monotone and subtwine

Johan G. F. Belinfante
2007 June 16

```
In[1]:= SetDirectory["l:"]; << goedel94.15a; << tools.m

        :Package Title: goedel94.15a      2007 June 15 at 8:15 a.m.

        It is now:  2007 Jun 16 at 14:27

        Loading Simplification Rules

        TOOLS.M                                      Revised 2007 June 10

        weightlimit = 40
```

summary

The predicates **monotone[w, x, y]** and **subtwine[w, x, y]** are defined as follows:

```
In[2]:= monotone[w, x, y]

Out[2]= subclass[composite[w, x, inverse[w]], y]

In[3]:= subtwine[w, x, y]

Out[3]= subclass[composite[w, x], composite[y, w]]
```

The class of all sets **w** for each of these conditions are expressible in terms of **cliques** and **transvar**, respectively:

```
In[4]:= class[w, monotone[w, x, y]]

Out[4]= cliques[complement[cross[x, complement[y]]]]

In[5]:= class[w, subtwine[w, x, y]]

Out[5]= transvar[cross[inverse[x], Id], cross[Id, y]]
```

In this notebook, the relationship between these conditions is studied. In general, monotonicity condition and the subtwining condition are not equivalent, but they can become equivalent in the presence of additional conditions.

the case x = y = Id

For the special case **x = y = Id**, the monotonicity condition requires **w** to be singlevalued, while the subtwining condition puts no condition whatsoever on **w**.

```
In[6]:= monotone[w, Id, Id]

Out[6]= FUNCTION[composite[Id, w]]

In[7]:= subtwine[w, Id, Id]

Out[7]= True
```

Obviously, these conditions are equivalent if one requires **w** to be a function. It is this observation that will be generalized in this notebook.

---

subtwine => monotone

In general, the **subtwine** condition implies the **monotone** condition for functions:

```
In[8]:= implies[and[FUNCTION[w], subtwine[w, x, y]], monotone[w, x, y]]

Out[8]= True
```

Theorem. (Variable free restatement of the above observation.)

```
In[9]:= Map[equal[V, #] &, dif[intersection[FUNS, transvar[cross[inverse[x], Id], cross[Id, y]]],
          cliques[complement[cross[x, complement[y]]]]] // complement // Normality]

Out[9]= subclass[intersection[FUNS, transvar[cross[inverse[x], Id], cross[Id, y]]],
         cliques[complement[cross[x, complement[y]]]]] == True

In[10]:= subclass[intersection[FUNS, transvar[cross[inverse[x_], Id], cross[Id, y_]]],
           cliques[complement[cross[x_, complement[y_]]]]] := True
```

Corollary. (One can obviously replace **FUNS** with any **map[u, v]** class. This will be needed later.)

```
In[11]:= SubstTest[implies, and[subclass[r, s], subclass[s, t]], subclass[r, t],
           {r → map[u, v], s → FUNS, t → union[cliques[complement[cross[x, complement[y]]]],
              complement[transvar[cross[inverse[x], Id], cross[Id, y]]]]}] // Reverse

Out[11]= subclass[intersection[map[u, v], transvar[cross[inverse[x], Id], cross[Id, y]]],
           cliques[complement[cross[x, complement[y]]]]] == True

In[12]:= subclass[intersection[map[u_, v_], transvar[cross[inverse[x_], Id], cross[Id, y_]]],
           cliques[complement[cross[x_, complement[y_]]]]] := True
```

---

domains

Lemma.

```
In[13]:= SubstTest[subclass, transvar[u, v], transvar[composite[t, u], composite[t, v]],
           {t → FIRST, u → cross[x, Id], v → cross[Id, y]}] // Reverse

Out[13]= subclass[transvar[cross[x, Id], cross[Id, y]],
           transvar[composite[x, FIRST], composite[FIRST, id[cart[V, domain[y]]]]]] == True

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

Theorem.

```
In[15]:= transvar[composite[x, FIRST], FIRST] // Normality

Out[15]= transvar[composite[x, FIRST], FIRST] == image[inverse[IMAGE[FIRST]], invar[x]]

In[16]:= transvar[composite[x_, FIRST], FIRST] := image[inverse[IMAGE[FIRST]], invar[x]]
```

Lemma.

