fix[composite[complement[image[complement[EQUIDIFF], x]], id[omega],
               complement[composite[SECOND, intersection[composite[inverse[NATADD], NATADD],
                  composite[inverse[FIRST], complement[x], FIRST]], inverse[SECOND]]]]]]]],
          complement[image[V, intersection[omega, fix[composite[
               complement[image[complement[EQUIDIFF], y]], id[omega],
               complement[composite[SECOND, intersection[composite[inverse[NATADD], NATADD],
                  composite[inverse[FIRST], complement[y], FIRST]], inverse[SECOND]]]]]]]],
          complement[image[V, singleton[x]]], complement[image[V, singleton[y]]]] ==
         intadd[
          x,
          y]
```

```
In[6]:= ((First[%] /. {x -> x_, y -> y_}) == Last[%]) /. Equal -> SetDelayed

In[7]:= A[image[INTADD, cart[singleton[x], singleton[y]]]] // Normality

Out[7]= A[image[INTADD, cart[singleton[x], singleton[y]]]] == intadd[x, y]

In[8]:= A[image[INTADD, cart[singleton[x_], singleton[y_]]]] := intadd[x, y]
```

V rule

The expression **intadd[x,y]** reduces to **V** when **x** or **y** fails to be an integer.

```
In[9]:= SubstTest[equal, V, APPLY[u, v], {u -> INTADD, v -> PAIR[x, y]}]

Out[9]= equal[V, intadd[x, y]] == or[not[member[x, Z]], not[member[y, Z]]]

In[10]:= equal[V, intadd[x_, y_]] := or[not[member[x, Z]], not[member[y, Z]]]
```

For example:

```
In[11]:= equal[V, intadd[0, x]]

Out[11]= True

In[12]:= intadd[0, x_] := V
```

vertical section rule

```
In[13]:= SubstTest[implies, FUNCTION[w],
           equal[image[w, singleton[z]], singleton[APPLY[w, z]]],
             {w -> INTADD, z -> PAIR[x, y]}]

Out[13]= equal[image[INTADD, cart[singleton[x], singleton[y]]], singleton[intadd[x, y]]] == True

In[14]:= image[INTADD, cart[singleton[x_], singleton[y_]]] := singleton[intadd[x, y]]
```

sethood rule

```
In[15]:= Map[not, SubstTest[equal, 0, image[w, singleton[PAIR[x, y]]], w -> INTADD]]

Out[15]= member[intadd[x, y], V] == and[member[x, Z], member[y, Z]]

In[16]:= member[intadd[x_, y_], V] := and[member[x, Z], member[y, Z]]
```

integer−hood rule

The integer−hood rule is similar to the sethood rule.

```
In[17]:= SubstTest[subclass, image[w, singleton[PAIR[x, y]]], range[w], w -> INTADD]

Out[17]= or[member[intadd[x, y], Z], not[member[x, Z]], not[member[y, Z]]] == True

In[18]:= or[member[intadd[x_, y_], Z], not[member[x_, Z]], not[member[y_, Z]]] := True

In[19]:= SubstTest[implies, member[w, Z], member[w, V], w -> intadd[x, y]]

Out[19]= or[and[member[x, Z], member[y, Z]], not[member[intadd[x, y], Z]]] == True

In[20]:= or[and[member[x_, Z], member[y_, Z]], not[member[intadd[x_, y_], Z]]] := True

In[21]:= equiv[member[intadd[x, y], Z], and[member[x, Z], member[y, Z]]]

Out[21]= True

In[22]:= member[intadd[x_, y_], Z] := and[member[x, Z], member[y, Z]]
```

some observations

```
In[23]:= cond = True; simplify = True;
```

The connection between **INTADD** and **intadd[x,y]** can be expressed as follows:

```
In[24]:= lambda[pair[x, y], intadd[x, y]]

Out[24]= INTADD

In[25]:= member[pair[pair[x, y], intadd[x, y]], INTADD]

Out[25]= and[member[x, Z], member[y, Z]]
```

an example

In this section addition of natural numbers is related to addition of non−negative integers. The idea is to use this fact:

```
In[26]:= implies[FUNCTION[w], equal[image[w, singleton[u]], singleton[APPLY[w, u]]]]

Out[26]= True
```

The first lemmas have nothing to do with arithmetic.

