minimal open base for an Alexandrov topology

Johan G. F. Belinfante
2013 December 31

```
In[1]:= SetDirectory["l:"]; << goedel.13dec30a
```

```
:Package Title: goedel.13dec30a                2013 December 30 at 6:37 p.m.

Loading takes about seventeen minutes, half that time due to builtin pauses.

It is now:  2013 Dec 31 at 14:6

Loading Simplification Rules

TOOLS.M is now incorporated in the GOEDEL program as of 2010 September 3

weightlimit = 40

Loading completed.

It is now:  2013 Dec 31 at 14:24
```

summary

A collection of sets **s** is a **base** for a topology **t** if **t = Uclosure[s]**. An Alexandrov topology is one that is closed under arbitrary intersections. If **t ∈ ALEX** is an Alexandrov topology, the sets of the form **hull[t, {x}]** for **x ∈ U[t]** form an open base for **t**.

derivation

In this section some rewrite rules for Alexandrov topologies are derived.

Theorem.

```
In[2]:= SubstTest[implies, and[member[x, y], subclass[y, z]],
          member[x, z], {y → ALEX, z → fix[ACLOSURE]}] // Reverse
```

```
Out[2]= or[equal[x, Aclosure[x]], not[member[x, ALEX]]] == True
```

```
In[3]:= or[equal[x_, Aclosure[x_]], not[member[x_, ALEX]]] := True
```

Corollary.

```
In[4]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p2, p3], p4],
          not[implies[p1, p4]], {p1 → member[x, ALEX], p2 → equal[x, Aclosure[x]],
           p3 → equal[Aclosure[x], fix[HULL[x]]], p4 → equal[x, fix[HULL[x]]]}]] // Reverse

Out[4]= or[equal[x, fix[HULL[x]]], not[member[x, ALEX]]] == True

In[5]:= or[equal[x_, fix[HULL[x_]]], not[member[x_, ALEX]]] := True
```

Corollary.

```
In[6]:= Map[not, SubstTest[and, implies[p1, p2],
          implies[p2, p3], not[implies[p1, p3]], {p1 -> member[x, ALEX],
           p2 → equal[x, fix[HULL[x]]], p3 → subclass[image[HULL[x], y], x]}]] // Reverse

Out[6]= or[not[member[x, ALEX]], subclass[image[HULL[x], y], x]] == True

In[7]:= or[not[member[x_, ALEX]], subclass[image[HULL[x_], y_], x_]] := True
```

Uclosure result

In this section a general **Uclosure** inclusion is derived.

Theorem.

```
In[9]:= ImageComp[composite[HULL[x], SINGLETON], id[U[x]], V] // Reverse

Out[9]= image[HULL[x], image[SINGLETON, U[x]]] == image[HULL[x], range[SINGLETON]]

In[10]:= image[HULL[x_], image[SINGLETON, U[x_]]] := image[HULL[x], range[SINGLETON]]
```

Lemma.

```
In[11]:= Map[implies[member[x, U[t]], #] &, SubstTest[implies,
           and[member[v, s], member[x, v], subclass[v, u]], member[x, core[s, u]],
           {v → hull[t, set[x]], s -> image[HULL[t], range[SINGLETON]]}]] // Reverse

Out[11]= or[member[x, core[image[HULL[t], range[SINGLETON]], u]], not[member[x, U[t]]],
           not[member[hull[t, set[x]], image[HULL[t], range[SINGLETON]]]],
           not[subclass[hull[t, set[x]], u]]] == True

In[12]:= (% /. {t → t_, u → u_, x → x_}) /. Equal → SetDelayed
```

Lemma.

```
In[13]:= Map[implies[member[x, U[t]], #] &, SubstTest[member, APPLY[funpart[z], x],
           range[funpart[z]], z → composite[HULL[t], SINGLETON, id[U[t]]]]] // Reverse

Out[13]= or[member[hull[t, set[x]], image[HULL[t], range[SINGLETON]]],
           not[member[x, U[t]]]] == True

In[14]:= (% /. {t → t_, x → x_}) /. Equal → SetDelayed
```

