monotone[x, y]

Johan G. F. Belinfante
2010 August 11

```
In[1]:= SetDirectory["l:"]; << goedel.10aug10a;<< tools.m

        :Package Title: goedel.10aug10a           2010 August 10 at 7:25 a.m.

        It is now:  2010 Aug 12 at 9:1

        Loading Simplification Rules

        TOOLS.M                                   Revised 2010 July 26

        weightlimit = 40
```

summary

Several variable-free statements concerning monotone functions are available in the **GOEDEL** program, but in their present form they suffer from the defect that it is not readily recognizable that they are even about monotonicity due to the occurrence of certain **cliques** constructions. To render these statements more perspicuous, a new binary constructor **monotone[x, y]** is introduced in this notebook, and some of its basic properties are derived. One of the more elegant applications derived below is an improved variable-free formulation of Zermelo's fixed point theorem, replacing a less transparent rule obtained 2006 June 10.

definition

The binary constructor **monotone[x, y]** is defined by the following membership rule.

```
In[2]:= member[t_, monotone[x_, y_]] :=
          and[member[t, V], subclass[t, cart[V, V]], subclass[composite[t, x, inverse[t]], y]]
```

The condition that **t** be a relation in this definition is included just for convenience, and does not severely limit the applicability of this constructor to monotone relations. If for some reason one does not wish to assume that **t** be a relation, one can instead make use of the class **image[inverse[IMAGE[id[cart[V, V]]]], monotone[x,y]]** of all sets **t** which satisfy **t ∘ x ∘ inverse[t] ⊂ y**. In the more likely case that one wants to further restrict attention to functions, one may sometimes need to intersect **monotone[x, y]** with some appropriate class of functions, such as **FUNS** or **map[u, v]**.

normalization

The normalization rule for **monotone[x, y]** involves the unary constructor **cliques[z]**. A rather general version of this rule is given in the following theorem which in most cases will suffice to automatically replace certain expressions involving the **cliques** constructor with new ones containing the new **monotone** constructor for statements about monotonicity.

Theorem. A general normalization rule.

```
In[3]:= image[inverse[IMAGE[id[cart[V, V]]]], monotone[x, complement[y]]] // Normality // Reverse

Out[3]= cliques[complement[cross[x, y]]] ==
         image[inverse[IMAGE[id[cart[V, V]]]], monotone[x, complement[y]]]

In[4]:= cliques[complement[cross[x_, y_]]] :=
         image[inverse[IMAGE[id[cart[V, V]]]], monotone[x, complement[y]]]
```

To obtain simple expressions, this general rule often needs to be supplemented with the following more special rule.

Theorem. A useful simplification rule.

```
In[5]:= monotone[x, y] // Normality // Reverse

Out[5]= intersection[monotone[x, y], P[cart[V, V]]] == monotone[x, y]

In[6]:= intersection[monotone[x_, y_], P[cart[V, V]]] := monotone[x, y]
```

Existing variable-free theorems about monotonicity are transformed by these normalization rules, and will therefore need to be replaced with new ones. Only a few such cases will be considered in this notebook. The rest will need to be taken care of another time.

some basic properties of monotone[x, y]

Many properties of **monotone[x, y]** can be derived from available properties of **cliques[z]**.

Theorem. The constructor **monotone[x, y]** only depends on the relational part of **x**.

```
In[7]:= SubstTest[cliques, intersection[complement[cross[t, complement[y]]],
          complement[cart[V, complement[cart[V, V]]]]], t → composite[Id, x]]

Out[7]= monotone[composite[Id, x], y] == monotone[x, y]

In[8]:= monotone[composite[Id, x_], y_] := monotone[x, y]
```

Theorem. The constructor **monotone[x, y]** only depends on the relational part of **y**.

```
In[9]:= SubstTest[cliques, intersection[complement[cross[x, complement[t]]],
          complement[cart[V, complement[cart[V, V]]]]], t → composite[Id, y]]

Out[9]= monotone[x, composite[Id, y]] == monotone[x, y]

In[10]:= monotone[x_, composite[Id, y_]] := monotone[x, y]
```

Theorem. Every subset of a monotone relation is monotone.

```
In[11]:= SubstTest[image, inverse[S], cliques[z],
           z -> intersection[complement[cross[x, complement[y]]],
             complement[cart[V, complement[cart[V, V]]]]]] // Reverse

Out[11]= image[inverse[S], monotone[x, y]] == monotone[x, y]

In[12]:= image[inverse[S], monotone[x_, y_]] := monotone[x, y]
```

The following simplification rule will be needed later.

