images and inverse images of CURRY

Johan G. F. Belinfante
2006 November 23

```
In[1]:= SetDirectory["l:"]; << goedel87.16b; << tools.m

        :Package Title: goedel87.16b          2006 November 16 at 2:45 p.m.

        It is now:  2006 Nov 23 at 15:36

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 November 22

        weightlimit = 40
```

summary

Applying **CURRY** to a mapping from **cart[x, y]** to **z** yields a mapping from **x** to **map[y, z]**, provided **y** is not empty. Applying **inverse[CURRY]** allows one to go in the opposite direction. An unconditional rewrite rule expressing these facts is derived in this notebook. A variable-free formulation is obtained by using reification. Membership rules for images and inverse images are derived in the final two sections of this notebook.

an image equation

A conditional equation for **image[CURRY,map[cart[x,y],z]]** is derived in this section by combining two inclusions, one obtained by considering the process of currying, and another by considering the reverse process of uncurrying. For currying, one obtains this inclusion:

```
In[2]:= Map[equal[V, #] &, SubstTest[class, t,
          or[equal[0, y], member[APPLY[funpart[u], setpart[t]], v], not[member[setpart[t], w]]],
          {u → CURRY, v -> map[x, map[y, z]], w -> map[cart[x, y], z]}]]

Out[2]= or[equal[0, y],
          subclass[map[cart[x, y], z], image[inverse[CURRY], map[x, map[y, z]]]]] == True

In[3]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

One can convert this inclusion for inverse images to one for direct images by using this lemma:

```
In[4]:= SubstTest[implies, subclass[u, image[inverse[funpart[t]], v]],
          subclass[image[funpart[t], u], v], t → CURRY] // Reverse

Out[4]= or[not[subclass[u, image[inverse[CURRY], v]]], subclass[image[CURRY, u], v]] == True
```

```
In[5]:= or[not[subclass[u_, image[inverse[CURRY], v_]]], subclass[image[CURRY, u_], v_]] := True
```

The result of doing so yields this inclusion:

```
In[6]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p2, p3], not[implies[p1, p3]], {p1 → not[equal[0, y]],
            p2 -> subclass[map[cart[x, y], z], image[inverse[CURRY], map[x, map[y, z]]]],
            p3 → subclass[image[CURRY, map[cart[x, y], z]], map[x, map[y, z]]]}]] // Reverse

Out[6]= or[equal[0, y], subclass[image[CURRY, map[cart[x, y], z]], map[x, map[y, z]]]] == True

In[7]:= or[equal[0, y_],
          subclass[image[CURRY, map[cart[x_, y_], z_]], map[x_, map[y_, z_]]]] := True
```

By considering the reverse process of uncurrying, an inclusion in the other direction is obtained:

```
In[8]:= Map[equal[V, #] &, SubstTest[class, t,
          or[equal[0, y], member[APPLY[funpart[u], setpart[t]], v], not[member[setpart[t], w]]],
          {u → inverse[CURRY], v -> map[cart[x, y], z], w -> map[x, map[y, z]]}]]

Out[8]= or[equal[0, y], subclass[map[x, map[y, z]], image[CURRY, map[cart[x, y], z]]]] == True

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

These two inclusions can be combined to yield a conditional equation:

```
In[10]:= SubstTest[and, implies[p, subclass[u, v]], implies[p, subclass[v, u]],
          {p → not[empty[y]], u -> image[CURRY, map[cart[x, y], z]], v -> map[x, map[y, z]]}]

Out[10]= or[equal[0, y], equal[image[CURRY, map[cart[x, y], z]], map[x, map[y, z]]]] == True

In[11]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Comment. The condition that **y** be not empty can not simply be omitted. When **y = 0**, the equation reduces to this statement:

```
In[12]:= equal[image[CURRY, map[cart[x, y], z]], map[x, map[y, z]]] /. y → 0

Out[12]= equal[0, x]
```

To get an unconditional equation, the equation must be modified. The condition that **y** be nonempty can be eliminated by slightly modifying the equation. This lemma will be needed:

```
In[13]:= SubstTest[implies, not[equal[0, y]],
           equal[image[CURRY, map[cart[t, y], z]], map[t, map[y, z]]],
           t → intersection[x, image[V, y]]] // Reverse

Out[13]= or[equal[0, y], equal[image[CURRY, map[cart[x, y], z]],
            map[intersection[x, image[V, y]], map[y, z]]]] == True

