

definitions for GLB[x] and LUB[x]

Johan G. F. Belinfante
2005 September 27

```
In[1]:= SetDirectory["1:"]; << goedel73.18b; << tools.m

:Package Title: goedel73.18b          2005 September 18 at 7:40 p.m.

It is now: 2005 Sep 27 at 11:37

Loading Simplification Rules

TOOLS.M          Revised 2005 September 18

weightlimit = 40
```

summary

The concepts of greatest lower bound and least upper bound are basic, and have many important applications. For example, for the case of the subset relation S , the functions **BIGCAP** and **BIGCUP** are related to greatest lower bounds and least upper bounds, respectively:

```
In[2]:= class[pair[u, v], member[v, greatest[z, lb[z, u]]]] /. z -> S
Out[2]= BIGCAP
```

```
In[3]:= class[pair[u, v], member[v, least[z, ub[z, u]]]] /. z -> S
Out[3]= BIGCUP
```

For many applications these literal concepts of greatest lower bounds and least upper bounds produce complicated formulas. For example, if one simply replaces the subset relation S in these formulas with a cartesian product, one finds:

```
In[4]:= class[pair[u, v], member[v, greatest[z, lb[z, u]]]] /. z -> cart[x, y]
Out[4]= union[cart[intersection[complement[set[0]], P[y]], intersection[x, y]],
  cart[P[y], intersection[x, y, complement[image[V, complement[x]]]]],
  cart[set[0], intersection[y, complement[image[V, complement[x]]]]]]
```

A much simpler result is obtained by making a minor modification, replacing **lb[z,u]** by its intersection with **domain[z]**.

```
In[5]:= class[pair[u, v], member[v, greatest[z, intersection[domain[z], lb[z, u]]]] /.
  z -> cart[x, y]
Out[5]= cart[P[y], intersection[x, y]]
```

Intersecting with **domain[z]** only affects how the empty set is treated:

```
In[6]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]],
  subclass[u, w], {u → lb[x, y], v → image[inverse[x], y], w → domain[x]}]
Out[6]= or[equal[0, y], subclass[lb[x, y], domain[x]]] == True
In[7]:= or[equal[0, y_], subclass[lb[x_, y_], domain[x_]]] := True
```

In principle, one could redefine the lower bound constructor **lb[x,y]** by including an intersection with **domain[x]**, but doing so does not appear to be particularly useful. In this notebook, a fresh start is made. The old definitions of **GLB** and **LUB** have been removed from the **GOEDEL** program, along with all rewrite rules involving **GLB** and **LUB**. New membership rules wrapped with **class** have been added to the **GOEDEL** program, providing a starting point from which various basic properties of **GLB[x]** and **LUB[x]** will be derived.

wrapped and unwrapped membership rules

The following membership rules for **GLB[x]** and **LUB[x]**, wrapped with **class**, serve as new definitions of these constructors.

```
In[8]:= Begin["Goedel`Private`"];
In[9]:= FirstMatch[class[w_, member[x_, HoldPattern[GLB[y_]]]]]
Out[9]= class[w_, member[x_, GLB[y_]]] := Module[{u = Unique[], v =
  Unique[]}, class[w, exists[u, v, and[equal[x, pair[u, v]], member[
  u, V], member[v, domain[y]], subclass[u, image[y, set[v]]], subclass[
  intersection[domain[y], lb[y, u]], image[inverse[y], set[v]]]]]]]
In[10]:= FirstMatch[class[w_, member[x_, HoldPattern[LUB[y_]]]]]
Out[10]= class[w_, member[x_, LUB[y_]]] := Module[{u = Unique[], v =
  Unique[]}, class[w, exists[u, v, and[equal[x, pair[u, v]], member[
  u, V], member[v, range[y]], subclass[u, image[inverse[y], set[v]]],
  subclass[intersection[range[y], ub[y, u]], image[y, set[v]]]]]]]
```

From the wrapped membership rule one can derive the following unwrapped membership rule for **GLB[z]** for the special case of ordered pairs:

```
In[11]:= member[pair[x, y], GLB[z]] // AssertTest
Out[11]= member[pair[x, y], GLB[z]] ==
  and[member[x, V], member[y, domain[z]], subclass[x, image[z, set[y]]],
  subclass[intersection[domain[z], lb[z, x]], image[inverse[z], set[y]]]]
In[12]:= member[pair[x_, y_], GLB[z_]] :=
  and[member[x, V], member[y, domain[z]], subclass[x, image[z, set[y]]],
  subclass[intersection[domain[z], lb[z, x]], image[inverse[z], set[y]]]]
```

A similar result holds for **LUB[z]**.

```

In[13]:= member[pair[x, y], LUB[z]] // AssertTest

Out[13]= member[pair[x, y], LUB[z]] ==
  and[member[x, V], member[y, range[z]], subclass[x, image[inverse[z], set[y]]],
  subclass[intersection[range[z], ub[z, x]], image[z, set[y]]]]

In[14]:= member[pair[x_, y_], LUB[z_]] :=
  and[member[x, V], member[y, range[z]], subclass[x, image[inverse[z], set[y]]],
  subclass[intersection[range[z], ub[z, x]], image[z, set[y]]]]

