an upper bound for factorials

Johan G. F. Belinfante
2007 January 15

```
In[1]:= SetDirectory["l:"]; << goedel89.14a; << tools.m

        :Package Title: goedel89.14a        2007 January 14 at 10:30 a.m.

        It is now:  2007 Jan 15 at 12:48

        Loading Simplification Rules

        TOOLS.M                       Revised 2007 January 7

        weightlimit = 40
```

summary

In this notebook it is shown that $\mathbf{n! \leq n^n}$. The proof is by induction on **n**.

recursion relation for FACTORIAL

The following orientation for the recursion relation for the factorial function is convenient here.

```
In[2]:= ApComp[FACTORIAL, SUCC, x] // Reverse

Out[2]= natadd[APPLY[FACTORIAL, x], natmul[x, APPLY[FACTORIAL, x]]] ==
         union[APPLY[FACTORIAL, succ[x]], complement[image[V, set[x]]]]

In[3]:= natadd[APPLY[FACTORIAL, x_], natmul[x_, APPLY[FACTORIAL, x_]]] :=
         union[APPLY[FACTORIAL, succ[x]], complement[image[V, set[x]]]]
```

the induction class

The following temporary abbreviation is introduced for the class to which induction will be applied.

```
In[4]:= ind := fix[composite[inverse[DUP], inverse[NATEXP], S, FACTORIAL]]
```

A membership rule for this class is needed.

```
In[5]:= member[x, ind] // AssertTest

Out[5]= member[x, fix[composite[inverse[DUP], inverse[NATEXP], S, FACTORIAL]]] ==
         and[member[x, omega], subclass[APPLY[FACTORIAL, x], natexp[x, x]]]
```

```
In[6]:= member[x_, fix[composite[inverse[DUP], inverse[NATEXP], S, FACTORIAL]]] :=
         and[member[x, omega], subclass[APPLY[FACTORIAL, x], natexp[x, x]]]
```

Comment. The base case is trivial:

```
In[7]:= member[0, ind]

Out[7]= True
```

---

lemmas

Lemma 1.

```
In[8]:= Map[not, SubstTest[implies, subclass[nat[x], nat[y]], subclass[
            natexp[nat[x], nat[z]], natexp[nat[y], nat[z]]], y → succ[nat[x]]] // Reverse]

Out[8]= member[natexp[succ[nat[x]], nat[z]], natexp[nat[x], nat[z]]] == False

In[9]:= member[natexp[succ[nat[x_]], nat[z_]], natexp[nat[x_], nat[z_]]] := False
```

Lemma 2.

```
In[10]:= Map[not,
           SubstTest[subclass, natmul[nat[u], nat[v]], natmul[nat[u], nat[w]], {u → succ[nat[x]],
             v → APPLY[FACTORIAL, nat[x]], w → natexp[succ[nat[x]], nat[x]]}]] // Reverse

Out[10]= member[natadd[natexp[succ[nat[x]], nat[x]],
            natmul[nat[x], natexp[succ[nat[x]], nat[x]]]], APPLY[FACTORIAL, succ[nat[x]]]] ==
          member[natexp[succ[nat[x]], nat[x]], APPLY[FACTORIAL, nat[x]]]

In[11]:= member[natadd[natexp[succ[nat[x_]], nat[x_]], natmul[nat[x_],
             natexp[succ[nat[x_]], nat[x_]]]], APPLY[FACTORIAL, succ[nat[x_]]]] :=
          member[natexp[succ[nat[x]], nat[x]], APPLY[FACTORIAL, nat[x]]]
```

The induction step.

```
In[12]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
           {u → APPLY[FACTORIAL, nat[x]],
            v → natexp[nat[x], nat[x]], w → natexp[succ[nat[x]], nat[x]]}] // Reverse

Out[12]= or[member[natexp[nat[x], nat[x]], APPLY[FACTORIAL, nat[x]]],
           not[member[natexp[succ[nat[x]], nat[x]], APPLY[FACTORIAL, nat[x]]]]] == True

In[13]:= (% /. x → x_) /. Equal → SetDelayed
```

Restatement.

```
In[14]:= implies[member[nat[x], ind], member[succ[nat[x]], ind]]

Out[14]= True
```

applying induction

To apply induction, the variable needs to be eliminated from the induction step. Before one can do that, the the **nat** wrappers need to be removed.

```
In[15]:= SubstTest[implies, and[equal[x, nat[t]], member[x, ind]],
           member[succ[x], ind], t → x] // Reverse

Out[15]= or[not[member[x, omega]],
           not[subclass[APPLY[FACTORIAL, x], natexp[x, x]]], subclass[APPLY[FACTORIAL, succ[x]],
            natadd[natexp[succ[x], x], natmul[x, natexp[succ[x], x]]]]] == True

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

Now the variable **x** can be eliminated.

