distributive law

Johan G. F. Belinfante
2002 August 27

```
<< goedel52.p16; << tools.m
```

```
:Package Title: goedel52.p16         2002 August 27 at 2:00 p.m.

It is now:  2002 Aug 27 at 21:34

Loading Simplification Rules

TOOLS.M                   Revised 2002 August 22

weightlimit = 40
```

■ summary

In this notebook, one of the distributive laws of arithmetic is derived. An interesting feature is that the rule does not need any explicit restrictions of the variables to natural numbers. When any of the three variables fails to be a natural number, the rule reduces to the tautology **V = V**.

■ some preliminary simplification rules

Since

```
equal[intersection[omega, singleton[natmul[x, y]]], singleton[natmul[x, y]]]
```

```
True
```

one is justified in adding the rule:

```
intersection[omega, singleton[natmul[x_, y_]]] := singleton[natmul[x, y]]
```

There is already a similar rule for **natadd**, so it is not necessary to add it.

■ commute rule

The rule for powers of a composite of two commuting factors is an important ingredient in our derivation:

```
SubstTest[implies, commute[u, v], equal[image[power[composite[u, v]], singleton[w]],
        composite[image[power[u], singleton[w]], image[power[v], singleton[w]]]],
    {u -> composite[NATADD, RIGHT[x]], v -> composite[NATADD, RIGHT[y]]}]
```

```
equal[composite[image[power[composite[NATADD, RIGHT[x]]], singleton[w]],
   image[power[composite[NATADD, RIGHT[y]]], singleton[w]]],
  image[power[composite[NATADD, RIGHT[natadd[x, y]]]], singleton[w]]] == True
```

```
composite[image[power[composite[NATADD, RIGHT[x_]]], singleton[w_]],
  image[power[composite[NATADD, RIGHT[y_]]], singleton[w_]]] :=
 image[power[composite[NATADD, RIGHT[natadd[x, y]]]], singleton[w]]
```

■ simplification rule

```
Map[image[#, singleton[w]] &,
 SubstTest[iterate, composite[NATADD, RIGHT[z]], intersection[
   image[V, intersection[omega, singleton[z]]], singleton[0]], z -> natadd[x, y]]]

intersection[image[V, intersection[omega, singleton[x]]],
  image[V, intersection[omega, singleton[y]]],
  image[iterate[composite[NATADD, RIGHT[natadd[x, y]]], singleton[0]], singleton[w]]] ==
 singleton[natmul[natadd[x, y], w]]

intersection[image[V, intersection[omega, singleton[x_]]],
  image[V, intersection[omega, singleton[y_]]],
  image[iterate[composite[NATADD, RIGHT[natadd[x_, y_]]], singleton[0]],
   singleton[w_]]] := singleton[natmul[natadd[x, y], w]]
```

■ iterate uniqueness argument

The uniqueness theorem for **iterate** is used to derive this fact:

```
SubstTest[implies, equal[composite[u, w], composite[w, SUCC]],
 equal[composite[w, id[omega]], iterate[u, image[w, singleton[0]]]],
{    u -> composite[NATADD, RIGHT[x]], w -> composite[NATADD, RIGHT[y], NATMUL, LEFT[x]]}]

equal[composite[id[image[V, intersection[omega, singleton[x]]]],
   iterate[composite[NATADD, RIGHT[x]], intersection[omega, singleton[y]]]],
  composite[NATADD, RIGHT[y], NATMUL, LEFT[x]]] == True

composite[id[image[V, intersection[omega, singleton[x_]]]],
  iterate[composite[NATADD, RIGHT[x_]], intersection[omega, singleton[y_]]]] :=
 composite[NATADD, RIGHT[y], NATMUL, LEFT[x]]
```

The intersection with **omega** can be eliminated:

```
SubstTest[composite, id[image[V, intersection[omega, singleton[x]]]],
 iterate[composite[NATADD, RIGHT[x]], intersection[omega, singleton[z]]],
    z -> natmul[u, v]]

composite[id[image[V, intersection[omega, singleton[x]]]],
  iterate[composite[NATADD, RIGHT[x]], singleton[natmul[u, v]]]] ==
 composite[NATADD, RIGHT[natmul[u, v]], NATMUL, LEFT[x]]

composite[id[image[V, intersection[omega, singleton[x_]]]],
  iterate[composite[NATADD, RIGHT[x_]], singleton[natmul[u_, v_]]]] :=
 composite[NATADD, RIGHT[natmul[u, v]], NATMUL, LEFT[x]]
```

The **ImageComp** test is applied to this:

```
ImageComp[id[image[V, intersection[omega, singleton[x]]]],
  iterate[composite[NATADD, RIGHT[x]], singleton[natmul[u, v]]], singleton[w]] // Reverse

intersection[image[V, intersection[omega, singleton[x]]],
  image[iterate[composite[NATADD, RIGHT[x]], singleton[natmul[u, v]]], singleton[w]]] ==
 singleton[natadd[natmul[u, v], natmul[x, w]]]
```

```
intersection[image[V, intersection[omega, singleton[x_]]],
  image[iterate[composite[NATADD, RIGHT[x_]], singleton[natmul[u_, v_]]],
   singleton[w_]]] := singleton[natadd[natmul[u, v], natmul[x, w]]]
```

■ Some more simplifications

We are essentially done, but to get a pretty final formula, the following simplification rules are useful:

```
equal[union[complement[image[V, intersection[omega, singleton[x]]]],
  natmul[natadd[x, y], w]],
    natmul[natadd[x, y], w]]
```

```
True
```

```
union[complement[image[V, intersection[omega, singleton[x_]]]],
  natmul[natadd[x_, y_], w_]] := natmul[natadd[x, y], w]
```

```
equal[union[complement[image[V, intersection[omega, singleton[w]]]],
  natadd[natmul[x, w], natmul[y, w]]],
    natadd[natmul[x, w], natmul[y, w]]]
```

```
True
```

```
union[complement[image[V, intersection[omega, singleton[w_]]]],
  natadd[natmul[x_, w_], natmul[y_, w_]]] := natadd[natmul[x, w], natmul[y, w]]
```

```
equal[union[complement[image[V, intersection[omega, singleton[x]]]],
  natadd[natmul[x, w], natmul[y, w]]],
    natadd[natmul[x, w], natmul[y, w]]]
```

```
True
```

```
union[complement[image[V, intersection[omega, singleton[x_]]]],
  natadd[natmul[x_, w_], natmul[y_, w_]]] := natadd[natmul[x, w], natmul[y, w]]
```

■ The last step

```
Map[A[intersection[image[V, intersection[omega, singleton[x]]], #]] &,
 ImageComp[image[power[composite[NATADD, RIGHT[x]]], singleton[w]],
  image[power[composite[NATADD, RIGHT[y]]], singleton[w]],
    intersection[singleton[0], image[V, intersection[omega, singleton[y]]]]]]
```

```
natmul[natadd[x, y], w] == natadd[natmul[x, w], natmul[y, w]]
```

It is unclear how best to orient this rule. The following is tentative:

```
natmul[natadd[x_, y_], z_] := natadd[natmul[x, z], natmul[y, z]]
```