spine[x,y]

Johan G. F. Belinfante
2007 June 10

In[1]:=
```
SetDirectory["l:"]; << goedel94.09a; << tools.m
```

```
:Package Title: goedel94.09a      2007 June 9 at 8:50 p.m.

It is now:  2007 Jun 10 at 11:27

Loading Simplification Rules

TOOLS.M                                    Revised 2007 June 6

weightlimit = 40
```

summary

In this notebook, the constructor **spine[x,y]** is defined, and some of its basic properties are derived. The author had introduced the concept of **spine** in July 1972 in a handout for an algebra class at Carnegie-Mellon University that contained a proof of Zorn's lemma from the axiom of choice. That this constructor can serve as a wrapper for chains follows from the observation that, more generally, the constructors **least[x, y]** and **greatest[x, y]** can function as wrappers for cliques.

definition

The constructor **spine[x,y]** is defined by the following membership rule.

In[2]:=
```
member[w_, spine[x_, y_]] :=
  and[member[w, y], subclass[y, union[image[x, set[w]], image[inverse[x], set[w]]]]]
```

normalization

The constructor **spine[x,y]** is normalized by the following rewrite rule:

In[3]:=
```
spine[x, y] // Normality // Reverse
```

Out[3]=
```
intersection[y, complement[fix[composite[complement[x], id[y], complement[x]]]]] ==
  spine[x, y]
```

In[4]:=
```
intersection[y_, complement[fix[composite[complement[x_], id[y_], complement[x_]]]]] :=
  spine[x, y]
```

The class **spine[x,y]** is the class of all members of **y** that are comparable with every member of **y** with respect to the relation **x**.

```
In[5]:= class[u, and[member[u, y],
          forall[v, implies[member[v, y], or[member[pair[u, v], x], member[pair[v, u], x]]]]]]

Out[5]= spine[x, y]
```

spine as least or greatest

Much of the theory of **spine[x, y]** can be derived from that of **greatest[x, y]**. These concepts are related as follows:

```
In[6]:= greatest[union[x, inverse[x]], y] // Normality

Out[6]= intersection[y, ub[union[x, inverse[x]], y]] == spine[x, y]

In[7]:= intersection[y_, ub[union[x_, inverse[x_]], y_]] := spine[x, y]
```

One could also use **least[x, y]** instead:

```
In[8]:= least[union[x, inverse[x]], y] // Normality

Out[8]= intersection[y, lb[union[x, inverse[x]], y]] == spine[x, y]

In[9]:= intersection[y_, lb[union[x_, inverse[x_]], y_]] := spine[x, y]
```

basic properties

The spine of a class is a subclass.

```
In[10]:= SubstTest[subclass, least[t, y], y, t → union[x, inverse[x]]] // Reverse

Out[10]= subclass[spine[x, y], y] == True

In[11]:= subclass[spine[x_, y_], y_] := True
```

Corollary.

```
In[12]:= equal[intersection[spine[x, y], y], spine[x, y]]

Out[12]= True

In[13]:= intersection[y_, spine[x_, y_]] := spine[x, y]
```

It does not matter if one replaces **x** with **composite[Id,x]**.

```
In[14]:= spine[composite[Id, x], y] // Normality

Out[14]= spine[composite[Id, x], y] == spine[x, y]
```

```
In[15]:= spine[composite[Id, x_], y_] := spine[x, y]
```

It also does not matter if one replaces **x** with its inverse.

```
In[16]:= spine[inverse[x], y] // Normality

Out[16]= spine[inverse[x], y] == spine[x, y]

In[17]:= spine[inverse[x_], y_] := spine[x, y]
```

idempotence

Theorem. The spine of a spine is itself. Comment. A similar property holds for the constructors **least** and **greatest**.

```
In[18]:= SubstTest[least, t, least[t, y], t → union[x, inverse[x]]] // Reverse

Out[18]= spine[x, spine[x, y]] == spine[x, y]

In[19]:= spine[x_, spine[x_, y_]] := spine[x, y]
```

wrapper properties

One can regard the constructor **spine[x, y]** as a wrapper for a generic **x**-chain. In this section the basic wrapper properties are derived. More generally, both **least[x, y]** and **greatest[x, y]** can be regarded as wrappers for generic **x**-cliques, and this fact is used to derive the corresponding rewrite rules for **spine**. For **greatest** and **least**, the following wrapper-removal rules are available:

```
In[20]:= equal[y, greatest[x, y]]

Out[20]= subclass[cart[y, y], x]

In[21]:= equal[y, least[x, y]]

Out[21]= subclass[cart[y, y], x]
```

