bases for topologies

Johan G. F. Belinfante
2003 April 10

```
<< goedel52.r48; << tools.m

:Package Title: goedel52.r47          2003 April 4 at 8:10 p.m

It is now:  2003 Apr 10 at 8:6

Loading Simplification Rules

TOOLS.M                       Revised 2003 April 1

weightlimit = 40
```

■ summary

Topologies are often defined by taking **Uclosure** of a basis. In this notebook it is shown that the class of topologies can be obtained by **UCLOSURE** from the class **CAPclosed**. The key step is to show that **CAPclosed** is invariant under **UCLOSURE**.

```
forall[z, implies[member[z, CAPclosed], member[Uclosure[z], CAPclosed]]] // assert

subclass[image[UCLOSURE, CAPclosed], CAPclosed]
```

This will be our first task.

■ a consequence of monotonicity

The lemma derived here just uses monotonicity.

```
Map[implies[and[subclass[x, z], subclass[y, z]], #] &,
 SubstTest[implies, subclass[u, v], subclass[image[CAP, u], image[CAP, v]],
    {u -> cart[x, y], v -> cart[z, z]}]]

or[not[subclass[x, z]], not[subclass[y, z]],
  subclass[image[CAP, cart[x, y]], image[CAP, cart[z, z]]]] == True

or[not[subclass[x_, z_]], not[subclass[y_, z_]],
  subclass[image[CAP, cart[x_, y_]], image[CAP, cart[z_, z_]]]] := True

Map[not, SubstTest[and, implies[p1, p2], implies[p3, p4], implies[and[p2, p4], p5],
  not[implies[and[p1, p3], p5]], {p1 -> and[subclass[x, z], subclass[y, z]],
        p2 -> subclass[image[CAP, cart[x, y]], image[CAP, cart[z, z]]],
        p3 -> equal[z, image[CAP, cart[z, z]]],
   p4 -> subclass[image[CAP, cart[z, z]], z],
        p5 -> subclass[image[CAP, cart[x, y]], z]}]]

or[not[equal[z, image[CAP, cart[z, z]]]], not[subclass[x, z]],
  not[subclass[y, z]], subclass[image[CAP, cart[x, y]], z]] == True
```

```
or[not[equal[z_, image[CAP, cart[z_, z_]]]], not[subclass[x_, z_]],
  not[subclass[y_, z_]], subclass[image[CAP, cart[x_, y_]], z_]] := True
```

■ intersections of unions

The key formula here is a distributive law:

```
U[image[CAP, cart[x, y]]]

intersection[U[x], U[y]]

SubstTest[implies, and[subclass[w, z], member[w, V]],
 member[U[w], Uclosure[z]], w -> image[CAP, cart[x, y]]]

or[member[intersection[U[x], U[y]], Uclosure[z]],
  not[member[image[CAP, cart[x, y]], V]],
  not[subclass[image[CAP, cart[x, y]], z]]] == True

or[member[intersection[U[x_], U[y_]], Uclosure[z_]],
  not[member[image[CAP, cart[x_, y_]], V]],
  not[subclass[image[CAP, cart[x_, y_]], z_]]] := True
```

Lemma:

```
SubstTest[implies, and[FUNCTION[z], member[w, V]], member[image[z, w], V], z -> CAP]

or[member[image[CAP, w], V], not[member[w, V]]] == True

or[member[image[CAP, w_], V], not[member[w_, V]]] := True
```

