# abstraction

*Johan G. F. Belinfante*
*2003 July 6*

```
In[1]:= << goedel52.s46; << tools.m

        :Package Title: goedel52.s46      2003 July 6 at 1:05 p.m.

        It is now:  2003 Jul 8 at 10:36

        Loading Simplification Rules

        TOOLS.M                       Revised 2003 July 8

        weightlimit = 40
```

## ■ summary

The process called abstraction in this notebook is remininscent of abstraction in combinatory logic, but it it far less general. As an application of these ideas, some connections are derived between certain constructors in Gödel's class theory.

## ■ definition of abstraction

The definition of **abstract** is given in the tools.m file.

```
In[2]:= Begin["Goedel'Private'"]

Out[2]= Goedel'Private'
```

The definition of abstraction is in terms of a more general process called reification.

```
In[3]:= ?? abstract

        abstract[x,F[x]]  yields a class  y  satisfying  F[x] = image[y,x]  if such a class exists

        abstract[x_, y_] := composite[reify[x, y], SINGLETON]

In[4]:= ? reify

        reify[x,F[x]] is the relation of all pair[x,y] with y belonging to F[x]
```

The simplest case is this:

```
In[5]:= abstract[x, image[z, x]]

Out[5]= composite[Id, z]
```

## ■ unary functors

Not all unary functors can be written in the form **F[x] = image[y,x]**. This does work for these:

```
In[6]:= Select[UnaryFunctors, equal[#[x], image[abstract[y, #[y]], x]] &]

Out[6]= {domain, fix, flip, id, inverse, range, rotate, twist, U}
```

The results of abstraction are:

```
In[7]:= Map[abstract[x, #[x]] &, %]

Out[7]= {FIRST, inverse[DUP], cross[SWAP, Id], DUP, SWAP, SECOND, ROT, TWIST, inverse[E]}
```

It is easy to verify that these results are correct:

```
In[8]:= Map[image[#, x] &, %]

Out[8]= {domain[x], fix[x], composite[x, SWAP],
         id[x], inverse[x], range[x], rotate[x], twist[x], U[x]}
```

## ■ binary functors

For a binary functor, one can abstract twice, once for each variable. The simplest case is this:

```
In[9]:= abstract[x, abstract[y, image[image[z, x], y]]]

Out[9]= composite[id[cart[V, V]], z]
```

Not all binary functors can be expressed in the form **F[x,y] = image[image[z,x],y]**. It does work for these:

```
In[10]:= Select[BinaryFunctors,
          equal[#[x, y], image[image[abstract[u, abstract[v, #[u, v]]], x], y]] &]

Out[10]= {cart, composite, cross, image, intersection}
```

The results of abstracting twice in these cases is:

```
In[11]:= Map[abstract[u, abstract[v, #[u, v]]] &, %] // TableForm

Out[11]//TableForm=
        composite[id[inverse[SECOND]], inverse[SECOND], inverse[FIRST]]
        composite[cross[Id, SWAP], inverse[RIF]]
        composite[id[cross[inverse[SECOND], inverse[SECOND]]], inverse[SECOND], cross[inverse[F
        id[cart[V, V]]
        DUP
```

The following verifies that this works:

```
In[12]:= Map[image[image[#, x], y] &, %]

Out[12]= {cart[x, y], composite[x, y], cross[x, y], image[x, y], intersection[x, y]}
```

## ■ abstracting in the opposite order

Given an expression **F[x,y]**, can one write this in any of the forms **image[image[u,x],y]**, **image[image[v,y],x]** or **image[w,cart[x,y]]**? In this section it is shown that, if any of these expressions is possible, then all are possible, and some formulas relating **u**, **v** and **w** are derived.

