intersections of sub-binops

Johan G. F. Belinfante
2009 February 25

```
In[1]:= SetDirectory["l:"]; << goedel.09feb24a; << tools.m

        :Package Title: goedel.09feb24a        2009 February 24 at 4:55 p.m.

        It is now:  2009 Feb 25 at 3:21

        Loading Simplification Rules

        TOOLS.M                                        Revised 2009 February 18

        weightlimit = 40
```

summary

The class **BINOPS** of binary operations is not closed under intersections. In this notebook it will be shown that the set **intersection[BINOPS, P[x]]** of binary operations contained in a given binary operation **x** does have the property of being closed under arbitrary intersections. The key idea motivating the derivation is the observation that the class **binclosed[x]** is closed under arbitrary intersections. To make use of this idea one needs some formula for the class of binary operations contained in a given one that brings into play the class **binclosed[x]**. One such formula is in fact already available:

```
In[2]:= image[IMAGE[composite[id[binop[x]], inverse[FIRST]]],
          image[CART, id[binclosed[binop[x]]]]]

Out[2]= intersection[BINOPS, P[binop[x]]]
```

This formula is unfortunately not quite up to the task because many of the currently available **Aclosure** rewrite rules are explicitly limited to sets, whereas the above formula involves a proper class:

```
In[3]:= member[binclosed[binop[x]], V]

Out[3]= False
```

The strategy used in this notebook is to modify the above formula for the class of binary operations contained in **binop[x]** by cutting down the proper class **binclosed[binop[x]]** to a set.

terminology

For brevity it will be said that a class is a **sub-binop** of **x** if it is a binary operation as well as being a subclass of **x**.

derivation

Lemma. This new rewrite rule closely resembles one derived in a recent notebook about sub-binops. The extra wrinkle added here is the introduction of an intersection with an arbitrary class **y**. This small modification is what enables one to cut the proper class **binclosed[binop[x]]** down to a set.

```
In[4]:= SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]],
          {t -> IMAGE[composite[id[binop[x]], inverse[FIRST]]],
           u -> image[CART, id[intersection[y, binclosed[binop[x]]]]],
           v -> image[CART, id[binclosed[binop[x]]]]}] // Reverse

Out[4]= subclass[image[IMAGE[composite[id[binop[x]], inverse[FIRST]]],
           image[CART, id[intersection[y, binclosed[binop[x]]]]]], BINOPS] == True

In[5]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The next step will be to derive an inclusion in the opposite direction.

Lemma.

```
In[6]:= SubstTest[implies, and[member[u, domain[funpart[t]]],
           equal[y, APPLY[funpart[t], u]], member[u, v]], member[y, image[funpart[t], v]],
          {t -> composite[IMAGE[composite[id[binop[x]], inverse[FIRST]]], CART, DUP],
           u → fix[domain[y]],
           v → intersection[binclosed[binop[x]], P[fix[domain[binop[x]]]]]}] // Reverse

Out[6]= or[member[y, image[IMAGE[composite[id[binop[x]], inverse[FIRST]]],
            image[CART, id[intersection[binclosed[binop[x]], P[fix[domain[binop[x]]]]]]]]],
          not[equal[y, composite[binop[x], id[cart[fix[domain[y]], fix[domain[y]]]]]],
          not[member[fix[domain[y]], V]],
          not[subclass[fix[domain[y]], fix[domain[binop[x]]]], not[subclass[
            image[binop[x], cart[fix[domain[y]], fix[domain[y]]], fix[domain[y]]]] == True

In[7]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. If a binary operation **y** is a subclass of **binop[x]** it belongs to a certain class of restrictions of **binop[x]**.

```
In[8]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
           implies[p1, p4], implies[and[p1, p2], p5], implies[p2, p6],
           not[implies[and[p1, p2], p7]], {p1 → member[y, BINOPS], p2 → subclass[y, binop[x]],
            p3 → equal[y, composite[binop[x], id[cart[fix[domain[y]], fix[domain[y]]]]]],
            p4 -> member[fix[domain[y]], V], p5 ->
             subclass[image[binop[x], cart[fix[domain[y]], fix[domain[y]]]], fix[domain[y]]],
            p6 -> subclass[fix[domain[y]], fix[domain[binop[x]]]], p7 ->
             member[y, image[IMAGE[composite[id[binop[x]], inverse[FIRST]]], image[CART, id[
                 intersection[binclosed[binop[x]], P[fix[domain[binop[x]]]]]]]]]}]] // Reverse

Out[8]= or[member[y, image[IMAGE[composite[id[binop[x]], inverse[FIRST]]],
            image[CART, id[intersection[binclosed[binop[x]], P[fix[domain[binop[x]]]]]]]]],
          not[member[y, BINOPS]], not[subclass[y, binop[x]]]] == True
```

```
In[9]:= (% /. {x → x_, y → y_}) /. Equal→ SetDelayed
```

Lemma. Eliminate the variable **y**.

```
In[10]:= Map[equal[V, #] &, dif[intersection[BINOPS, P[binop[x]]],
           image[IMAGE[composite[id[binop[x]], inverse[FIRST]]],
            image[CART, id[intersection[binclosed[binop[x]], P[fix[domain[binop[x]]]]]]]]] //
          complement // Normality]

Out[10]= subclass[intersection[BINOPS, P[binop[x]]],
           image[IMAGE[composite[id[binop[x]], inverse[FIRST]]], image[CART,
            id[intersection[binclosed[binop[x]], P[fix[domain[binop[x]]]]]]]] == True
```

```
In[11]:= (% /. x → x_) /. Equal→ SetDelayed
```

The two inclusions can be combined into an equation.