```
In[17]:= SubstTest[implies, subclass[u, v], subclass[transvar[t, u], transvar[t, v]],
           {t -> composite[x, FIRST], u -> composite[FIRST, id[w]], v → FIRST}] // Reverse

Out[17]= subclass[image[IMAGE[FIRST],
            transvar[composite[x, FIRST], composite[FIRST, id[w]]]], invar[x]] == True

In[18]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

Theorem.

```
In[19]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
           {u -> transvar[cross[x, Id], cross[Id, y]],
            v -> transvar[composite[x, FIRST], composite[FIRST, id[cart[V, domain[y]]]]],
            w → image[inverse[IMAGE[FIRST]], invar[x]]}] // Reverse

Out[19]= subclass[image[IMAGE[FIRST], transvar[cross[x, Id], cross[Id, y]]], invar[x]] == True

In[20]:= subclass[image[IMAGE[FIRST], transvar[cross[x_, Id], cross[Id, y_]]], invar[x_]] := True
```

subcommutant[Di]

Theorem.

```
In[21]:= transvar[cross[Id, Di], cross[Di, Id]] // Normality

Out[21]= transvar[cross[Id, Di], cross[Di, Id]] == subcommutant[Di]

In[22]:= transvar[cross[Id, Di], cross[Di, Id]] := subcommutant[Di]
```

Theorem.

```
In[23]:= transvar[composite[x, SECOND], SECOND] // Normality

Out[23]= transvar[composite[x, SECOND], SECOND] == image[inverse[IMAGE[SECOND]], invar[x]]
```

```
In[24]:= transvar[composite[x_, SECOND], SECOND] := image[inverse[IMAGE[SECOND]], invar[x]]
```

Lemma.

```
In[25]:= SubstTest[subclass, transvar[u, v], transvar[composite[t, u], composite[t, v]],
           {t → SECOND, u → cross[Id, Di], v → cross[Di, Id]}] // Reverse

Out[25]= equal[0, domain[U[subcommutant[Di]]]] == True

In[26]:= domain[U[subcommutant[Di]]] := 0
```

Lemma.

```
In[27]:= SubstTest[composite, u, id[domain[u]], u → U[subcommutant[Di]]]

Out[27]= composite[Id, U[subcommutant[Di]]] == 0

In[28]:= % /. Equal → SetDelayed
```

Theorem.

```
In[29]:= equal[subcommutant[Di], P[complement[cart[V, V]]]] // AssertTest

Out[29]= equal[P[complement[cart[V, V]]], subcommutant[Di]] == True

In[30]:= subcommutant[Di] := P[complement[cart[V, V]]]
```

Lemma.

```
In[37]:= SubstTest[subclass, union[x, y], x, y -> image[Di, x]]

Out[37]= subclass[image[Di, x], x] == or[equal[0, x], equal[V, x]]

In[38]:= subclass[image[Di, x_], x_] := or[equal[0, x], equal[V, x]]
```

Theorem.

```
In[40]:= SubstTest[implies, subclass[u, v], subclass[domain[u], domain[v]],
           {u → composite[x, Di], v → composite[Di, x]}] // Reverse

Out[40]= or[equal[0, domain[x]], equal[V, domain[x]],
           not[subclass[composite[x, Di], composite[Di, x]]]] == True

In[41]:= or[equal[0, domain[x_]], equal[V, domain[x_]],
           not[subclass[composite[x_, Di], composite[Di, x_]]]] := True
```

the case x = y = Di

For the special case **x = y = Di**, the monotonicity condition is equivalent to the statement that the inverse of **w** is a function.

```
In[31]:= monotone[w, Di, Di]

Out[31]= FUNCTION[inverse[w]]
```

The subtwining condition requires that **inverse[w]** subcommute with **Di**, which

```
In[32]:= class[w, subtwine[w, Di, Di]]

Out[32]= P[complement[cart[V, V]]]
```

In partiular, the only small function that subcommutes with **Di** is **0**.

```
In[42]:= intersection[FUNS, subcommutant[Di]]

Out[42]= set[0]
```

For example, the only identity functions that subcommute with **Di** are **0** and **Id**.

```
In[43]:= subtwine[id[x], Di, Di]

Out[43]= or[equal[0, x], equal[V, x]]
```

---

monotone => subtwine

If one adds an additional condition, monotonicity implies the subtwining condition. (One need not require that **w** be a function for this.)

