

the ADJOIN formula

Johan G. F. Belinfante
2002 May 24

```
<< goedel52.n96; << tools.m

:Package Title: GOEDEL52.N96          2002 May 24 at 1:55 p.m.

It is now: 2002 May 24 at 17:30

Loading Simplification Rules

TOOLS.M          Revised 2002 May 22

weightlimit = 40
```

■ Introduction

This notebook illustrates a number of powerful techniques that are generally useful, including the elimination of variables by reification, the use of induction, and the replacement of conditional equations by unconditional equations.

■ ADJOIN[x] and CUT[x] = IMAGE[id[x]]

The specific purpose of this notebook is to derive two formulas for the restriction of **IMAGE[SUCC]** to **omega**. The two can be related using the following identities:

```
composite[IMAGE[id[complement[x]]], ADJOIN[x]] // VSNormality

composite[IMAGE[id[complement[x]]], ADJOIN[x]] ==
  composite[id[image[v, singleton[x]]], IMAGE[id[complement[x]]]]

composite[IMAGE[id[complement[x_]]], ADJOIN[x_]] :=
  composite[id[image[v, singleton[x]]], IMAGE[id[complement[x]]]]

composite[ADJOIN[x], IMAGE[id[complement[x]]]] // VSNormality

composite[ADJOIN[x], IMAGE[id[complement[x]]]] == ADJOIN[x]

composite[ADJOIN[x_], IMAGE[id[complement[x_]]]] := ADJOIN[x]
```

We recall the characterizations of these functions:

```
ADJOIN[x] == class[pair[u, v], equal[v, union[x, u]]]

True

IMAGE[id[x]] == class[pair[u, v], equal[v, intersection[x, u]]]

True
```

The connection with **image** comes from the following observation

```
intersection[x, u] == image[id[x], u]
True
```

■ two membership rules

Rule 1:

```
member[pair[x, y], composite[inverse[SUCC], z]] // AssertTest

member[pair[x, y], composite[inverse[SUCC], z]] ==
and[member[x, V], member[y, V], member[pair[x, succ[y]], z]]

member[pair[x_, y_], composite[inverse[SUCC], z_]] :=
and[member[x, V], member[y, V], member[pair[x, succ[y]], z]]
```

Rule 2:

```
member[pair[x, y], composite[z, IMAGE[SUCC]]] // AssertTest

member[pair[x, y], composite[z, IMAGE[SUCC]]] ==
and[member[x, V], member[y, V], member[pair[image[SUCC, x], y], z]]

member[pair[x_, y_], composite[z_, IMAGE[SUCC]]] :=
and[member[x, V], member[y, V], member[pair[image[SUCC, x], y], z]]
```

■ temporary successor equations

We derive two temporary rules needed to cope with successors.

```
SubstTest[implies, equal[u, v], equal[union[u, w], union[v, w]], w -> singleton[u]]
or[equal[succ[u], union[v, singleton[u]]], not[equal[u, v]]] == True

or[equal[succ[u_], union[v_, singleton[u_]]], not[equal[u_, v_]]] := True

SubstTest[implies, member[x, fix[composite[inverse[SUCC], ADJOIN[w], IMAGE[SUCC]]]],
member[succ[x], fix[composite[inverse[SUCC], ADJOIN[w], IMAGE[SUCC]]]],
w -> singleton[0]]

or[equal[succ[succ[x]], union[image[SUCC, x], pairset[0, succ[x]]]],
not[equal[succ[x], union[image[SUCC, x], singleton[0]]]], not[member[x, V]]] == True

or[equal[succ[succ[x_]], union[image[SUCC, x_], pairset[0, succ[x_]]]],
not[equal[succ[x_], union[image[SUCC, x_], singleton[0]]]], not[member[x_, V]]] := True
```

■ applying induction

We prepare for induction by eliminating a quantifier:

```

Map[equal[V, #] &, SubstTest[class, x, implies[member[x, z], member[succ[x], z]],
  z -> fix[composite[inverse[SUCC], ADJOIN[singleton[0]], IMAGE[SUCC]]]] // Reverse

subclass[image[SUCC, fix[composite[inverse[SUCC], ADJOIN[singleton[0]], IMAGE[SUCC]]]],
  fix[composite[inverse[SUCC], ADJOIN[singleton[0]], IMAGE[SUCC]]]] == True

subclass[image[SUCC, fix[composite[inverse[SUCC], ADJOIN[singleton[0]], IMAGE[SUCC]]]],
  fix[composite[inverse[SUCC], ADJOIN[singleton[0]], IMAGE[SUCC]]]] := True