```
In[17]:= Map[equal[V, #] &, SubstTest[class, x,
           implies[and[member[x, omega], member[x, y]], member[succ[x], y]], y → ind]]

Out[17]= subclass[intersection[omega,
            image[SUCC, fix[composite[inverse[DUP], inverse[NATEXP], S, FACTORIAL]]]],
           fix[composite[inverse[DUP], inverse[NATEXP], S, FACTORIAL]]] == True

In[18]:= % /. Equal → SetDelayed
```

At this point one can apply mathematical induction. This yields a variable-free version of the upper bound theorem.

```
In[19]:= SubstTest[implies, INDUCTIVE[w],
           subclass[omega, w], w → intersection[omega, ind]] // Reverse

Out[19]= subclass[omega, fix[composite[inverse[DUP], inverse[NATEXP], S, FACTORIAL]]] == True

In[20]:= % /. Equal → SetDelayed
```

Main Theorem. The result derived becomes more comprehensible if one reintroduces a variable.

```
In[21]:= SubstTest[implies, and[member[x, y], subclass[y, z]], member[x, z],
           {y → omega, z -> fix[composite[inverse[DUP], inverse[NATEXP], S, FACTORIAL]]}] //
          Reverse

Out[21]= or[not[member[x, omega]], subclass[APPLY[FACTORIAL, x], natexp[x, x]]] == True

In[22]:= or[not[member[x_, omega]], subclass[APPLY[FACTORIAL, x_], natexp[x_, x_]]] := True
```

variable-free formulations

The variable-free version of the upper bound theorem can also be improved upon. One needs an inclusion in the reverse direction:

```
In[23]:= SubstTest[subclass, fix[composite[x, y]], domain[y], y → FACTORIAL] // Reverse

Out[23]= subclass[fix[composite[x, FACTORIAL]], omega] == True

In[24]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem.

```
In[25]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> omega, v -> fix[composite[inverse[DUP], inverse[NATEXP], S, FACTORIAL]]}]

Out[25]= equal[omega, fix[composite[inverse[DUP], inverse[NATEXP], S, FACTORIAL]]] == True

In[26]:= fix[composite[inverse[DUP], inverse[NATEXP], S, FACTORIAL]] := omega
```

The **fix** constructor can be eliminated. To do so, a lemma is needed:

```
In[32]:= intersection[omega, complement[
            fix[composite[inverse[FACTORIAL], inverse[S], NATEXP, DUP]]]] // InvertFixTest

Out[32]= intersection[omega,
           complement[fix[composite[inverse[FACTORIAL], inverse[S], NATEXP, DUP]]]] == 0

In[33]:= % /. Equal → SetDelayed
```

Theorem. (A restatement without the **fix** constructor.)

```
In[35]:= Map[empty, SubstTest[composite, x, id[domain[x]],
           x -> dif[FACTORIAL, composite[inverse[S], NATEXP, DUP]]]]

Out[35]= subclass[composite[NATEXP, DUP, inverse[FACTORIAL]], S] == True

In[36]:= subclass[composite[NATEXP, DUP, inverse[FACTORIAL]], S] := True
```

Corollary 1. Another equivalent version.

```
In[38]:= subclass[FACTORIAL, composite[inverse[S], NATEXP, DUP]] // AssertTest

Out[38]= subclass[FACTORIAL, composite[inverse[S], NATEXP, DUP]] == True

In[39]:= subclass[FACTORIAL, composite[inverse[S], NATEXP, DUP]] := True
```

Lemma.

```
In[47]:= SubstTest[composite, x, id[domain[x]],
          x -> dif[composite[NATEXP, DUP], composite[S, FACTORIAL]]]

Out[47]= intersection[composite[NATEXP, DUP], composite[complement[S], FACTORIAL]] == 0

In[48]:= % /. Equal → SetDelayed
```

Corollary 2.

```
In[50]:= SubstTest[empty, dif[u, v], {u -> composite[NATEXP, DUP], v -> composite[S, FACTORIAL]}]
Out[50]= subclass[composite[NATEXP, DUP], composite[S, FACTORIAL]] == True

In[51]:= subclass[composite[NATEXP, DUP], composite[S, FACTORIAL]] := True
```