Corollary.

```
In[13]:= SubstTest[subclass, image[inverse[S], x],
           image[inverse[S], t], t → monotone[y, z]] // Reverse

Out[13]= subclass[image[inverse[S], x], monotone[y, z]] == subclass[x, monotone[y, z]]

In[14]:= subclass[image[inverse[S], x_], monotone[y_, z_]] := subclass[x, monotone[y, z]]
```

The usefulness of the constructor **monotone[x, y]** is not limited to relations that are sets. One often has to deal with monotone functions such as **CORE[x]** and **IMAGE[x]** that are proper classes. Such functions are not elements of **monotone[x, y]** but are related to the **monotone** constructor in the following way.

Theorem. A relation is monotone if all its subsets are monotone.

```
In[15]:= SubstTest[subclass, P[t], cliques[z],
           z -> intersection[complement[cross[x, complement[y]]],
             complement[cart[V, complement[cart[V, V]]]]]] // Reverse

Out[15]= subclass[P[t], monotone[x, y]] ==
          and[subclass[t, cart[V, V]], subclass[composite[t, x, inverse[t]], y]]

In[16]:= subclass[P[t_], monotone[x_, y_]] :=
          and[subclass[t, cart[V, V]], subclass[composite[t, x, inverse[t]], y]]
```

non-sethood property for monotone[x, y]

The class **monotone[x, y]** is either the singleton of the empty set or a proper class.

Lemma. An explicit formula for the sum class of **monotone[x, y]**.

```
In[17]:= SubstTest[U, cliques[z], z -> intersection[complement[cross[x, complement[y]]],
             complement[cart[V, complement[cart[V, V]]]]]] // Reverse
```

```
Out[17]= U[monotone[x, y]] == union[cart[V, fix[y]], cart[complement[fix[x]], V]]
```

```
In[18]:= U[monotone[x_, y_]] := union[cart[V, fix[y]], cart[complement[fix[x]], V]]
```

Corollary. A necessary and sufficient condition for **monotone[x, y]** to be a set.

```
In[19]:= SubstTest[member, U[t], V, t → monotone[x, y]]
```

```
Out[19]= member[monotone[x, y], V] == and[equal[0, fix[y]], equal[V, fix[x]]]
```

```
In[20]:= member[monotone[x_, y_], V] := and[equal[0, fix[y]], equal[V, fix[x]]]
```

It will now be shown that this condition is equivalent to the statement that **monotone[x, y]** is the singleton of **0**.

Lemma.

```
In[21]:= SubstTest[empty, U[t], t → monotone[x, y]]
```

```
Out[21]= subclass[monotone[x, y], set[0]] == and[equal[0, fix[y]], equal[V, fix[x]]]
```

```
In[22]:= subclass[monotone[x_, y_], set[0]] := and[equal[0, fix[y]], equal[V, fix[x]]]
```

Theorem.

```
In[23]:= SubstTest[and, member[0, t], subclass[t, set[0]], t → monotone[x, y]]
```

```
Out[23]= equal[monotone[x, y], set[0]] == and[equal[0, fix[y]], equal[V, fix[x]]]
```

```
In[24]:= equal[monotone[x_, y_], set[0]] := and[equal[0, fix[y]], equal[V, fix[x]]]
```

An example:

```
In[25]:= monotone[Id, Di] // Normality
```

```
Out[25]= monotone[Id, Di] == set[0]
```

```
In[26]:= monotone[Id, Di] := set[0]
```

some special cases

Theorem. The class **FUNS** of all functions.

```
In[27]:= SubstTest[cliques, intersection[complement[cross[x, complement[y]]],
           complement[cart[V, complement[cart[V, V]]]]], {x → Id, y → Id}]
```

```
Out[27]= monotone[Id, Id] == FUNS
```

```
In[28]:= monotone[Id, Id] := FUNS
```

Theorem. The class of inverses of functions.

```
In[29]:= monotone[Di, Di] // Normality

Out[29]= monotone[Di, Di] == image[INVERSE, FUNS]

In[30]:= monotone[Di, Di] := image[INVERSE, FUNS]
```

Theorem. An extreme case.

```
In[31]:= monotone[0, x] // Normality

Out[31]= monotone[0, x] == P[cart[V, V]]

In[32]:= monotone[0, x_] := P[cart[V, V]]
```

Theorem. Another extreme case.

```
In[33]:= monotone[x, V] // Normality

Out[33]= monotone[x, V] == P[cart[V, V]]

In[34]:= monotone[x_, V] := P[cart[V, V]]
```

monotonicity properties

In this section it is shown that the binary constructor **monotone[x, y]** is antitone with respect to its first argument and monotone with respect to its second argument.

