Quaife's corollary of (LDDEF3)

Johan G. F. Belinfante
2007 April 5

```
In[1]:= SetDirectory["l:"]; << goedel92.04a; << tools.m

        :Package Title: goedel92.04a        2007 April 4 at 9:25 p.m.

        It is now:  2007 Apr 5 at 16:23

        Loading Simplification Rules

        TOOLS.M                       Revised 2007 March 25

        weightlimit = 40
```

summary

Quaife defined a predicate **LD[x, y, z]**, which can be interpreted in the **GOEDEL** program as the statement **member[z, ld[x, y]]**, where the set of linear differences **ld[x, y]** is given by

```
In[2]:= image[image[inverse[NATADD], image[DIV, set[x]]], image[DIV, set[y]]]

Out[2]= ld[x, y]
```

In this notebook it is shown that with this interpretation, the corollary of Quaife's definition **(LDDEF3)** holds.

```
In[3]:= "Art Quaife, Automated Development of Fundamental
           Mathematical Theories, Appendix 3. Theorems Proved in Peano's
           Arithmetic, Kluwer Academic Publishers, Dordrecht, 1992. Cf. p. 200";
```

The corollary in question involves floored substraction, here called **monus**. The main idea behind the derivation is that **monus[x, y]** is either equal to **0** or else **natsub[x, y]**. Since a similar result for **natsub** is already available, one only needs to consider the case of **0**. This special case is the first clause of Quaife's Theorem **(LD4)**.

Quaife's theorem (LD4a)

Theorem. Analog of Quaife's Theorem **(LD4a)**.

```
In[4]:= member[0, ld[x, y]] // AssertTest

Out[4]= member[0, ld[x, y]] == and[member[x, omega], member[y, omega]]

In[5]:= member[0, ld[x_, y_]] := and[member[x, omega], member[y, omega]]
```

simplification rules

Lemma.

```
In[6]:= Map[or[member[x, omega], #] &,
          SubstTest[implies, member[w, t], not[empty[t]], t → ld[x, y]]] // Reverse
```

```
Out[6]= or[member[x, omega], not[member[w, ld[x, y]]]] == True
```

```
In[7]:= or[member[x_, omega], not[member[w_, ld[x_, y_]]]] := True
```

Lemma.

```
In[8]:= Map[or[member[y, omega], #] &,
          SubstTest[implies, member[w, t], not[empty[t]], t → ld[x, y]]] // Reverse
```

```
Out[8]= or[member[y, omega], not[member[w, ld[x, y]]]] == True
```

```
In[9]:= or[member[y_, omega], not[member[w_, ld[x_, y_]]]] := True
```

Simplification rule.

```
In[10]:= equiv[and[member[w, ld[x, y]], member[y, omega]], member[w, ld[x, y]]]
```

```
Out[10]= True
```

```
In[11]:= and[member[w_, ld[x_, y_]], member[y_, omega]] := member[w, ld[x, y]]
```

a lemma

If a natural number **x** is less than another number **y**, then **y** cannot also be less than or equal to **x**.

```
In[12]:= and[member[x, y], not[subclass[x, y]], member[y, omega]] // NotNotTest
```

```
Out[12]= and[member[x, y], member[y, omega], not[subclass[x, y]]] == False
```

```
In[13]:= and[member[x_, y_], member[y_, omega], not[subclass[x_, y_]]] := False
```

Specializing to the case of products, one finds:

```
In[14]:= SubstTest[and, member[s, t], member[t, omega], not[subclass[s, t]],
            {s -> natmul[v, y], t -> natmul[u, x]}] // Reverse // MapNotNot
```

```
Out[14]= and[member[u, omega], member[x, omega], member[natmul[v, y], natmul[u, x]],
           not[subclass[natmul[v, y], natmul[u, x]]]] == False
```

```
In[15]:= (% /. {u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

To clean this up, another lemma is needed:

```
In[16]:= Map[or[member[x, omega], #] &,
           SubstTest[implies, not[subclass[w, t]], not[equal[V, t]], t → natmul[x, y]]] // Reverse

Out[16]= or[member[x, omega], subclass[w, natmul[x, y]]] == True

In[17]:= or[member[x_, omega], subclass[w_, natmul[x_, y_]]] := True
```

The cleaned-up version is:

```
In[18]:= SubstTest[and, implies[p1, p3], implies[p1, p4],
            and[p1, p2], {p1 → not[subclass[natmul[v, y], natmul[u, x]]],
             p2 -> member[natmul[v, y], natmul[u, x]], p3 → member[u, omega],
             p4 → member[x, omega]}] // Reverse // MapNotNot

Out[18]= and[member[natmul[v, y], natmul[u, x]],
           not[subclass[natmul[v, y], natmul[u, x]]]] == False

In[19]:= and[member[natmul[v_, y_], natmul[u_, x_]],
           not[subclass[natmul[v_, y_], natmul[u_, x_]]]] := False
```

Corollary (LD3cor)

Lemma. This is an application of Theorem **(LD4a)**.

```
In[20]:= SubstTest[implies, and[equal[0, t], member[0, z]], member[t, z],
            {t -> nat[natsub[natmul[u, x], natmul[v, y]]], z -> ld[x, y]}] // Reverse // MapNotNot

Out[20]= or[and[member[u, omega], member[natmul[v, y], natmul[u, x]]],
           member[nat[natsub[natmul[u, x], natmul[v, y]]], ld[x, y]],
           not[member[x, omega]], not[member[y, omega]]] == True

In[21]:= (% /. {u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. An existing membership rule involving **natsub** is specialized to the case that **natsub** and **monus** are equal.

```
In[22]:= SubstTest[implies, and[equal[w, t], member[w, z]], member[t, z],
            {t → monus[natmul[u, x], natmul[v, y]],
             w -> natsub[natmul[u, x], natmul[v, y]], z -> ld[x, y]}] // Reverse // MapNotNot

Out[22]= or[member[nat[natsub[natmul[u, x], natmul[v, y]]], ld[x, y]],
           not[member[u, omega]], not[member[v, omega]], not[member[x, omega]],
           not[member[y, omega]], not[subclass[natmul[v, y], natmul[u, x]]]] == True

In[23]:= (% /. {u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

Combining the above lemmas yields:

```
In[24]:= SubstTest[and, implies[p, r], implies[q, r],
           {p → and[member[u, omega], member[v, omega], member[x, omega],
              member[y, omega], subclass[natmul[v, y], natmul[u, x]]],
            q -> or[and[member[x, omega], member[y, omega], not[member[u, omega]]],
              and[member[x, omega], member[y, omega], not[member[natmul[v, y], natmul[u, x]]]]],
            r -> member[nat[natsub[natmul[u, x], natmul[v, y]]], ld[x, y]]}] // MapNotNot

Out[24]= or[member[nat[natsub[natmul[u, x], natmul[v, y]]], ld[x, y]],
           not[member[x, omega]], not[member[y, omega]]] == True

In[25]:= (% /. {u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. Analog of Quaife's third clause of **(LDDEF3)**.

```
In[26]:= equiv[member[nat[natsub[natmul[u, x], natmul[v, y]]], ld[x, y]],
            and[member[x, omega], member[y, omega]]] // not // not

Out[26]= True

In[27]:= member[nat[natsub[natmul[u_, x_], natmul[v_, y_]]], ld[x_, y_]] :=
          and[member[x, omega], member[y, omega]]
```