```

The proof by induction just takes one step now:

```

SubstTest[implies, INDUCTIVE[z], subclass[omega, z],
  z -> fix[composite[inverse[SUCC], ADJOIN[singleton[0]], IMAGE[SUCC]]]]

subclass[omega,
  fix[composite[inverse[SUCC], ADJOIN[singleton[0]], IMAGE[SUCC]]]] == True

subclass[omega,
  fix[composite[inverse[SUCC], ADJOIN[singleton[0]], IMAGE[SUCC]]]] := True

```

The hypothesis is an abbreviation:

```

INDUCTIVE[z]

and[member[0, z], subclass[image[SUCC, z], z]]

```

■ a more readable formulation

The result obtained can be better understood by reintroducing a variable.

```

SubstTest[implies, and[member[x, y], subclass[y, z]], member[x, z],
  {y -> omega,
  z -> fix[composite[inverse[SUCC], ADJOIN[singleton[0]], IMAGE[SUCC]]]} // MapNotNot

or[equal[succ[x], union[image[SUCC, x], singleton[0]]], not[member[x, omega]]] == True

or[equal[succ[x_], union[image[SUCC, x_], singleton[0]]], not[member[x_, omega]]] := True

```

■ A formula involving CUP

The following formula was known for the special case $x = V$, but we need the general case, too.

```

composite[CUP, id[composite[SINGLETON, id[x]]], inverse[FIRST]] // VSNormality

composite[CUP, id[composite[SINGLETON, id[x]]], inverse[FIRST]] == composite[SUCC, id[x]]

composite[CUP, id[composite[SINGLETON, id[x_]]], inverse[FIRST]] :=
  composite[SUCC, id[x]]

```

■ Two Corollaries

Corollary 1:

```

Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
  {p1 -> member[x, omega],
    p2 -> equal[succ[x], union[image[SUCC, x], singleton[0]]],
    p3 -> subclass[succ[x], union[image[SUCC, x], singleton[0]]]}]]
or[not[member[x, omega]], subclass[succ[x], union[image[SUCC, x], singleton[0]]]] == True

or[not[member[x_, omega]],
  subclass[succ[x_], union[image[SUCC, x_], singleton[0]]]] := True

```

A general simplification rule.

```

Map[or[subclass[y, x], #] &,
  SubstTest[implies, equal[w, x], subclass[w, x], w -> union[y, z]]]
or[not[equal[x, union[y, z]]], subclass[y, x]] == True

or[not[equal[x_, union[y_, z_]], subclass[y_, x_]] := True

```

Corollary 2.

```

Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
  {p1 -> member[x, omega],
    p2 -> equal[succ[x], union[image[SUCC, x], singleton[0]]],
    p3 -> subclass[image[SUCC, x], succ[x]]}]
or[not[member[x, omega]], subclass[image[SUCC, x], succ[x]]] == True

or[not[member[x_, omega]], subclass[image[SUCC, x_], succ[x_]]] := True

```

■ a symdif argument

Our conditional equality can be made into an unconditional one by replacing the condition `member[x,omega]` by an intersection of each side of the equation with the class `image[V,intersection[omega,singleton[x]]`. The replacement is not automatic; what we want is:

```

equal[intersection[image[V, intersection[omega, singleton[x]]], succ[x]],
  intersection[image[V, intersection[omega, singleton[x]]],
  union[image[SUCC, x], singleton[0]]]]
or[equal[succ[x],
  union[intersection[image[SUCC, x], image[V, intersection[omega, singleton[x]]]],
  intersection[image[V, intersection[omega, singleton[x]]], singleton[0]]]],
  not[member[x, omega]]]

```

Amazingly, we get for free a logically equivalent assertion:

```

equal[0, symdif[intersection[image[V, intersection[omega, singleton[x]]], succ[x]],
  intersection[image[V, intersection[omega, singleton[x]]],
  union[image[SUCC, x], singleton[0]]]]]

