cross-sections of relations

Johan G. F. Belinfante
2004 October 28

```
In[1]:= SetDirectory["i:"]; << goedel62.26b; << tools.m

        :Package Title: goedel62.26b          2004 October 26 at 10:50 p.m.

        It is now:  2004 Oct 28 at 20:0

        Loading Simplification Rules

        TOOLS.M                      Revised 2004 October 26

        weightlimit = 40
```

summary

One of the nicer formulations of the axiom of choice is that any cartesian product of a family of nonempty sets with a nonempty index set is not empty. The cartesian product of a family of sets is defined in this notebook as the set of cross-sections of a relation. One should think of an indexed family of sets as the collection of vertical sections of a relation. The **index set** is just the domain of the relation. A cross-section of **x** is a function which is a subclass of **x** having the same domain as **x**. The cartesian product **X[x]** is defined to be the set of cross-sections of **x**.

definition

The following membership rule serves to define **X[x]**.

```
In[2]:= member[w_, X[x_]] :=
         and[equal[domain[w], domain[x]], FUNCTION[w], member[w, V], subclass[w, x]]
```

normalization

```
In[3]:= X[x] // Normality // Reverse

Out[3]= intersection[map[domain[x], V], P[x]] == X[x]
```

```
In[4]:= intersection[map[domain[x_], V], P[x_]] := X[x]
```

reify rule

```
In[5]:= SubstTest[reify, x, intersection[map[domain[f[x]], v], P[f[x]]], v → V]

Out[5]= reify[x, X[f[x]]] ==
         composite[id[FUNS], intersection[composite[inverse[IMAGE[FIRST]],
            VERTSECT[composite[FIRST, reify[x, f[x]]]]], inverse[LB[reify[x, f[x]]]]]]

In[6]:= reify[x_, X[y_]] :=
         composite[id[FUNS], intersection[composite[inverse[IMAGE[FIRST]],
            VERTSECT[composite[FIRST, reify[x, y]]]], inverse[LB[reify[x, y]]]]]
```

simplification rules

A few simplification rules are derived here. More will be added later as needed.

```
In[7]:= Map[equal[0, #] &, dif[X[x], P[x]] // Normality]

Out[7]= subclass[U[X[x]], x] == True

In[8]:= subclass[U[X[x_]], x_] := True

In[9]:= equal[intersection[P[x], X[x]], X[x]]

Out[9]= True

In[10]:= intersection[P[x_], X[x_]] := X[x]
```

There are various related rules. For example:

```
In[11]:= AssInt[X[x], P[x], P[y]]

Out[11]= intersection[P[intersection[x, y]], X[x]] == intersection[P[y], X[x]]

In[12]:= intersection[P[intersection[x_, y_]], X[x_]] := intersection[P[y], X[x]]

In[13]:= AssInt[map[domain[x], V], P[cart[V, V]], P[x]]

Out[13]= intersection[map[domain[x], V], P[composite[Id, x]]] == X[x]

In[14]:= intersection[map[domain[x_], V], P[composite[Id, x_]]] := X[x]

In[15]:= X[composite[Id, x]] // Normality

Out[15]= X[composite[Id, x]] == X[x]
```

```
In[16]:= X[composite[Id, x_]] := X[x]

In[17]:= AssInt[map[domain[x], V], P[x], P[composite[Id, x]]] // Reverse

Out[17]= intersection[P[composite[Id, x]], X[x]] == X[x]

In[18]:= intersection[P[composite[Id, x_]], X[x_]] := X[x]
```

some basic examples

```
In[19]:= X[0] // Normality

Out[19]= X[0] == singleton[0]

In[20]:= X[0] := singleton[0]

In[21]:= X[V] // Normality

Out[21]= X[V] == 0

In[22]:= X[V] := 0

In[23]:= X[cart[x, V]] // Normality

Out[23]= X[cart[x, V]] == map[x, V]

In[24]:= X[cart[x_, V]] := map[x, V]
```

function rule

The cartesian product of a family of singletons is a singleton, assuming that the family is a set.

```
In[25]:= SubstTest[implies,
          and[equal[domain[y], domain[z]], equal[y, composite[z, id[domain[y]]]]],
          equal[y, composite[Id, z]], z → funpart[x]]

Out[25]= or[equal[y, funpart[x]], not[equal[domain[y], domain[funpart[x]]]],
           not[subclass[y, funpart[x]]]] == True

In[26]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed

In[27]:= equiv[equal[y, funpart[x]],
          and[equal[domain[y], domain[funpart[x]]], subclass[y, funpart[x]]]]

Out[27]= True
```

```
In[28]:= and[equal[domain[y_], domain[funpart[x_]]], subclass[y_, funpart[x_]]] :=
          equal[y, funpart[x]]

In[29]:= equal[intersection[FUNS, singleton[funpart[x]]], singleton[funpart[x]]]

Out[29]= True

In[30]:= intersection[FUNS, singleton[funpart[x_]]] := singleton[funpart[x]]

In[31]:= X[funpart[x]] // Normality

Out[31]= X[funpart[x]] == singleton[funpart[x]]

In[32]:= X[funpart[x_]] := singleton[funpart[x]]

In[33]:= SubstTest[implies, equal[y, funpart[x]], equal[X[y], singleton[y]], y → x]

Out[33]= or[equal[singleton[x], X[x]], not[FUNCTION[x]]] == True

In[34]:= or[equal[singleton[x_], X[x_]], not[FUNCTION[x_]]] := True
```

The following conditional rule is of a similar nature.

```
In[35]:= X[x_] := singleton[x] /; FUNCTION[x]
```

---

sethood rule

```
In[36]:= Map[implies[member[x, y], #] &, SubstTest[implies,
           and[subclass[u, v], member[v, V]], member[u, V], {u → X[x], v → P[x]}]]

Out[36]= or[member[X[x], V], not[member[x, y]]] == True

In[37]:= or[member[X[x_], V], not[member[x_, y_]]] := True
```

---

empty X[x]

```
In[38]:= Map[equal[0, #] &, dif[X[x], map[domain[x], V]] // Normality]

Out[38]= subclass[X[x], map[domain[x], V]] == True

In[39]:= subclass[X[x_], map[domain[x_], V]] := True

In[40]:= Map[implies[not[member[domain[x], V]], #] &, SubstTest[and, subclass[u, v],
           equal[0, v], {u → X[x], v -> map[domain[x], V]}]] // Reverse

Out[40]= or[equal[0, X[x]], member[domain[x], V]] == True

In[41]:= or[equal[0, X[x_]], member[domain[x_], V]] := True
```