Lemma.

```
In[15]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p1, p2, p3], p4],
           not[implies[and[p1, p3], p4]], {p1 → member[x, U[t]],
            p2 -> member[hull[t, set[x]], image[HULL[t], range[SINGLETON]]],
            p3 -> subclass[hull[t, set[x]], u],
            p4 -> member[x, core[image[HULL[t], range[SINGLETON]], u]]}]] // Reverse

Out[15]= or[member[x, core[image[HULL[t], range[SINGLETON]], u]],
           not[member[x, U[t]]], not[subclass[hull[t, set[x]], u]]] == True

In[16]:= (% /. {t → t_, u → u_, x → x_}) /. Equal → SetDelayed
```

Theorem.

```
In[17]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p2, p3], p4],
           not[implies[p1, p4]], {p1 → and[member[x, u], member[u, t]],
            p2 -> subclass[hull[t, set[x]], u], p3 → member[x, U[t]],
            p4 -> member[x, core[image[HULL[t], range[SINGLETON]], u]]}]] // Reverse

Out[17]= or[member[x, core[image[HULL[t], range[SINGLETON]], u]],
           not[member[u, t]], not[member[x, u]]] == True

In[18]:= (% /. {t → t_, u → u_, x → x_}) /. Equal → SetDelayed
```

Lemma.

```
In[19]:= Map[equal[V, domain[#]] &, SubstTest[reify, x,
           case[or[member[x, core[s, u]], not[member[u, t]], not[member[x, u]]]],
           s -> image[HULL[t], range[SINGLETON]]]]

Out[19]= or[not[member[u, t]], subclass[u, core[image[HULL[t], range[SINGLETON]], u]]] == True

In[20]:= (% /. {t → t_, u → u_}) /. Equal → SetDelayed
```

Theorem.

```
In[25]:= Map[equal[V, domain[#]] &,
          SubstTest[reify, u, case[or[not[member[u, x]], subclass[u, core[s, u]]]],
           s -> image[HULL[x], range[SINGLETON]]]]

Out[25]= subclass[x, Uclosure[image[HULL[x], range[SINGLETON]]]] == True

In[26]:= subclass[x_, Uclosure[image[HULL[x_], range[SINGLETON]]]] := True
```

main theorem

Lemma.

```
In[27]:= SubstTest[implies, and[subclass[s, t], member[t, TOPS], subclass[t, Uclosure[s]]],
           equal[t, Uclosure[s]], s -> Uclosure[image[HULL[t], range[SINGLETON]]]] // Reverse

Out[27]= or[equal[t, Uclosure[image[HULL[t], range[SINGLETON]]]], not[member[t, TOPS]],
           not[subclass[Uclosure[image[HULL[t], range[SINGLETON]]], t]]] == True

In[28]:= (% /. t → t_) /. Equal → SetDelayed
```

Main theorem.

```
In[29]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
            implies[p1, p4], implies[and[p2, p5], p6], not[implies[p1, p6]],
            {p1 → member[x, ALEX], p2 → member[x, TOPS], p3 → equal[x, Uclosure[x]],
             p4 -> subclass[image[HULL[x], range[SINGLETON]], x],
             p5 -> subclass[Uclosure[image[HULL[x], range[SINGLETON]]], x],
             p6 -> equal[x, Uclosure[image[HULL[x], range[SINGLETON]]]]}]] // Reverse

Out[29]= or[equal[x, Uclosure[image[HULL[x], range[SINGLETON]]]], not[member[x, ALEX]]] == True

In[30]:= or[equal[x_, Uclosure[image[HULL[x_], range[SINGLETON]]]],
           not[member[x_, ALEX]]] := True
```

Corollary. A variable-free restatement.

```
In[31]:= Map[equal[V, domain[#]] &, SubstTest[reify, t,
           case[or[equal[t, Uclosure[image[HULL[t], r]]], not[member[t, ALEX]]]],
           r → range[SINGLETON]]]

Out[31]= subclass[ALEX, fix[composite[UCLOSURE, IMAGE[SECOND],
             IMAGE[id[cart[range[SINGLETON], V]]], LAMBHULL]]] == True

In[32]:= subclass[ALEX, fix[composite[UCLOSURE, IMAGE[SECOND],
             IMAGE[id[cart[range[SINGLETON], V]]], LAMBHULL]]] := True
```