Theorem. This new formula for the class of sub-binops of a given binary operation resembles the old one except that the proper class **binclosed[binop[x]]** has now been cut down to a set by intersecting with **P[fix[domain[binop[x]]]]**. This intersection retains the property of being closed under arbitrary intersections.

```
In[12]:= SubstTest[and, subclass[u, v], subclass[v, u], {u -> intersection[BINOPS, P[binop[x]]],
           v -> image[IMAGE[composite[id[binop[x]], inverse[FIRST]]],
             image[CART, id[intersection[binclosed[binop[x]], P[fix[domain[binop[x]]]]]]]}]

Out[12]= equal[image[IMAGE[composite[id[binop[x]], inverse[FIRST]]],
            image[CART, id[intersection[binclosed[binop[x]], P[fix[domain[binop[x]]]]]]],
           intersection[BINOPS, P[binop[x]]]] == True

In[13]:= image[IMAGE[composite[id[binop[x_]], inverse[FIRST]]],
           image[CART, id[intersection[binclosed[binop[x_]], P[fix[domain[binop[x_]]]]]]] :=
          intersection[BINOPS, P[binop[x]]]
```

It is now easy to achieve the main goal of this notebook, the **Aclosure** rewrite rule.

Theorem. The class of sub-binops of a given binary operation **binop[x]** is closed under arbitrary intersections.

```
In[14]:= SubstTest[Aclosure, image[IMAGE[oopart[u]], setpart[v]],
           {u -> composite[id[binop[x]], inverse[FIRST]], v -> image[CART,
              id[intersection[binclosed[binop[x]], P[fix[domain[binop[x]]]]]]]}] // Reverse

Out[14]= Aclosure[intersection[BINOPS, P[binop[x]]]] == intersection[BINOPS, P[binop[x]]]

In[15]:= Aclosure[intersection[BINOPS, P[binop[x_]]]] := intersection[BINOPS, P[binop[x]]]
```

the sub-binop generated by any subset

An important application is discussed in this section. It will be shown that if **y** is any subset of **binop[x],** then the intersection of all sub-binops of **binop[x]** that contain a given set **y** is a sub-binop of **binop[x]** that contains **y**. This sub-binop is traditionally called the sub-binop **generated** by the subset **y**. In the terminology of the **GOEDEL** program this intersection is the class **hull[intersection[BINOPS, P[binop[x]]], y]**.

Observation. The sub-binop generated by **binop[x]** is itself.

```
In[16]:= hull[intersection[BINOPS, P[binop[x]]], binop[x]]

Out[16]= binop[x]
```

Theorem. If **y** is a subset of **binop[x]**, then so is **hull[intersection[BINOPS, P[binop[x]]], y]**, and conversely.

```
In[17]:= SubstTest[subclass, hull[t, u], hull[t, v],
           {t -> intersection[BINOPS, P[binop[x]]], u → y, v → binop[x]}] // Reverse

Out[17]= subclass[hull[intersection[BINOPS, P[binop[x]]], y], binop[x]] == subclass[y, binop[x]]

In[18]:= subclass[hull[intersection[BINOPS, P[binop[x_]]], y_], binop[x_]] :=
          subclass[y, binop[x]]
```

Lemma. If **y** is a subset of **binop[x]**, then **y** is a subset of a member of the set of sub-binops of **binop[x]**. This is obviously true because **y** is a subset of **binop[x]** and **binop[x]** is a sub-binop of itself.

```
In[19]:= SubstTest[implies, and[subclass[u, v], member[v, w]], member[u, image[inverse[S], w]],
           {u → y, v → binop[x], w -> intersection[BINOPS, P[binop[x]]]}] // Reverse

Out[19]= or[member[y, image[inverse[S], intersection[BINOPS, P[binop[x]]]]],
           not[subclass[y, binop[x]]]] == True

In[20]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. If **y** is any subset of **binop[x]**, then **hull[intersection[BINOPS, P[binop[x]]], y]** is a binary operation.

```
In[21]:= Map[implies[subclass[y, binop[x]], #] &, SubstTest[member, hull[t, y],
            fix[HULL[t], t -> intersection[BINOPS, P[binop[x]]]]] // Reverse

Out[21]= or[member[hull[intersection[BINOPS, P[binop[x]]], y], BINOPS],
           not[subclass[y, binop[x]]]] == True

In[22]:= or[member[hull[intersection[BINOPS, P[binop[x_]]], y_], BINOPS],
           not[subclass[y_, binop[x_]]]] := True
```