```

These definitions can be restated as follows:

```

In[15]:= member[pair[x, y], GLB[z]] ==
  and[member[x, V], member[y, greatest[z, intersection[domain[z], lb[z, x]]]]]

Out[15]= True

In[16]:= member[pair[x, y], LUB[z]] ==
  and[member[x, V], member[y, least[z, intersection[range[z], ub[z, x]]]]]

Out[16]= True

```

Our main concerns in this notebook are with normalization rules for these constructors.

normalization for GLB[x]

The basic **Normality** result for the **GLB[x]** constructor is this:

```

In[17]:= GLB[x] // Normality // Reverse

Out[17]= composite[id[domain[x]], intersection[
  complement[composite[complement[x], id[domain[x]], LB[x]]], LB[x]]] = GLB[x]

In[18]:= composite[id[domain[x_]], intersection[
  complement[composite[complement[x_], id[domain[x_]], LB[x_]]], LB[x_]]] := GLB[x]

```

A second application of **Normality** yields a basic rewrite rule:

```

In[19]:= GLB[x] // Normality // Reverse

Out[19]= composite[Id, GLB[x]] = GLB[x]

In[20]:= composite[Id, GLB[x_]] := GLB[x]

```

The **GLB** constructor ignores any elements that are not ordered pairs.

```

In[21]:= GLB[composite[Id, x]] // ReInNormality

Out[21]= GLB[composite[Id, x]] = GLB[x]

In[22]:= GLB[composite[Id, x_]] := GLB[x]

```

normalization for LUB[x]

Similar results hold for the **LUB[x]** constructor. The basic **Normality** result is this:

```
In[23]:= LUB[x] // Normality // Reverse
```

```
Out[23]= composite[id[range[x]], intersection[
  complement[composite[complement[inverse[x]], id[range[x]], UB[x]]], UB[x]]] == LUB[x]
```

```
In[24]:= composite[id[range[x_]], intersection[complement[
  composite[complement[inverse[x_]], id[range[x_]], UB[x_]]], UB[x_]]] := LUB[x]
```

A second application yields:

```
In[25]:= LUB[x] // Normality // Reverse
```

```
Out[25]= composite[Id, LUB[x]] == LUB[x]
```

```
In[26]:= composite[Id, LUB[x_]] := LUB[x]
```

The **LUB** constructor also ignores any elements that are not ordered pairs.

```
In[27]:= LUB[composite[Id, x]] // ReInNormality
```

```
Out[27]= LUB[composite[Id, x]] == LUB[x]
```

```
In[28]:= LUB[composite[Id, x_]] := LUB[x]
```

interrelations between GLB and LUB

Each of the constructors **GLB** and **LUB** is related to the other as follows:

```
In[29]:= GLB[inverse[x]] // ReInNormality
```

```
Out[29]= GLB[inverse[x]] == LUB[x]
```

```
In[30]:= GLB[inverse[x_]] := LUB[x]
```

```
In[31]:= LUB[inverse[x]] // ReInNormality
```

```
Out[31]= LUB[inverse[x]] == GLB[x]
```

```
In[32]:= LUB[inverse[x_]] := GLB[x]
```

vertical sections

Vertical section rules provide an important alternative to membership rules, and are needed in order to be able to take full advantage of the **VSNormality** test, and its relatives. The vertical section rule for **GLB** can be derived using the more primitive **Normality** test:

```
In[33]:= image[GLB[x], set[y]] // Normality
Out[33]= image[GLB[x], set[y]] == intersection[domain[x],
        image[V, set[y]], lb[x, y], ub[x, intersection[domain[x], lb[x, y]]]]

In[34]:= image[GLB[x_], set[y_]] := intersection[domain[x],
        image[V, set[y]], lb[x, y], ub[x, intersection[domain[x], lb[x, y]]]]
```

Replacing **x** with its inverse yields an analogous result for **LUB**.

```
In[35]:= SubstTest[image, GLB[z], set[y], z → inverse[x]]
Out[35]= image[LUB[x], set[y]] == intersection[image[V, set[y]],
        lb[x, intersection[range[x], ub[x, y]]], range[x], ub[x, y]]

In[36]:= image[LUB[x_], set[y_]] := intersection[image[V, set[y]],
        lb[x, intersection[range[x], ub[x, y]]], range[x], ub[x, y]]
```

examples

The results for the subset relation **S** are not affected by the revision of the definitions of **GLB** and **LUB**.

```
In[37]:= GLB[S] // RelnNormality
Out[37]= GLB[S] == BIGCAP

In[38]:= GLB[S] := BIGCAP

In[39]:= LUB[S] // RelnNormality
Out[39]= LUB[S] == BIGCUP

In[40]:= LUB[S] := BIGCUP
```

For the case of cartesian products, one obtains the simple formulas mentioned in the introduction of this notebook.

```
In[41]:= GLB[cart[x, y]] // RelnNormality
Out[41]= GLB[cart[x, y]] == cart[P[y], intersection[x, y]]

In[42]:= GLB[cart[x_, y_]] := cart[P[y], intersection[x, y]]
```

Similarly:

```
In[43]:= LUB[cart[x, y]] // ReInNormality
```

```
Out[43]= LUB[cart[x, y]] = cart[P[x], intersection[x, y]]
```

```
In[44]:= LUB[cart[x_, y_]] := cart[P[x], intersection[x, y]]
```