image[BIGCAP, x ⋂ P[y]]

Johan G. F. Belinfante
2012 July 28

```
In[1]:= SetDirectory["l:"]; << goedel.12jul24a
```

```
:Package Title: goedel.12jul24a                 2012 July 24 at 3:50 p.m.

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

It is now:  2012 Jul 28 at 10:0

Loading Simplification Rules

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

weightlimit = 40

Loading completed.

It is now:  2012 Jul 28 at 10:15
```

summary

The **Aclosure** of a class **x** is the class **image[BIGCAP, P[x]]**. In this notebook some rewrite rules are derived for expressions of a slightly more general form in which the power class **P[x]** is intersected with some other class. Such rules are useful for computing the **Aclosure** if one considers separately the contributions from subsets of **x** of various different cardinalities. The empty subset of **x** does not contribute anything, of course. There already is a rule for that:

```
In[2]:= image[BIGCAP, set[0]]
```

```
Out[2]= 0
```

Likewise, for the singleton subsets, existing rules suffice:

```
In[3]:= intersection[P[x], range[SINGLETON]]
```

```
Out[3]= image[SINGLETON, x]
```

```
In[4]:= image[BIGCAP, image[SINGLETON, x]]
```

```
Out[4]= x
```

The class of sets of cardinality 2 is **image[PAIRSET, Di]**. In this case it is natural to replace the function **BIGCAP** by its cousin **CAP**, defined as follows:

```
In[5]:= composite[BIGCAP, PAIRSET]
```

```
Out[5]= CAP
```

a membership result about image[BIGCAP, x]

The intersection of all elements of a class **x** is the class **A[x]** = $\bigcap$ **x**. This is a set for any non-empty class. It is a nuisance that the empty set frequently needs to be handled separately, but rarely causes any real difficulty. For any collection **x** of sets, their corresponding intersections belong to the class **image[BIGCAP, x]**. The following theorem provides a new criterion for membership in this class that can be used in addition to an existing one. The difference is only that the condition of non-emptiness has been shifted from the variable x to y. Instead of requiring **x** to be a non-empty member of **y**, one can instead simply require that **y** have no empty members.

Theorem. If **x ∈ y** and if **y** has no empty members, then **A[x] ∈ image[BIGCAP, y]**.

```
In[6]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[and[p1, p3], p4],
           not[implies[and[p1, p2], p4]], {p1 → member[x, y], p2 → not[member[0, y]],
            p3 → not[equal[0, x]], p4 -> member[A[x], image[BIGCAP, y]]}]] // Reverse

Out[6]= or[member[0, y], member[A[x], image[BIGCAP, y]], not[member[x, y]]] == True

In[7]:= or[member[0, y_], member[A[x_], image[BIGCAP, y_]], not[member[x_, y_]]] := True
```

restrictions of SINGLETON and PAIRSET

Any subset of a function **f** is a restriction. On account of this, one can always rewrite any expression of the form **id[y] ∘ f** as **f ∘ id[x]**. Doing so is sometimes awkward, and when **f** is one-to-one, the rewriting could go in either direction. For this reason there is no general rule in the **GOEDEL** program for all functions. In this section only some special cases are considered that appear to be unproblematic.

Theorem. A special rule for the one-to-one function **SINGLETON**.

```
In[12]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
           equal[u, composite[v, id[domain[u]]]],
           {u -> composite[id[P[x]], SINGLETON], v → SINGLETON}] // Reverse

Out[12]= equal[composite[SINGLETON, id[x]], composite[id[P[x]], SINGLETON]] == True

In[13]:= composite[id[P[x_]], SINGLETON] := composite[SINGLETON, id[x]]
```

Theorem. A similar result for the function **PAIRSET**.

```
In[14]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
           equal[u, composite[v, id[domain[u]]]],
           {u -> composite[id[P[x]], PAIRSET], v → PAIRSET}] // Reverse

Out[14]= equal[composite[PAIRSET, id[cart[x, x]]], composite[id[P[x]], PAIRSET]] == True

In[15]:= composite[id[P[x_]], PAIRSET] := composite[PAIRSET, id[cart[x, x]]]
```

BIGCAP and CAP

When considering pairsets, one encounters expressions of the form **image[BIGCAP, P[x] ⋂ image[PAIRSET, Di]]**. These can be rewritten using **CAP**. The following theorem has been generalized slightly by replacing **Di** with an arbitrary class.

Theorem. A connection between images of **BIGCAP** and **CAP**.

```
In[16]:= ImageComp[composite[BIGCAP, id[P[y]]], PAIRSET, x] // Reverse

Out[16]= image[BIGCAP, intersection[image[PAIRSET, x], P[y]]] ==
          image[CAP, composite[id[y], x, id[y]]]

In[17]:= image[BIGCAP, intersection[image[PAIRSET, x_], P[y_]]] :=
          image[CAP, composite[id[y], x, id[y]]]
```

Theorem. If **w ⊂ x** has cardinality 2, then **A[w] ∈ image[CAP, id[x] ∘ Di ∘ id[x]]**. (For greater generality, **Di** can again be replaced with an arbitrary class **y**.)

```
In[18]:= SubstTest[or, member[0, t], member[A[w], image[BIGCAP, t]],
           not[member[w, t]], t → intersection[P[x], image[PAIRSET, y]]] // Reverse

Out[18]= or[member[A[w], image[CAP, composite[id[x], y, id[x]]]],
           not[member[w, image[PAIRSET, y]]], not[subclass[w, x]]] == True

In[19]:= or[member[A[w_], image[CAP, composite[id[x_], y_, id[x_]]]],
           not[member[w_, image[PAIRSET, y_]]], not[subclass[w_, x_]]] := True
```

reassembling Aclosure

When the power class is broken up into the singletons and everything else, one may need the following to put everything back together.