```

True

Now it is easy to get the equality statement from the `symdif` assertion!

```

SubstTest[equal, 0, symdif[u, v],
  {u -> intersection[image[V, intersection[omega, singleton[x]]], succ[x]],
   v -> intersection[image[V, intersection[omega, singleton[x]]],
    union[image[SUCC, x], singleton[0]]]}] // Reverse

or[equal[succ[x],
  union[intersection[image[SUCC, x], image[V, intersection[omega, singleton[x]]]],
  intersection[image[V, intersection[omega, singleton[x]]], singleton[0]]]],
not[member[x, omega]]] == True

or[equal[succ[x_],
  union[intersection[image[SUCC, x_], image[V, intersection[omega, singleton[x_]]]],
  intersection[image[V, intersection[omega, singleton[x_]]], singleton[0]]]],
not[member[x_, omega]]] := True

```

We verify that the equality we want is now recognized as true:

```

equal[intersection[image[V, intersection[omega, singleton[x]]], succ[x]],
  intersection[image[V, intersection[omega, singleton[x]]],
  union[image[SUCC, x], singleton[0]]]]

True

```

Replacing **equal** with **Equal** yields:

```

Equal[intersection[image[V, intersection[omega, singleton[x]]], succ[x]],
  intersection[image[V, intersection[omega, singleton[x]]],
  union[image[SUCC, x], singleton[0]]]] // Reverse

union[intersection[image[SUCC, x], image[V, intersection[omega, singleton[x]]]],
  intersection[image[V, intersection[omega, singleton[x]]], singleton[0]]] ==
  intersection[image[V, intersection[omega, singleton[x]]], succ[x]]

```

The next step is to reify this formula, thereby eliminating the variable **x**. This yields a relational formula without variables:

```

Map[class[pair[x, y], member[y, #]] &, %]

union[cart[omega, singleton[0]], composite[inverse[E], IMAGE[SUCC], id[omega]]] ==
  union[composite[inverse[E], id[omega]], id[omega]]

```

Mapping with **VERTSECT** yields the **ADJOIN** formula:

```

Map[composite[VERTSECT[#], id[omega]] &, %]

composite[ADJOIN[singleton[0]], IMAGE[SUCC], id[omega]] == composite[id[omega], SUCC]

composite[ADJOIN[singleton[0]], IMAGE[SUCC], id[omega]] := composite[id[omega], SUCC]

```

■ The other formula

The other formula is obtained by transposing the **singleton[0]** to the other side of the equation. This works better if we add a simplification rule:

```

equal[composite[id[complement[singleton[0]]], SUCC], SUCC] // AssertTest

equal[SUCC, composite[id[complement[singleton[0]]], SUCC]] == True

composite[id[complement[singleton[0]]], SUCC] := SUCC

```

We actually need to **IMAGE** counterpart of this:

```
SubstTest[IMAGE,
  composite[id[complement[singleton[0]]], funpart[x], x -> SUCC] // Reverse
composite[IMAGE[id[complement[singleton[0]]], IMAGE[SUCC]] == IMAGE[SUCC]

composite[IMAGE[id[complement[singleton[0]]], IMAGE[SUCC]] := IMAGE[SUCC]
```

The desired transposition is now accomplished using associativity of composition:

```
Assoc[IMAGE[id[complement[singleton[0]]],
  ADJOIN[singleton[0], composite[IMAGE[SUCC], id[omega]]] // Reverse
composite[IMAGE[SUCC], id[omega]] ==
  composite[IMAGE[id[complement[singleton[0]]], id[omega], SUCC]

composite[IMAGE[SUCC], id[omega]] :=
  composite[IMAGE[id[complement[singleton[0]]], id[omega], SUCC]
```

■ Compatibility issue

The compatibility of our two formulas is not immediately obvious.

```
Assoc[ADJOIN[singleton[0], IMAGE[SUCC], id[omega]]
composite[ADJOIN[singleton[0], id[omega], SUCC] == composite[id[omega], SUCC]

composite[ADJOIN[singleton[0], id[omega], SUCC] := composite[id[omega], SUCC]
```

This holds because **0** belongs to the successor of every natural number:

```
implies[member[x, omega], member[0, succ[x]]]
True
```