

monotonicity rules for map and MAP

Johan G. F. Belinfante
2003 May 7

```
In[1]:= << goedel52.r63; << tools.m

:Package Title: goedel52.r63          2003 May 6 at 5:50 a.m.

It is now: 2003 May 12 at 11:16

Loading Simplification Rules

TOOLS.M                               Revised 2003 May 11

weightlimit = 40
```

■ summary

A monotonicity rule for **map**[**x**,**y**], and a corresponding property for the function **MAP** are derived in this notebook. The class of mappings from **x** to **y** is defined by:

```
In[2]:= class[w, and[FUNCTION[w], equal[domain[w], x], subclass[range[w], y]]]
```

```
Out[2]= map[x, y]
```

The binary function **MAP** is defined by:

```
In[3]:= lambda[pair[x, y], map[x, y]]
```

```
Out[3]= MAP
```

The derivation is mainly of interest as an illustration of how one goes about eliminating set-variables from statements.

■ monotonicity of map[x,y]

The constructor **map**[**x**,**y**] is monotone with respect to its second argument.

```
In[4]:= implies[subclass[y, z], subclass[map[x, y], map[x, z]]] // AssertTest
```

```
Out[4]= or[not[subclass[y, z]], subclass[map[x, y], map[x, z]]] == True
```

```
In[5]:= or[not[subclass[y_, z_]], subclass[map[x_, y_], map[x_, z_]]] := True
```

■ monotonicity in terms of MAP

The derivation of a variable-free version of this law is a bit tricky. One way to do it is to make use of the function `LEFT[x]`.

```
In[6]:= class[pair[u, pair[v, w]], and[equal[v, x], equal[u, w]]]
```

```
Out[6]= LEFT[x]
```

The derivation begins with an application of the tool `VSRerenormality`. This tool builds in two `assert`'s.

```
In[7]:= intersection[S, composite[inverse[LEFT[x]],
    inverse[MAP], complement[S], MAP, LEFT[x]]] // VSRerenormality
```

```
Out[7]= intersection[S,
    composite[inverse[LEFT[x]], inverse[MAP], complement[S], MAP, LEFT[x]]] == 0
```

```
In[8]:= intersection[S,
    composite[inverse[LEFT[x_]], inverse[MAP], complement[S], MAP, LEFT[x_]]] := 0
```

From this one derives a conditional inclusion:

```
In[9]:= SubstTest[equal, 0, dif[u, v],
    {u -> S, v -> complement[composite[inverse[LEFT[x]], inverse[MAP],
    complement[S], MAP, LEFT[x]]]}] // Reverse
```

```
Out[9]= or[not[member[x, V]],
    subclass[S, composite[inverse[LEFT[x]], inverse[MAP], S, MAP, LEFT[x]]]] == True
```

```
In[10]:= or[not[member[x_, V]],
    subclass[S, composite[inverse[LEFT[x_]], inverse[MAP], S, MAP, LEFT[x_]]]] := True
```

The condition that `x` be a set is necessary as well as sufficient:

```
In[11]:= SubstTest[implies, subclass[u, v],
    subclass[image[inverse[u], singleton[0]], image[inverse[v], singleton[0]]],
    {u -> S, v -> composite[inverse[LEFT[x]], inverse[MAP], S, MAP, LEFT[x]]}]
```

```
Out[11]= or[member[x, V],
    not[subclass[S, composite[inverse[LEFT[x]], inverse[MAP], S, MAP, LEFT[x]]]]] == True
```

```
In[12]:= or[member[x_, V], not[
    subclass[S, composite[inverse[LEFT[x_]], inverse[MAP], S, MAP, LEFT[x_]]]]] := True
```

Thus one can introduce a rewrite rule:

```
In[13]:= equiv[
    subclass[S, composite[inverse[LEFT[x]], inverse[MAP], S, MAP, LEFT[x]], member[x, V]]
```

```
Out[13]= True
```

```
In[14]:= subclass[S, composite[inverse[LEFT[x_]], inverse[MAP], S, MAP, LEFT[x_]]] :=
    member[x, V]
```