Lemma:

```
Map[implies[and[member[x, V], member[y, V]], #] &,
 SubstTest[implies, member[w, V], member[image[CAP, w], V], w -> cart[x, y]]]

or[member[image[CAP, cart[x, y]], V], not[member[x, V]], not[member[y, V]]] == True

or[member[image[CAP, cart[x_, y_]], V], not[member[x_, V]], not[member[y_, V]]] := True

Map[not,
 SubstTest[and, implies[p1, p2], implies[and[p3, p4], p5], implies[and[p2, p5], p6],
        not[implies[and[p1, p3, p4], p6]],
        {p1 -> and[member[x, V], member[y, V]],
   p2 -> member[image[CAP, cart[x, y]], V],
            p3 -> equal[z, image[CAP, cart[z, z]]],
   p4 -> and[subclass[x, z], subclass[y, z]],
            p5 -> subclass[image[CAP, cart[x, y]], z],
            p6 -> member[intersection[U[x], U[y]], Uclosure[z]]}]]

or[member[intersection[U[x], U[y]], Uclosure[z]], not[equal[z, image[CAP, cart[z, z]]]],
  not[member[x, V]], not[member[y, V]], not[subclass[x, z]], not[subclass[y, z]]] == True
```

This simplifies when **z** is a set:

```
Map[implies[member[z, V], #] &, %] // MapNotNot

or[member[intersection[U[x], U[y]], Uclosure[z]], not[equal[z, image[CAP, cart[z, z]]]],
  not[member[z, V]], not[subclass[x, z]], not[subclass[y, z]]] == True
```

```
or[member[intersection[U[x_], U[y_]], Uclosure[z_]],
  not[equal[z_, image[CAP, cart[z_, z_]]]], not[member[z_, V]],
  not[subclass[x_, z_]], not[subclass[y_, z_]]] := True
```

This proves:

```
implies[and[member[z, CAPclosed], member[x, P[z]], member[y, P[z]]],
 member[intersection[U[x], U[y]], Uclosure[z]]]
```

```
True
```

■ eliminating the variables x and y via universal quantifiers

Lemma:

```
member[pair[u, v], composite[inverse[BIGCUP], w]] // AssertTest
```

```
member[pair[u, v], composite[inverse[BIGCUP], w]] ==
 and[member[u, V], member[v, V], member[pair[u, U[v]], w]]
```

```
member[pair[u_, v_], composite[inverse[BIGCUP], w_]] :=
 and[member[u, V], member[v, V], member[pair[u, U[v]], w]]
```

Universal quantification is defined in terms of existential quantification via negation. For this reason we need the following negative statement:

```
and[equal[z, image[CAP, cart[z, z]]], member[z, V],
  not[member[intersection[U[x], U[y]], Uclosure[z]]],
  subclass[x, z], subclass[y, z]] // NotNotTest
```

```
and[equal[z, image[CAP, cart[z, z]]],
  member[z, V], not[member[intersection[U[x], U[y]], Uclosure[z]]],
  subclass[x, z], subclass[y, z]] == False
```

```
and[equal[z_, image[CAP, cart[z_, z_]]], member[z_, V],
  not[member[intersection[U[x_], U[y_]], Uclosure[z_]]],
  subclass[x_, z_], subclass[y_, z_]] := False
```

The variables **x** and **y** can now be removed by universal quantifiers:

```
SubstTest[assert, forall[x, y, implies[member[z, u], member[pair[x, y], v]]],
            {u -> CAPclosed,
   v -> union[complement[cart[P[z], P[z]]], image[cross[inverse[BIGCUP],
       inverse[BIGCUP]], image[inverse[CAP], Uclosure[z]]]]}] // Reverse
```

```
or[not[equal[z, image[CAP, cart[z, z]]]], not[member[z, V]], subclass[cart[P[z], P[z]],
   composite[inverse[BIGCUP], image[inverse[CAP], Uclosure[z]], BIGCUP]]] == True
```

```
or[not[equal[z_, image[CAP, cart[z_, z_]]]],
  not[member[z_, V]], subclass[cart[P[z_], P[z_]],
   composite[inverse[BIGCUP], image[inverse[CAP], Uclosure[z_]], BIGCUP]]] := True
```