One can use abstraction to discover how **u** and **v** are related:

```
In[13]:= Map[abstract[y, abstract[x, #]] &,  image[image[u, x], y] == image[image[v, y], x]]

Out[13]= inverse[rotate[composite[inverse[u], SWAP]]] == composite[id[cart[V, V]], v]
```

This verifies the relation between the **u** and **v** forms:

```
In[14]:= Map[image[image[#, y], x] &, inverse[rotate[composite[inverse[u], SWAP]]] == v]

Out[14]= image[image[u, x], y] == image[image[v, y], x]
```

How to get the **cart** form can be discovered as follows:

```
In[15]:= Map[abstract[x, abstract[y, #]] &, image[image[u, x], y] == image[w, cart[x, y]]]

Out[15]= composite[id[cart[V, V]], u] == composite[SWAP, inverse[rotate[composite[w, SWAP]]]]
```

One can solve for **w** in terms of **u** as follows:

```
In[16]:= Map[rotate[inverse[#]] &, %]

Out[16]= rotate[inverse[u]] == composite[w, id[cart[V, V]]]
```

The formulas discovered are easily verified:

```
In[17]:= Map[image[image[#, x], y] &,  u == composite[SWAP, inverse[rotate[composite[w, SWAP]]]]]

Out[17]= image[image[u, x], y] == image[w, cart[x, y]]
```

```
In[18]:= Map[image[#, cart[x, y]] &, rotate[inverse[u]] == w]

Out[18]= image[image[u, x], y] == image[w, cart[x, y]]
```

Comment: The **cart** formula can be flipped, if desired:

```
In[19]:= image[flip[w], cart[x, y]]

Out[19]= image[w, cart[y, x]]
```

## ■ connections between certain binary functors

The process or rotating the inverse can be applied to each of the classes obtained by double abstraction from the binary functors **cart, composite, cross, image, intersection**, yielding in each case a function.

```
In[20]:= Map[rotate[inverse[#]] &,
          {composite[id[inverse[SECOND]], inverse[SECOND], inverse[FIRST]],
           composite[cross[Id, SWAP], inverse[RIF]],
           composite[id[cross[inverse[SECOND], inverse[SECOND]]], inverse[SECOND],
            cross[inverse[FIRST], inverse[FIRST]]], id[cart[V, V]], DUP}]

Out[20]= {id[cart[V, V]], composite[SWAP, RIF], TWIST,
          composite[SECOND, FIRST, id[FIRST]], inverse[DUP]}
```

All of these binary functors can therefore be expressed in terms of **cart**.

```
In[21]:= Map[image[#, cart[x, y]] &, %]

Out[21]= {cart[x, y], composite[x, y], cross[x, y], image[x, y], intersection[x, y]}
```

Comment: the name **RIF** stands for rotation−invariantfunction:

```
In[22]:= FUNCTION[RIF]

Out[22]= True

In[23]:= rotate[RIF]

Out[23]= RIF
```

## ■ an application

For each of the binary functors considered in the preceding section, there is a corresponding binary function:

```
In[24]:= Map[lambda[pair[x, y], #] &,
          {cart[x, y], composite[x, y], cross[x, y], image[x, y], intersection[x, y]}]

Out[24]= {CART, COMPOSE, CROSS, IMG, CAP}
```

These functions can all be expressed in terms of **CART** using the results obtained above. These formulas also involve the function

```
In[25]:= lambda[y, image[x, y]]

Out[25]= IMAGE[x]
```

These are the formulas one obtains:

```
In[26]:= composite[IMAGE[composite[SWAP, RIF]], CART]

Out[26]= COMPOSE

In[27]:= composite[IMAGE[TWIST], CART]

Out[27]= CROSS

In[28]:= composite[IMAGE[composite[SECOND, FIRST, id[FIRST]]], CART]

Out[28]= IMG
```

```
In[29]:= composite[IMAGE[inverse[DUP]], CART]

Out[29]= CAP
```

Comment: the formulas for the cases of **COMPOSE** and **IMG** can also be written compactly as follows:

```
In[30]:= composite[IMAGE[twist[ROT]], CART]

Out[30]= COMPOSE

In[31]:= composite[IMAGE[rotate[Id]], CART]

Out[31]= IMG
```