Eliminating the `inverse`'s is easy:

```
In[15]:= SubstTest[implies, subclass[u, v], subclass[composite[w, u], composite[w, v]],
  {u -> S, v -> composite[inverse[LEFT[x]], inverse[MAP], S, MAP, LEFT[x]],
   w -> composite[MAP, LEFT[x]]}]
```

```
Out[15]= or[not[member[x, V]],
  subclass[composite[MAP, LEFT[x], S], composite[S, MAP, LEFT[x]]] == True
```

```
In[16]:= or[not[member[x_, V]],
  subclass[composite[MAP, LEFT[x_], S], composite[S, MAP, LEFT[x_]]] := True
```

This conditional inclusion also holds without the membership condition:

```
In[17]:= SubstTest[implies, equal[0, u], subclass[u, v],
  {u -> composite[MAP, LEFT[x], S], v -> composite[S, MAP, LEFT[x]]}]
```

```
Out[17]= or[member[x, V],
  subclass[composite[MAP, LEFT[x], S], composite[S, MAP, LEFT[x]]] == True
```

```
In[18]:= or[member[x_, V],
  subclass[composite[MAP, LEFT[x_], S], composite[S, MAP, LEFT[x_]]] := True
```

Thus one can simplify the statement:

```
In[19]:= SubstTest[and, implies[p1, p2], implies[not[p1], p2],
  {p1 -> member[x, V],
   p2 -> subclass[composite[MAP, LEFT[x], S], composite[S, MAP, LEFT[x]]]} // Reverse
```

```
Out[19]= subclass[composite[MAP, LEFT[x], S], composite[S, MAP, LEFT[x]]] == True
```

```
In[20]:= subclass[composite[MAP, LEFT[x_], S], composite[S, MAP, LEFT[x_]]] := True
```

This can be viewed as a statement of subcommutativity.

```
In[21]:= subcommute[composite[MAP, LEFT[x]], S]
```

```
Out[21]= True
```

The variable x that appears in this statement can be eliminated as follows:

```
In[22]:= Map[equal[V, #] &,
  SubstTest[class, x, subcommute[composite[z, LEFT[x]], S], z -> MAP]] // Reverse
```

```
Out[22]= subclass[composite[FIRST, intersection[composite[inverse[MAP], FIRST],
  composite[inverse[SECOND], S, SECOND]]], rotate[composite[S, MAP, SWAP]]] == True
```

```
In[23]:= subclass[composite[FIRST, intersection[composite[inverse[MAP], FIRST],
  composite[inverse[SECOND], S, SECOND]]], rotate[composite[S, MAP, SWAP]]] := True
```

This variable-free statement can be simplified by applying **flip** and **rotate**:

```
In[24]:= SubstTest[implies, subclass[u, v], subclass[flip[u], flip[v]],
  {u -> composite[FIRST, intersection[composite[inverse[MAP], FIRST],
  composite[inverse[SECOND], S, SECOND]]], v -> rotate[composite[S, MAP, SWAP]]}]
```

```
Out[24]= subclass[composite[FIRST, intersection[
  composite[inverse[MAP], SECOND], composite[inverse[SECOND], S, FIRST]]],
  composite[rotate[composite[S, MAP, SWAP]], SWAP]] == True
```

```
In[25]:= subclass[composite[FIRST, intersection[
  composite[inverse[MAP], SECOND], composite[inverse[SECOND], S, FIRST]]],
  composite[rotate[composite[S, MAP, SWAP]], SWAP]] := True
```

```
In[26]:= SubstTest[subclass, rotate[u], rotate[v],  
  {u -> composite[FIRST, intersection[  
    composite[inverse[MAP], SECOND], composite[inverse[SECOND], S, FIRST]]},  
  v -> composite[rotate[composite[S, MAP, SWAP]], SWAP}}
```

```
Out[26]= subclass[composite[MAP, cross[Id, S]], composite[S, MAP]] == True
```

This is the final version of the variable-free formulation of the monotonicity rule:

```
In[27]:= subclass[composite[MAP, cross[Id, S]], composite[S, MAP]] := True
```