Otter's proof of Theorem ON−SUC−7

Johan G. F. Belinfante
revised 2003 January 21

```
<< goedel52.r05; << tools.m

:Package Title: goedel52.r05          2003 January 20 at 4:45 p.m.

It is now:  2003 Jan 20 at 17:35

Loading Simplification Rules

TOOLS.M                       Revised 2002 December 27

weightlimit = 40
```

■ summary

This notebook contains a derivation of Theorem **ON−SUC−7** as it was proved using **Otter** in 1997 June 6. The only purpose of reproducing the old **Otter** proof is just to illustrate how one can use the **GOEDEL** program to manually derive theorems in more or less the same way that one would do it using an automated theorem prover. A second version of **Otter**'s proof is presented which avoids the use of variables.

■ Otter's proof of Theorem ON−SUC−7

The main ingredient of **Otter**'s proof is Theorem **ON−SUC−6**, which states that limit ordinals are successor−invariant. The **GOEDEL** program already has a variable−freeversion of this theorem, so we just need to restore the variables:

```
SubstTest[implies, and[member[x, y], subclass[y, z]], member[x, z],
    {y -> intersection[OMEGA, fix[BIGCUP]], z -> invar[SUCC]}] // MapNotNot

or[not[equal[x, U[x]]], not[member[x, OMEGA]], subclass[image[SUCC, x], x]] == True

or[not[equal[x_, U[x_]]], not[member[x_, OMEGA]], subclass[image[SUCC, x_], x_]] := True
```

The rest of the proof consists of using ordinary mathematical induction, and Corollary **ON−1−B**.

```
Map[not, SubstTest[and, implies[p1, or[p3, p4]],
  implies[and[p1, p2], p5], implies[and[p4, p5], p6],
        not[implies[and[p1, p2], or[p3, p6]]],
{p1 -> member[x, OMEGA], p2 -> equal[U[x], x], p3 -> equal[0, x], p4 -> member[0, x],
            p5 -> invariant[SUCC, x], p6 -> subclass[omega, x]}]]

or[equal[0, x], not[equal[x, U[x]]], not[member[x, OMEGA]], subclass[omega, x]] == True
```

This is Theorem **ON−SUC−7**.

```
or[equal[0, x_], not[equal[x_, U[x_]]],
  not[member[x_, OMEGA]], subclass[omega, x_]] := True
```

■ a variable−freeproof of Theorem ON−SUC−7

Ordinary mathematical induction can be stated in variable−freeform as follows:

```
subclass[intersection[invar[SUCC], image[E, singleton[0]]], image[S, singleton[omega]]]

True
```

The theorem that limit ordinals are successor invariant is also available in variable−freeform:

```
subclass[intersection[OMEGA, fix[BIGCUP]], invar[SUCC]]

True
```

The only non−obviousstep required is this reformulation of the variable−freeformulation of mathematical induction.

```
subclass[invar[SUCC],
  union[image[S, singleton[omega]], P[complement[singleton[0]]]]] // AssertTest

subclass[invar[SUCC],
  union[image[S, singleton[omega]], P[complement[singleton[0]]]]] == True

subclass[invar[SUCC],
  union[image[S, singleton[omega]], P[complement[singleton[0]]]]] := True
```

The rest of the proof just combines the above facts, using the transitivity of inclusion.

```
SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
    {u -> intersection[OMEGA, fix[BIGCUP]], v -> invar[SUCC],
        w -> union[image[S, singleton[omega]], complement[image[E, singleton[0]]]]}]

subclass[intersection[OMEGA, fix[BIGCUP]],
  union[image[S, singleton[omega]], P[complement[singleton[0]]]]] == True
```

This is a variable−freeform of Theorem **ON−SUC−7**.

```
subclass[intersection[OMEGA, fix[BIGCUP]],
  union[image[S, singleton[omega]], P[complement[singleton[0]]]]] := True
```

The following corollary follows immediately:

```
SubstTest[subclass, dif[u, v], w,
    {u -> intersection[OMEGA, fix[BIGCUP]], v -> complement[image[E, singleton[0]]],
        w -> image[S, singleton[omega]]}]

subclass[omega,
  A[intersection[OMEGA, complement[P[complement[singleton[0]]], fix[BIGCUP]]]] == True

subclass[omega,
  A[intersection[OMEGA, complement[P[complement[singleton[0]]], fix[BIGCUP]]]] := True
```

This inclusion can in fact be strengthened to an equation, but we do not pursue this further here.

■ variable−freeequational reformulation of Corollary ON−1−B

The variable−freeproof of Theorem **ON−SUC−7** avoids Corollary **ON−1−B** In this section we derive two variable−free equations related to Corollary **ON−1−B** The present version of the **GOEDEL** program contains a version of Corollary **ON−1−B**with variables, so applying **Normality** brings variables into play and yields the variable−freeversion..

```
Map[equal[V, #] &,
 dif[OMEGA, union[singleton[0], image[E, singleton[0]]]] // complement // Normality]
```

```
subclass[intersection[OMEGA, P[complement[singleton[0]]]], singleton[0]] == True
```

```
subclass[intersection[OMEGA, P[complement[singleton[0]]]], singleton[0]] := True
```

The inclusion holds in both directions, so we get an equational version of Corollary **ON−1−B**.

```
SubstTest[and, subclass[u, v], subclass[v, u],
    {u -> intersection[OMEGA, P[complement[singleton[0]]]], v -> singleton[0]}]
```

```
True == equal[intersection[OMEGA, P[complement[singleton[0]]]], singleton[0]]
```

```
Equal[intersection[OMEGA, P[complement[singleton[0]]]], singleton[0]]
```

```
intersection[OMEGA, P[complement[singleton[0]]]] == singleton[0]
```

```
intersection[OMEGA, P[complement[singleton[0]]]] := singleton[0]
```

Another variable−freeequation can be derived from Corollary **ON−1−B** as follows.

```
member[0, A[intersection[OMEGA, complement[singleton[0]]]]] // AssertTest
```

```
member[0, A[intersection[OMEGA, complement[singleton[0]]]]] == True
```

```
member[0, A[intersection[OMEGA, complement[singleton[0]]]]] := True
```

```
SubstTest[implies, member[x, y], subclass[A[y], x],
    {x -> singleton[0], y -> intersection[OMEGA, complement[singleton[0]]]}]
```

```
subclass[A[intersection[OMEGA, complement[singleton[0]]]], singleton[0]] == True
```

```
subclass[A[intersection[OMEGA, complement[singleton[0]]]], singleton[0]] := True
```

Since the inclusion holds in both directions, we get an equation which says that **singleton[0]** is the least ordinal greater than **0**.

```
SubstTest[and, subclass[u, v], subclass[v, u],
    {u -> A[intersection[OMEGA, complement[singleton[0]]]], v -> singleton[0]}]
```

```
True == equal[A[intersection[OMEGA, complement[singleton[0]]]], singleton[0]]
```

```
A[intersection[OMEGA, complement[singleton[0]]]] := singleton[0]
```