Lemma.

```
In[44]:= SubstTest[implies, subclass[u, v], subclass[composite[t, u], composite[t, v]],
           {t -> composite[w, x], u → id[domain[x]], v → composite[inverse[w], w]}] // Reverse

Out[44]= or[not[subclass[domain[x], domain[w]]],
           subclass[composite[w, x], composite[w, x, inverse[w], w]]] == True

In[45]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

Theorem

```
In[46]:= Map[not, SubstTest[and, implies[p1, p4], implies[p2, p3],
            implies[and[p3, p4], p5], not[implies[and[p1, p2], p5]],
            {p1 -> monotone[w, x, y], p2 -> subclass[domain[x], domain[w]],
             p3 → subclass[composite[w, x], composite[w, x, inverse[w], w]],
             p4 → subclass[composite[w, x, inverse[w], w], composite[y, w]],
             p5 → subtwine[w, x, y]}]] // Reverse

Out[46]= or[not[subclass[composite[w, x, inverse[w]], y]], not[subclass[domain[x], domain[w]]],
           subclass[composite[w, x], composite[y, w]]] == True
```

```
In[47]:= or[not[subclass[composite[w_, x_, inverse[w_]], y_]],
           not[subclass[domain[x_], domain[w_]]],
           subclass[composite[w_, x_], composite[y_, w_]]] := True
```

Theorem. (Variable-free restatement.)

```
In[48]:= Map[equal[V, #] &, SubstTest[class, z,
           implies[and[subclass[domain[x], domain[z]], member[z, u]], member[z, v]],
           {u -> cliques[complement[cross[x, complement[y]]]],
            v → transvar[cross[inverse[x], Id], cross[Id, y]]}]]
```

```
Out[48]= subclass[intersection[cliques[complement[cross[x, complement[y]]]],
            image[inverse[IMAGE[FIRST]], image[S, set[domain[x]]]]],
           transvar[cross[inverse[x], Id], cross[Id, y]]] == True
```

```
In[49]:= subclass[intersection[cliques[complement[cross[x_, complement[y_]]]],
            image[inverse[IMAGE[FIRST]], image[S, set[domain[x_]]]]],
           transvar[cross[inverse[x_], Id], cross[Id, y_]]] := True
```

Corollary.

```
In[50]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
           {u → map[domain[x], z], v -> image[inverse[IMAGE[FIRST]], image[S, set[domain[x]]]],
            w → union[transvar[cross[inverse[x], Id], cross[Id, y]],
              complement[cliques[complement[cross[x, complement[y]]]]]]}] // Reverse
```

```
Out[50]= subclass[intersection[cliques[complement[cross[x, complement[y]]]],
            map[domain[x], z]], transvar[cross[inverse[x], Id], cross[Id, y]]] == True
```

```
In[51]:= subclass[intersection[cliques[complement[cross[x_, complement[y_]]]],
            map[domain[x_], z_]], transvar[cross[inverse[x_], Id], cross[Id, y_]]] := True
```

main theorem

Combining the two inclusions derived above, one obtains an equation:

```
In[52]:= SubstTest[and, subclass[u, v], subclass[v, u],
           {u -> intersection[cliques[complement[cross[x, complement[y]]]], map[domain[x], z]],
            v -> intersection[transvar[cross[inverse[x], Id], cross[Id, y]], map[domain[x], z]]}]
```

```
Out[52]= equal[intersection[cliques[complement[cross[x, complement[y]]]], map[domain[x], z]],
            intersection[map[domain[x], z], transvar[cross[inverse[x], Id], cross[Id, y]]]] == True
```

```
In[53]:= intersection[map[domain[x_], z_], transvar[cross[inverse[x_], Id], cross[Id, y_]]] :=
           intersection[cliques[complement[cross[x, complement[y]]]], map[domain[x], z]]
```