In[14]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

The following unconditional equation holds, and is made into a rewrite rule:

```
In[15]:= SubstTest[and, implies[p, q], or[p, q], {p → not[empty[y]],
           q → equal[image[CURRY, map[cart[x, y], z]],
             map[intersection[x, image[V, y]], map[y, z]]]}]

Out[15]= equal[image[CURRY, map[cart[x, y], z]],
           map[intersection[x, image[V, y]], map[y, z]]] == True

In[16]:= image[CURRY, map[cart[x_, y_], z_]] := map[intersection[x, image[V, y]], map[y, z]]
```

variable-free formulation

To obtain a variable-free formulation, two simplification rules are needed. The first one is fairly obvious:

```
In[17]:= Assoc[id[FUNS], composite[inverse[DORA], cross[Id, inverse[S]]], cross[x, Id]]

Out[17]= composite[id[FUNS], inverse[DORA], cross[x, inverse[S]]] ==
          composite[inverse[E], MAP, cross[x, Id]]

In[18]:= composite[id[FUNS], inverse[DORA], cross[x_, inverse[S]]] :=
          composite[inverse[E], MAP, cross[x, Id]]
```

Based on the expectation that the following quantity will appear in the final result, a normalization result is sought:

```
In[19]:= composite[inverse[E], MAP, cross[Id, MAP], ASSOC] // ReifTriNormality // Reverse

Out[19]= composite[id[FUNS], intersection[composite[inverse[IMAGE[FIRST]], FIRST, FIRST],
           composite[inverse[IMAGE[SECOND]], inverse[S], MAP, cross[SECOND, Id]]]] ==
          composite[inverse[E], MAP, cross[Id, MAP], ASSOC]

In[20]:= composite[id[FUNS], intersection[composite[inverse[IMAGE[FIRST]], FIRST, FIRST],
           composite[inverse[IMAGE[SECOND]], inverse[S], MAP, cross[SECOND, Id]]]] :=
          composite[inverse[E], MAP, cross[Id, MAP], ASSOC]
```

With these two simplification rules in place, reification yields this variable-free result:

```
In[21]:= Map[composite[VERTSECT[#], id[cart[cart[V, V], V]]] &, SubstTest[reify, x,
           image[z, map[cart[first[first[x]], second[first[x]]], second[x]]], z → CURRY]]

Out[21]= composite[IMAGE[CURRY], MAP, cross[CART, Id]] ==
          union[cart[cart[cart[V, set[0]], V], set[set[0]]],
           composite[MAP, cross[Id, MAP], ASSOC, id[cart[cart[V, complement[set[0]]], V]]]]

In[22]:= composite[IMAGE[CURRY], MAP, cross[CART, Id]] :=
          union[cart[cart[cart[V, set[0]], V], set[set[0]]],
           composite[MAP, cross[Id, MAP], ASSOC, id[cart[cart[V, complement[set[0]]], V]]]]
```

additional facts about inverse images

The following implication resembles one derived in an earlier section.

```
In[23]:= SubstTest[implies, subclass[u, image[inverse[funpart[t]], v]],
           subclass[image[funpart[t], u], v], t → inverse[CURRY]] // Reverse

Out[23]= or[not[subclass[u, image[CURRY, v]]], subclass[image[inverse[CURRY], u], v]] == True

In[24]:= or[not[subclass[u_, image[CURRY, v_]]], subclass[image[inverse[CURRY], u_], v_]] := True
```

Theorem.

```
In[25]:= SubstTest[implies, member[APPLY[funpart[t], w], x],
           member[w, image[inverse[funpart[t]], x]], t → CURRY] // Reverse

Out[25]= or[member[w, image[inverse[CURRY], x]], not[member[APPLY[CURRY, w], x]]] == True

In[26]:= or[member[w_, image[inverse[CURRY], x_]], not[member[APPLY[CURRY, w_], x_]]] := True
```

Converse theorem.

```
In[27]:= SubstTest[implies, member[w, image[inverse[funpart[t]], x]],
           member[APPLY[funpart[t], w], x], t → CURRY] // Reverse

Out[27]= or[member[APPLY[CURRY, w], x], not[member[w, image[inverse[CURRY], x]]]] == True

In[28]:= or[member[APPLY[CURRY, w_], x_], not[member[w_, image[inverse[CURRY], x_]]]] := True
```

The following somewhat trivial general fact is needed as a simplification rule.

```
In[29]:= Map[not, SubstTest[and, implies[p1, p2], not[implies[p1, p3]],
             {p1 → member[APPLY[x, y], z], p2 → member[y, domain[x]], p3 → member[y, V]}]] // Reverse

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

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

Theorem.

```
In[31]:= SubstTest[implies, member[w, image[u, x]],
           member[w, range[u]], u → inverse[CURRY]] // Reverse // MapNotNot

Out[31]= or[not[member[w, image[inverse[CURRY], x]]], subclass[w, cart[cart[V, V], V]]] == True

In[32]:= or[not[member[w_, image[inverse[CURRY], x]]], subclass[w_, cart[cart[V, V], V]]] := True
```

Corollary.

```
In[33]:= Map[not, SubstTest[and, implies[p1, p2], not[implies[p1, p3]],
            {p1 -> member[APPLY[CURRY, w], x], p2 -> member[w, image[inverse[CURRY], x]],
             p3 -> subclass[w, cart[cart[V, V], V]]}]] // Reverse

Out[33]= or[not[member[APPLY[CURRY, w], x]], subclass[w, cart[cart[V, V], V]]] == True

In[34]:= or[not[member[APPLY[CURRY, w_], x_]], subclass[w_, cart[cart[V, V], V]]] := True
```

---

additional facts about images

Theorem.