Theorem. The **monotone** constructor transforms unions to intersections with respect to its first argument.

```
In[35]:= SubstTest[cliques, intersection[complement[cross[t, complement[z]]],
           complement[cart[V, complement[cart[V, V]]]]], t → union[x, y]]

Out[35]= monotone[union[x, y], z] == intersection[monotone[x, z], monotone[y, z]]

In[36]:= monotone[union[x_, y_], z_] := intersection[monotone[x, z], monotone[y, z]]
```

Theorem. The constructor **monotone[x, y]** is antitone with respect to its first argument.

```
In[37]:= SubstTest[implies, equal[y, union[t, x]],
           subclass[monotone[y, z], monotone[x, z]], t → y] // Reverse

Out[37]= or[not[subclass[x, y]], subclass[monotone[y, z], monotone[x, z]]] == True

In[38]:= or[not[subclass[x_, y_]], subclass[monotone[y_, z_], monotone[x_, z_]]] := True
```

Theorem. The **monotone** constructor preserves intersections with respect to its second argument.

```
In[39]:= SubstTest[cliques, intersection[complement[cross[x, complement[t]]],
            complement[cart[V, complement[cart[V, V]]]]], t → intersection[y, z]] // Reverse

Out[39]= intersection[monotone[x, y], monotone[x, z]] == monotone[x, intersection[y, z]]

In[40]:= intersection[monotone[x_, y_], monotone[x_, z_]] := monotone[x, intersection[y, z]]
```

Corollary.

```
In[41]:= SubstTest[subclass, intersection[u, v],
           u, {u -> monotone[x, y], v -> monotone[x, z]}] // Reverse

Out[41]= subclass[monotone[x, intersection[y, z]], monotone[x, y]] == True

In[42]:= subclass[monotone[x_, intersection[y_, z_]], monotone[x_, y_]] := True
```

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

```
In[43]:= SubstTest[implies, equal[y, intersection[t, z]],
           subclass[monotone[x, y], monotone[x, z]], t → y] // Reverse

Out[43]= or[not[subclass[y, z]], subclass[monotone[x, y], monotone[x, z]]] == True

In[44]:= or[not[subclass[y_, z_]], subclass[monotone[x_, y_], monotone[x_, z_]]] := True
```

inverse rules

Since **u ⊂ v** implies that **inverse[u] ⊂ inverse[v]**, it follows that any monotonicity statement **t ∘ x ∘ inverse[t] ⊂ y** implies another monotonicity statement, **t ∘ inverse[x] ∘ inverse[t] ⊂ inverse[y]**. One take advantage of this fact to eliminate any occurrence of **inverse** from the second argument of **monotone** by moving it to the first argument.

Theorem. Transfer of **inverse** from the second argument to the first argument.

```
In[45]:= SubstTest[cliques, intersection[complement[cross[x, complement[t]]],
           complement[cart[V, complement[cart[V, V]]]]], t → inverse[y]]

Out[45]= monotone[x, inverse[y]] == monotone[inverse[x], y]

In[46]:= monotone[x_, inverse[y_]] := monotone[inverse[x], y]
```

Theorem. A similar rule needed when **complement** occurs.

```
In[47]:= SubstTest[cliques, intersection[complement[cross[x, complement[t]]],
           complement[cart[V, complement[cart[V, V]]]]], t → complement[inverse[y]]]

Out[47]= monotone[x, complement[inverse[y]]] == monotone[inverse[x], complement[y]]

In[48]:= monotone[x_, complement[inverse[y_]]] := monotone[inverse[x], complement[y]]
```

the class monotone[S, S]

For the named functions in the **GOEDEL** program, the most common form of monotonicity is of the form **t ∘ S ∘ inverse[t] ⊂ S**. In this section, some results about such functions are derived.

Lemma. Another simplification rule involving the class of all functions.

```
In[49]:= SubstTest[intersection, monotone[x, y],
           monotone[x, z], {x → Id, y → S, z → inverse[S]}] // Reverse

Out[49]= monotone[Id, S] == FUNS

In[50]:= monotone[Id, S] := FUNS
```

Theorem. A simplification rule.

```
In[51]:= SubstTest[monotone, union[x, y], z, {x → Id, y → S, z → S}]

Out[51]= intersection[FUNS, monotone[S, S]] == monotone[S, S]

In[52]:= intersection[FUNS, monotone[S, S]] := monotone[S, S]
```

Corollary. Monotone relations are functions.

```
In[53]:= SubstTest[subclass, intersection[u, v], u, {u → FUNS, v → monotone[S, S]}] // Reverse

Out[53]= subclass[monotone[S, S], FUNS] == True

In[54]:= subclass[monotone[S, S], FUNS] := True
```

The monotonicity property of **HULL[x]** functions can be formulated as follows:

```
In[55]:= subclass[P[HULL[x]], monotone[S, S]]

Out[55]= True
```

A variable-free restatement of this property can be derived by using **reify** to eliminate the variable.