Theorem.

```
In[20]:= SubstTest[union, dif[u, v],
           intersection[u, v], {u → P[x], v → range[SINGLETON]}] // Reverse

Out[20]= union[image[SINGLETON, x], intersection[complement[range[SINGLETON]], P[x]]] == P[x]

In[21]:= union[image[SINGLETON, x_], intersection[complement[range[SINGLETON]], P[x_]]] := P[x]
```

Corollary.

```
In[22]:= SubstTest[image, BIGCAP, union[u, v],
          {u -> image[SINGLETON, x], v -> intersection[complement[range[SINGLETON]], P[x]]}]

Out[22]= union[x, image[BIGCAP, intersection[complement[range[SINGLETON]], P[x]]]] == Aclosure[x]
```

```
In[23]:= union[x_, image[BIGCAP, intersection[complement[range[SINGLETON]], P[x_]]]] :=
          Aclosure[x]
```

an anti-tone property

One often proves that two classes are equal by showing that each is a subclass of the other, and such inclusions may be a consequence of a monotonicity theorem. The intersection constructor is antitone, **x ⊂ y** implies **A[y] ⊂ A[x]**, whereas images reserve inclusions: **x ⊂ y** implies **image[BIGCAP, x] ⊂ image[BIGCAP, y]**. One also may encounter inclusions of the form **x ⊂ image[S, y]** says that each member of **x** is a subset of a member of **y**. Since each subset of a given class holding at least **n** elements contains a subset of with exactly **n** elements, the inclusion **x ⊂ image[S, y]** should hold for the case that **x** is the class of subsets of with exactly **n** elements and **y** is the class of subsets of with at least **n** elements.

Theorem. A rewrite rule for **image[BIGCAP, image[S, x]]**.

```
In[24]:= ImageComp[BIGCAP, S, x] // Reverse

Out[24]= image[BIGCAP, image[S, x]] ==
          union[image[V, intersection[x, set[0]]], image[inverse[S], image[BIGCAP, x]]]

In[25]:= image[BIGCAP, image[S, x_]] :=
          union[image[V, intersection[x, set[0]]], image[inverse[S], image[BIGCAP, x]]]
```

The preceding theorem provides an equation, but involves images of **BIGCAP** that are not restricted to a given power class. An inclusion that does will now be derived.

Lemma.

```
In[50]:= dif[composite[BIGCAP, id[P[x]], S], union[cart[set[0], Aclosure[x]],
             composite[inverse[S], BIGCAP, id[P[x]]]]] // range // Normality

Out[50]= image[BIGCAP,
           intersection[image[S, intersection[complement[P[x]], complement[set[0]]], P[x]]] == 0

In[51]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[52]:= SubstTest[composite, id[range[t]], t, t -> dif[composite[BIGCAP, id[P[x]], S],
            union[cart[set[0], Aclosure[x]], composite[inverse[S], BIGCAP, id[P[x]]]]]]

Out[52]= union[composite[intersection[composite[complement[inverse[S]], BIGCAP],
             composite[BIGCAP, id[P[x]], S]], id[P[x]]], composite[BIGCAP,
            id[P[x]], S, id[intersection[complement[P[x]], complement[set[0]]]]]] == 0

In[53]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[56]:= SubstTest[empty, dif[u, v], {u -> composite[BIGCAP, id[P[x]], S],
           v -> union[cart[set[0], Aclosure[x]], composite[inverse[S], BIGCAP, id[P[x]]]]}]

Out[56]= subclass[composite[BIGCAP, id[P[x]], S],
           union[cart[set[0], Aclosure[x]], composite[inverse[S], BIGCAP, id[P[x]]]]] == True

In[57]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. An inclusion.

```
In[61]:= Map[implies[not[member[0, y]], #] &,
           SubstTest[implies, subclass[u, v], subclass[image[u, y], image[v, y]],
            {u -> composite[BIGCAP, id[P[x]], S], v -> union[cart[set[0], Aclosure[x]],
               composite[inverse[S], BIGCAP, id[P[x]]]]}]] // Reverse

Out[61]= or[member[0, y], subclass[image[BIGCAP, intersection[image[S, y], P[x]]],
            image[inverse[S], image[BIGCAP, intersection[y, P[x]]]]]] == True

In[62]:= or[member[0, y_], subclass[image[BIGCAP, intersection[image[S, y_], P[x_]]],
            image[inverse[S], image[BIGCAP, intersection[y_, P[x_]]]]]] := True
```

serendipity

The following result was discovered, but not used in the course of this study.

Theorem. A simplification rule.

```
In[67]:= ImageComp[S, S, image[PAIRSET, Di]] // Reverse

Out[67]= image[S, intersection[complement[range[SINGLETON]], complement[set[0]]]] ==
          intersection[complement[range[SINGLETON]], complement[set[0]]]

In[68]:= image[S, intersection[complement[range[SINGLETON]], complement[set[0]]]] :=
          intersection[complement[range[SINGLETON]], complement[set[0]]]
```