Lemma.

```
SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
    {u -> cart[P[z], P[z]],
  v -> composite[inverse[BIGCUP], image[inverse[CAP], Uclosure[z]], BIGCUP],
      w -> cross[BIGCUP, BIGCUP]}]

or[not[subclass[cart[P[z], P[z]],
    composite[inverse[BIGCUP], image[inverse[CAP], Uclosure[z]], BIGCUP]]],
  subclass[image[CAP, cart[Uclosure[z], Uclosure[z]]], Uclosure[z]]] == True

or[not[subclass[cart[P[z_], P[z_]],
    composite[inverse[BIGCUP], image[inverse[CAP], Uclosure[z_]], BIGCUP]]],
  subclass[image[CAP, cart[Uclosure[z_], Uclosure[z_]]], Uclosure[z_]]] := True
```

Replacing one inclusion by an equality:

```
Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
    {p1 -> member[z, CAPclosed], p2 -> subclass[cart[P[z], P[z]],
     composite[inverse[BIGCUP], image[inverse[CAP], Uclosure[z]], BIGCUP]],
        p3 -> subclass[image[CAP, cart[Uclosure[z], Uclosure[z]]], Uclosure[z]]}]]

or[not[equal[z, image[CAP, cart[z, z]]]], not[member[z, V]],
  subclass[image[CAP, cart[Uclosure[z], Uclosure[z]]], Uclosure[z]]] == True

or[not[equal[z_, image[CAP, cart[z_, z_]]]], not[member[z_, V]],
  subclass[image[CAP, cart[Uclosure[z_], Uclosure[z_]]], Uclosure[z_]]] := True
```

Replacing the other inclusion by an equality:

```
SubstTest[and, implies[p, subclass[u, v]], implies[p, subclass[v, u]],
    {p -> member[z, CAPclosed],
   u -> image[CAP, cart[Uclosure[z], Uclosure[z]]], v -> Uclosure[z]}] // Reverse

or[equal[image[CAP, cart[Uclosure[z], Uclosure[z]]], Uclosure[z]],
  not[equal[z, image[CAP, cart[z, z]]]], not[member[z, V]]] == True

or[equal[image[CAP, cart[Uclosure[z_], Uclosure[z_]]], Uclosure[z_]],
  not[equal[z_, image[CAP, cart[z_, z_]]]], not[member[z_, V]]] := True
```

Eliminating the variable **z**.

```
Map[equal[V, #] &, SubstTest[class, z,
   implies[member[z, w], member[Uclosure[z], w]], w -> CAPclosed]] // Reverse

subclass[image[UCLOSURE, CAPclosed], CAPclosed] == True
```

That is, the class **CAPclosed** is invariant under **UCLOSURE**.

```
subclass[image[UCLOSURE, CAPclosed], CAPclosed] := True
```

■ reformulation as an equation

The goal in this section is to recast the inclusion derived in the preceding section as an equation.

```
SubstTest[subclass, u, intersection[v, w],
    {u -> image[UCLOSURE, CAPclosed], v -> CAPclosed, w -> fix[UCLOSURE]}]

subclass[image[UCLOSURE, CAPclosed], TOPS] == True

subclass[image[UCLOSURE, CAPclosed], TOPS] := True
```

```
ImageComp[UCLOSURE, id[fix[UCLOSURE]], TOPS] // Reverse
```

```
image[UCLOSURE, TOPS] == TOPS
```

```
image[UCLOSURE, TOPS] := TOPS
```

Reverse inclusion:

```
SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
    {u -> TOPS, v -> CAPclosed, w -> UCLOSURE}]
```

```
subclass[TOPS, image[UCLOSURE, CAPclosed]] == True
```

```
subclass[TOPS, image[UCLOSURE, CAPclosed]] := True
```

An equational version:

```
SubstTest[and, subclass[u, v], subclass[v, u],
    {u -> TOPS, v -> image[UCLOSURE, CAPclosed]}]
```

```
True == equal[TOPS, image[UCLOSURE, CAPclosed]]
```

This equation can be made into a rewrite rule.

```
image[UCLOSURE, CAPclosed] := TOPS
```