Theorem.

```
In[56]:= Map[empty[composite[Id, complement[#]]] &,
          SubstTest[reify, x, complement[dif[P[HULL[x]], t]], t -> monotone[S, S]]]

Out[56]= subclass[range[LAMBHULL], monotone[S, S]] == True

In[57]:= subclass[range[LAMBHULL], monotone[S, S]] := True
```

Theorem. A succinct statement of Zermelo's fixed point theorem.

```
In[58]:= Map[equal[V, #] &, SubstTest[class, t, member[setpart[t], u], u ->
            union[complement[map[P[x], P[x]]], complement[monotone[S, S]], complement[P[Di]]]]]

Out[58]= equal[0, intersection[map[P[x], P[x]], monotone[S, S], P[Di]]] == True

In[59]:= intersection[map[P[x_], P[x_]], monotone[S, S], P[Di]] := 0
```

Corollary. A variable-free statement of Zermelo's fixed point theorem. There are no monotone fixed-point-free unary operations on any power set.

```
In[60]:= Map[U[range[VERTSECT[#]]] &, SubstTest[reify, x,
           intersection[map[P[x], P[x]], t], t → intersection[monotone[S, S], P[Di]]]]

Out[60]= intersection[UNOPS,
           image[inverse[IMAGE[FIRST]], range[POWER]], monotone[S, S], P[Di]] == 0

In[61]:= intersection[UNOPS,
           image[inverse[IMAGE[FIRST]], range[POWER]], monotone[S, S], P[Di]] := 0
```

antitone functions

The relative complement function **RC[x]** is usually described as being antitone or order-reversing. Formally:

```
In[62]:= subclass[composite[RC[x], inverse[S], RC[x]], S]

Out[62]= True
```

Theorem. A simplification rule.

```
In[63]:= SubstTest[monotone, union[x, y], z, {x → Id, y → inverse[S], z → S}]

Out[63]= intersection[FUNS, monotone[inverse[S], S]] == monotone[inverse[S], S]

In[64]:= intersection[FUNS, monotone[inverse[S], S]] := monotone[inverse[S], S]
```

Corollary. Antitone relations are functions.

```
In[65]:= SubstTest[subclass, intersection[u, v],
           u, {u → FUNS, v → monotone[inverse[S], S]}] // Reverse

Out[65]= subclass[monotone[inverse[S], S], FUNS] == True

In[66]:= subclass[monotone[inverse[S], S], FUNS] := True
```

A variable-free statement that all **RC[x]** functions are antitone can be derived using **reify** to eliminate the variable.

Theorem. All relative complement functions are antitone.

```
In[67]:= Map[empty[composite[Id, complement[#]]] &,
          SubstTest[reify, x, complement[dif[P[RC[x]], t]], t -> monotone[inverse[S], S]]]

Out[67]= subclass[range[RCF], monotone[inverse[S], S]] == True

In[68]:= subclass[range[RCF], monotone[inverse[S], S]] := True
```

closure properties

Since **cliques[x]** classes are closed under arbitrary intersections and under unions of chains, the same holds for **monotone[x, y]**. No rewrite rule is needed for the **Aclosure** result because this fact follows from the fact that any subset of a monotone relation is monotone.

```
In[69]:= Aclosure[monotone[x, y]]

Out[69]= monotone[x, y]
```

The following related result also follows automatically, and requires no separate new rewrite rule:

```
In[70]:= HULL[monotone[x, y]]

Out[70]= id[monotone[x, y]]
```

Theorem. The union of a chain of monotone relations is monotone.

```
In[71]:= SubstTest[Uchains, cliques[z], z -> intersection[complement[cross[x, complement[y]]],
              complement[cart[V, complement[cart[V, V]]]]]] // Reverse

Out[71]= Uchains[monotone[x, y]] == monotone[x, y]

In[72]:= Uchains[monotone[x_, y_]] := monotone[x, y]
```

The class **monotone[x, y]** is generally not closed under arbitrary unions. Again, no separate rewrite rule is needed for its **Uclosure**.

```
In[73]:= Uclosure[monotone[x, y]]

Out[73]= P[union[cart[V, fix[y]], cart[complement[fix[x]], V]]]
```