Theorem. Wrapper removal rule for **spine**.

```
In[22]:= SubstTest[equal, y, least[t, y], t → union[x, inverse[x]]] // Reverse

Out[22]= equal[y, spine[x, y]] == subclass[cart[y, y], union[x, inverse[x]]]

In[23]:= equal[y_, spine[x_, y_]] := subclass[cart[y, y], union[x, inverse[x]]]
```

Theorem. Spines are chains. This is derived here as a corollary of the idempotence property for **spine**.

```
In[24]:= SubstTest[equal, t, spine[x, t], t → spine[x, y]]

Out[24]= subclass[cart[spine[x, y], spine[x, y]], union[x, inverse[x]]] == True
```

```
In[25]:= subclass[cart[spine[x_, y_], spine[x_, y_]], union[x_, inverse[x_]]] := True
```

Conditional rule.

```
In[27]:= implies[subclass[cart[y, y], union[x, inverse[x]]], equal[spine[x, y], y]]

Out[27]= True

In[28]:= spine[x_, y_] := y /; subclass[cart[y, y], union[x, inverse[x]]]
```

sethood

Since **spine[x, y]** is a subclass of **y**, it is a set when **y** is a set.

```
In[29]:= SubstTest[implies, and[subclass[t, y], member[y, z]],
           member[t, V], t → spine[x, y]] // Reverse

Out[29]= or[member[spine[x, y], V], not[member[y, z]]] == True

In[30]:= or[member[spine[x_, y_], V], not[member[y_, z_]]] := True
```

special cases

No new rewrite rule is needed for this special case:

```
In[32]:= spine[x, 0]

Out[32]= 0
```

When the first argument is **0**, one has:

```
In[33]:= spine[0, x] // Normality

Out[33]= spine[0, x] == intersection[x, complement[image[V, x]]]

In[34]:= spine[0, x_] := intersection[x, complement[image[V, x]]]
```

The spine of a singleton is either itself or is empty.

```
In[35]:= spine[x, set[y]] // Normality

Out[35]= spine[x, set[y]] == intersection[fix[x], set[y]]

In[36]:= spine[x_, set[y_]] := intersection[fix[x], set[y]]
```

No new rewrite rule is needed for this special case:

```
In[39]:= spine[V, x]

Out[39]= x
```

reify rule

The **reify** rule for spine can be deduced from that for **greatest**. (Comment. A slightly different, but equivalent, formula is obtained if one uses **least** instead of **greatest**.)

```
In[40]:= SubstTest[reify, x, greatest[union[f[x], h[f[x]]], g[x]], h → inverse] // Reverse

Out[40]= reify[x, spine[f[x], g[x]]] ==
          composite[Id, intersection[complement[composite[SECOND, intersection[
               composite[inverse[FIRST], reify[x, g[x]]], composite[inverse[E],
                id[SYM], inverse[LB[complement[reify[x, f[x]]]]]]]]]], reify[x, g[x]]]]

In[44]:= reify[x_, spine[y_, z_]] :=
          composite[Id, intersection[complement[composite[SECOND, intersection[
               composite[inverse[FIRST], reify[x, z]], composite[inverse[E],
                id[SYM], inverse[LB[complement[reify[x, y]]]]]]]]], reify[x, z]]]
```

Example:

```
In[45]:= reify[y, spine[x, y]]

Out[45]= GREATEST[union[x, inverse[x]]]
```

an example for the reify rule

An example of the use of **reify** is provided in this section. The conditional rewrite rule applies to the case of any ordinal **ord[x]**, which is a chain of sets with respect to the inclusion relation **S**.

```
In[46]:= SubstTest[reify, x, spine[S, f[x]], f → ord]

Out[46]= composite[GREATEST[union[S, inverse[S]]], id[OMEGA]] == composite[inverse[E], id[OMEGA]]

In[47]:= composite[GREATEST[union[S, inverse[S]]], id[OMEGA]] :=
          composite[inverse[E], id[OMEGA]]
```

Corollary.

```
In[48]:= Map[composite[VERTSECT[#], id[ord[x]]] &,
          Assoc[GREATEST[union[S, inverse[S]]], id[OMEGA], id[ord[x]]]]

Out[48]= composite[VERTSECT[GREATEST[union[S, inverse[S]]]], id[ord[x]]] == id[ord[x]]

In[49]:= composite[VERTSECT[GREATEST[union[S, inverse[S]]]], id[ord[x_]]] := id[ord[x]]
```

serendipity

The following fact was discovered in the course of this study.

```
In[50]:= equal[intersection[GREATEST[x], composite[complement[x], GREATEST[x]]], 0]

Out[50]= True

In[51]:= intersection[composite[complement[x_], GREATEST[x_]], GREATEST[x_]] := 0
```