second form of induction

Johan G. F. Belinfante
2002 October 7

```
<< goedel52.p77; << tools.m
```

```
:Package Title: goedel52.p77          2002 October 6 at 9:12 p.m.

It is now:  2002 Oct 7 at 10:1

Loading Simplification Rules

TOOLS.M                   Revised 2002 August 30

weightlimit = 40
```

■ summary

The first form of mathematical induction says:

```
implies[and[member[0, x], subclass[image[SUCC, x], x]], subclass[omega, x]]
```

```
True
```

In this notebook, this first form of mathematical induction is used to derive a second form of induction. The derivation follows loosely the **Otter** proof found several years ago, but the details are very different. The **Otter** proof uses Skolem functions extensively, but the present derivation does not. On the other hand some use is made of formulas such as the following, which were not available for the **Otter** proof.

```
composite[inverse[E], id[omega], SUCC]
```

```
composite[id[omega], inverse[S], id[omega]]
```

■ a key lemma in the Otter proof

One source of annoyance in the derivation is the continual intervention of the following rewrite rule:

```
subclass[x, P[y]]
```

```
subclass[U[x], y]
```

One could temporarily remove this rule to avoid this nuisance, but then one would need to use **assert** whenever one does want the rule to be used. For example, the following key lemma used in the **Otter** proof is completely rewritten by the **GOEDEL** program:

```
implies[subclass[intersection[omega, P[x]], x],
    subclass[image[SUCC, intersection[omega, P[x]]], intersection[omega, P[x]]]]

or[not[subclass[intersection[omega, P[x]], x]],
 subclass[U[intersection[omega, image[SUCC, P[x]]]], x]]
```

This lemma will be derived below.

■ the first step

The following first step also deals with the annoying (but useful) rewrite rule about power classes.

```
SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
    {u -> intersection[omega, image[inverse[S], intersection[omega, P[x]]]],
        v -> image[inverse[S], intersection[omega, P[x]]], w -> P[x]}]

subclass[U[intersection[omega, image[inverse[S], intersection[omega, P[x]]]]], x] == True

subclass[
  U[intersection[omega, image[inverse[S], intersection[omega, P[x_]]]]], x_] := True
```

■ derivation of a key lemma

The following needs to be broken up into two rules.

```
Map[subclass[#, intersection[omega, P[x]]] &,
 ImageComp[inverse[E], composite[id[omega], SUCC], P[x]]]

True == and[subclass[U[intersection[omega, image[SUCC, P[x]]]], omega],
  subclass[U[U[intersection[omega, image[SUCC, P[x]]]]], x]]
```

Here is one:

```
Map[or[subclass[U[intersection[omega, image[SUCC, P[x]]]], omega], #] &, %] // Reverse

subclass[U[intersection[omega, image[SUCC, P[x]]]], omega] == True

subclass[U[intersection[omega, image[SUCC, P[x_]]]], omega] := True
```

Here is the other:

```
Map[subclass[#, intersection[omega, P[x]]] &,
  ImageComp[inverse[E], composite[id[omega], SUCC], P[x]]] // Reverse

subclass[U[U[intersection[omega, image[SUCC, P[x]]]]], x] == True

subclass[U[U[intersection[omega, image[SUCC, P[x_]]]]], x_] := True
```

This is just the transformed version of the key lemma from the **Otter** proof:

```
SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
    {u -> U[intersection[omega, image[SUCC, P[x]]]],
  v -> intersection[omega, P[x]], w -> x}]
```

```
or[not[subclass[intersection[omega, P[x]], x]],
  subclass[U[intersection[omega, image[SUCC, P[x]]]], x]] == True
```

```
or[not[subclass[intersection[omega, P[x_]], x_]],
  subclass[U[intersection[omega, image[SUCC, P[x_]]]], x_]] := True
```

■ the final steps

The idea is now to derive the second form of induction from the first form of induction as follows:

```
SubstTest[implies, and[member[0, w], subclass[image[SUCC, w], w]],
 subclass[omega, w], w -> intersection[omega, P[x]]]
```

```
or[not[subclass[U[intersection[omega, image[SUCC, P[x]]]], x]], subclass[omega, x]] ==
 True
```

```
or[not[subclass[U[intersection[omega, image[SUCC, P[x_]]]], x_]],
  subclass[omega, x_]] := True
```

The final step just uses the transitivity of inclusion.

```
Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
        {p1 -> subclass[intersection[omega, P[x]], x],
            p2 -> subclass[U[intersection[omega, image[SUCC, P[x]]]], x],
        p3 -> subclass[omega, x]}]]
```

```
or[not[subclass[intersection[omega, P[x]], x]], subclass[omega, x]] == True
```

This is the second form of mathematical induction:

```
or[not[subclass[intersection[omega, P[x_]], x_]], subclass[omega, x_]] := True
```