```
In[27]:= SubstTest[implies, and[member[v, s], equal[s, t]], member[v, t],
             {s -> image[w, singleton[u]], t -> singleton[APPLY[w, u]]}]

Out[27]= or[equal[v, A[image[w, singleton[u]]]],
           not[equal[image[w, singleton[u]], singleton[A[image[w, singleton[u]]]]]],
           not[member[u, V]], not[member[v, V]], not[member[pair[u, v], w]]] == True

In[28]:= (% /. {u -> u_, v -> v_, w -> w_}) /. Equal -> SetDelayed
```

```
In[29]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p2, p3], p4],
                 not[implies[and[p1, p3], p4]],
                 {p1 -> FUNCTION[w],
              p2 -> equal[image[w, singleton[u]], singleton[APPLY[w, u]]],
                    p3 -> member[pair[u, v], composite[Id, w]],
                    p4 -> equal[v, A[image[w, singleton[u]]]]}]]

Out[29]= or[equal[v, A[image[w, singleton[u]]]], not[FUNCTION[w]],
           not[member[u, V]], not[member[v, V]], not[member[pair[u, v], w]]] == True

In[30]:= SubstTest[implies, and[member[x, y], equal[y, z]], member[x, z],
             {z -> composite[Id, y], x -> pair[u, v]}]

Out[30]= or[and[member[u, V], member[v, V]],
           not[member[pair[u, v], y]], not[subclass[y, cart[V, V]]]] == True

In[31]:= or[and[member[u_, V], member[v_, V]],
           not[member[pair[u_, v_], y_]], not[subclass[y_, cart[V, V]]]] := True

In[32]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p2, p3], p4], implies[p1, p5],
                 implies[and[p5, p6], p3],
                 not[implies[and[p1, p6], p4]],
                 {p1 -> FUNCTION[w],
              p2 -> equal[image[w, singleton[u]], singleton[APPLY[w, u]]],
                    p3 -> member[pair[u, v], composite[Id, w]],
                    p4 -> equal[v, A[image[w, singleton[u]]]],
                    p5 -> equal[w, composite[Id, w]],
                    p6 -> member[pair[u, v], w]}]]

Out[32]= or[equal[v, A[image[w, singleton[u]]]],
           not[FUNCTION[w]], not[member[pair[u, v], w]]] == True

In[33]:= or[equal[v_, A[image[w_, singleton[u_]]]],
           not[FUNCTION[w_]], not[member[pair[u_, v_], w_]]] := True
```

This result is applied to the case of integer addition:

```
In[34]:= SubstTest[implies, and[FUNCTION[w], member[pair[u, v], w]], equal[v, APPLY[w, u]],
          {u -> pair[plus[x], plus[y]], v -> plus[natadd[x, y]], w -> INTADD}]

Out[34]= or[equal[intadd[plus[x], plus[y]], plus[natadd[x, y]]],
           not[member[x, omega]], not[member[y, omega]]] == True

In[35]:= or[equal[intadd[plus[x_], plus[y_]], plus[natadd[x_, y_]]],
           not[member[x_, omega]], not[member[y_, omega]]] := True
```

Conversely:

```
In[36]:= SubstTest[implies, and[equal[u, v], equal[v, w]], equal[u, w],
             {u -> plus[natadd[x, y]], v -> intadd[plus[x], plus[y]], w -> V}]

Out[36]= or[and[member[x, omega], member[y, omega]],
           not[equal[intadd[plus[x], plus[y]], plus[natadd[x, y]]]]] == True

In[37]:= or[and[member[x_, omega], member[y_, omega]],
           not[equal[intadd[plus[x_], plus[y_]], plus[natadd[x_, y_]]]]] := True

In[38]:= equiv[equal[intadd[plus[x], plus[y]], plus[natadd[x, y]]],
              and[member[x, omega], member[y, omega]]]

Out[38]= True
```

This may be added as a rewrite rule:

```
In[39]:= equal[intadd[plus[x_], plus[y_]], plus[natadd[x_, y_]]] :=
          and[member[x, omega], member[y, omega]]
```