```
In[35]:= SubstTest[implies, member[APPLY[funpart[t], w], x],
           member[w, image[inverse[funpart[t]], x]], t → inverse[CURRY]] // Reverse

Out[35]= or[member[w, image[CURRY, x]], not[member[APPLY[inverse[CURRY], w], x]]] == True

In[36]:= or[member[w_, image[CURRY, x_]], not[member[APPLY[inverse[CURRY], w_], x_]]] := True
```

Converse theorem.

```
In[37]:= SubstTest[implies, member[w, image[inverse[funpart[t]], x]],
           member[APPLY[funpart[t], w], x], t → inverse[CURRY]] // Reverse

Out[37]= or[member[APPLY[inverse[CURRY], w], x], not[member[w, image[CURRY, x]]]] == True

In[38]:= or[member[APPLY[inverse[CURRY], w_], x_], not[member[w_, image[CURRY, x_]]]] := True
```

Theorem.

```
In[39]:= SubstTest[implies, and[member[w, u], subclass[u, v]],
            member[w, v], {u -> image[CURRY, x], v -> FUNS}] // Reverse // MapNotNot

Out[39]= or[FUNCTION[w], not[member[w, image[CURRY, x]]]] == True

In[40]:= or[FUNCTION[w_], not[member[w_, image[CURRY, x_]]]] := True
```

Corollary.

```
In[41]:= Map[not, SubstTest[and, implies[p1, p2], not[implies[p1, p3]],
            {p1 -> member[APPLY[inverse[CURRY], w], x],
             p2 -> member[w, image[CURRY, x]], p3 → FUNCTION[w]}]] // Reverse

Out[41]= or[FUNCTION[w], not[member[APPLY[inverse[CURRY], w], x]]] == True

In[42]:= or[FUNCTION[w_], not[member[APPLY[inverse[CURRY], w_], x_]]] := True
```

Lemma.

```
In[43]:= Map[and[#, member[0, range[U[image[CURRY, x]]]]] &, SubstTest[subclass,
           image[t, x], range[t], t -> composite[inverse[E], IMAGE[SECOND], CURRY]]]

Out[43]= member[0, range[U[image[CURRY, x]]]] == False

In[44]:= member[0, range[U[image[CURRY, x_]]]] := False
```

Theorem.

```
In[45]:= SubstTest[implies, and[member[w, u], subclass[u, v]], member[w, v],
            {u -> image[CURRY, x], v → P[complement[cart[V, set[0]]]]}] // Reverse // MapNotNot

Out[45]= or[not[member[0, range[w]]], not[member[w, image[CURRY, x]]]] == True

In[46]:= or[not[member[0, range[w_]]], not[member[w_, image[CURRY, x_]]]] := True
```

Corollary.

```
In[47]:= Map[not, SubstTest[and, implies[p1, p2], not[implies[p1, p3]],
            {p1 -> member[APPLY[inverse[CURRY], w], x],
             p2 -> member[w, image[CURRY, x]], p3 → not[member[0, range[w]]]}]] // Reverse

Out[47]= or[not[member[0, range[w]]], not[member[APPLY[inverse[CURRY], w], x]]] == True

In[48]:= or[not[member[0, range[w_]]], not[member[APPLY[inverse[CURRY], w_], x_]]] := True
```

Lemma.

```
In[49]:= SubstTest[subclass, image[t, x], range[t],
           t -> composite[inverse[E], IMAGE[SECOND], CURRY]] // Reverse

Out[49]= subclass[U[range[U[image[CURRY, x]]]], cart[V, V]] == True

In[50]:= subclass[U[range[U[image[CURRY, x_]]]], cart[V, V]] := True
```

Theorem.

```
In[51]:= SubstTest[implies,
            and[member[w, u], subclass[u, v]], member[w, v], {u -> image[CURRY, x],
             v → image[inverse[IMAGE[SECOND]], P[P[cart[V, V]]]]}] // Reverse // MapNotNot

Out[51]= or[not[member[w, image[CURRY, x]]], subclass[U[range[w]], cart[V, V]]] == True

In[52]:= or[not[member[w_, image[CURRY, x_]]], subclass[U[range[w_]], cart[V, V]]] := True
```

Corollary.

```
In[53]:= Map[not, SubstTest[and, implies[p1, p2], not[implies[p1, p3]],
            {p1 -> member[APPLY[inverse[CURRY], w], x], p2 -> member[w, image[CURRY, x]],
             p3 → subclass[U[range[w]], cart[V, V]]}]] // Reverse

Out[53]= or[not[member[APPLY[inverse[CURRY], w], x]], subclass[U[range[w]], cart[V, V]]] == True
```

```
In[54]:= or[not[member[APPLY[inverse[CURRY], w_], x_]],
            subclass[U[range[w_]], cart[V, V]]] := True
```