strong forms of monotonicity

Johan G. F. Belinfante and Tiffany Goble
2003 October 21

```
In[1]:= << goedel52.t05; << tools.m

        :Package Title: goedel52.t05      2003 October 16 at 1:05 p.m.

        It is now:  2003 Oct 22 at 9:8

        Loading Simplification Rules

        TOOLS.M                       Revised 2003 September 29

        weightlimit = 40
```

summary

Some new theorems and examples of monotone functions are derived in this notebook. Most of the examples considered in this notebook are monotone functions with hereditary domains. In fact, many of them are total. For such functions there are various corollaries of monotonicity, some of which are already in the **GOEDEL** program. For most of the examples, it is convenient to derive the strongest inclusions first and to obtain the weaker ones as corollaries.

temporary definitions

The derivations to be presented are easier to explain using the following (temporary) terminology.

```
In[2]:= total[x_] := equal[domain[x], V]
```

```
In[3]:= hereditary[x_] := equal[image[inverse[S], x], x]
```

```
In[4]:= monotone[x_] := subclass[composite[x, S, inverse[x]], S]
```

The reverse inclusion is:

```
In[5]:= reversemonotone[x_] := subclass[S, composite[x, S, inverse[x]]]
```

For future reference we note that the class of all sets satisfying the **monotone** predicate is:

```
In[6]:= class[x, monotone[x]]

Out[6]= cliques[complement[cross[S, complement[S]]]]
```

known results

The following functions are already recognized by the **GOEDEL** program to be monotone:

```
In[7]:= Select[Funs[Union[NamedClasses, Map[#[x] &, UnaryFunctors]]], monotone] // Union

Out[7]= {0, BIGCUP, Id, INVERSE, POWER, CORE[x],
         id[x], IMAGE[DUP], IMAGE[x], IMAGE[inverse[DUP]]}
```

Some other monotone functions are currently not recognized by the **GOEDEL** program, even when a stronger form of monotonicity is recognized:

```
In[8]:= Select[Funs[Union[NamedClasses, Map[#[x] &, UnaryFunctors]]],
          subcommute[#, S] &] // Union

Out[8]= {0, ACLOSURE, BIGCUP, Id, POWER, RANK, TC, ADJOIN[x],
         CORE[x], HULL[x], IMAGE[DUP], IMAGE[x], IMAGE[inverse[DUP]]}

In[9]:= Select[Funs[Union[NamedClasses, Map[#[x] &, UnaryFunctors]]],
          reversemonotone[inverse[#]] &] // Union

Out[9]= {Id, POWER, TC, IMAGE[DUP]}
```

The goal is to fill in some of the missing facts about monotonicity.

two old theorems and a useful fact

The meaning of monotonicity is explained by the following. This result is not limited to functions.

```
In[10]:= assert[forall[u, v, x, y, implies[and[member[pair[u, x], z],
               member[pair[v, y], z], subclass[u, v]], subclass[x, y]]] == monotone[z]

Out[10]= True
```

We recall two old theorems, which may help to explain some of the results derived in this notebook.

```
In[11]:= implies[subcommute[x, S], hereditary[domain[x]]]

Out[11]= True

In[12]:= implies[and[monotone[x], hereditary[domain[x]]], subcommute[x, S]]

Out[12]= True
```

The following fact is useful:

```
In[13]:= subclass[id[y], composite[inverse[x], x]]

Out[13]= subclass[y, domain[x]]
```

some new theorems

The new theorems derived in this section may also help to understand some of the examples in this notebook.

Theorem 1.

```
In[14]:= or[equal[x, image[inverse[S], x]], not[equal[V, x]]] // AssertTest

Out[14]= or[equal[x, image[inverse[S], x]], not[equal[V, x]]] == True

In[15]:= or[equal[x_, image[inverse[S], x_]], not[equal[V, x_]]] := True
```

Restatement:

```
In[16]:= implies[total[x], hereditary[domain[x]]]

Out[16]= True
```

Lemma.

```
In[17]:= (SubstTest[implies, subclass[u, v], subclass[domain[u], domain[v]],
            {u -> S, v -> composite[inverse[x], S, x]}] /. {x -> x_}) /. Equal -> SetDelayed
```

Theorem 2.

```
In[18]:= SubstTest[and, implies[p, subclass[u, v]], subclass[v, u],
           {p -> subclass[S, composite[inverse[x], S, x]], u -> V, v -> domain[x]}] // Reverse

Out[18]= or[equal[V, domain[x]], not[subclass[S, composite[inverse[x], S, x]]]] == True

In[19]:= or[equal[V, domain[x_]], not[subclass[S, composite[inverse[x_], S, x_]]]] := True
```

Restatements:

```
In[20]:= implies[subclass[S, composite[inverse[x], S, x]], total[x]]

Out[20]= True

In[21]:= implies[reversemonotone[inverse[x]], total[x]]

Out[21]= True
```

Lemma.

```
In[22]:= SubstTest[implies, subclass[u, v], subclass[composite[u, w], composite[v, w]],
           {u -> Id, v -> composite[inverse[x], x], w -> S}]

Out[22]= or[not[equal[V, domain[x]]], subclass[S, composite[inverse[x], x, S]]] == True

In[23]:= (% /. x -> x_) /. Equal -> SetDelayed
```

Theorem.

```
In[24]:= Map[not, SubstTest[and, implies[p2, p3], implies[p1, p4],
           implies[and[p3, p4], p5], not[implies[and[p1, p2], p5]],
           {p1 -> subcommute[x, S], p2 -> total[x], p3 -> subclass[S, composite[inverse[x], x, S]],
            p4 -> subclass[composite[inverse[x], x, S], composite[inverse[x], S, x]],
            p5 -> reversemonotone[inverse[x]]}]]

Out[24]= or[not[equal[V, domain[x]]], not[subclass[composite[x, S], composite[S, x]]],
           subclass[S, composite[inverse[x], S, x]]] == True

In[25]:= or[not[equal[V, domain[x_]]], not[subclass[composite[x_, S], composite[S, x_]]],
           subclass[S, composite[inverse[x_], S, x_]]] := True
```

Restatement:

```
In[26]:= implies[and[total[x], subcommute[x, S]], reversemonotone[inverse[x]]]

Out[26]= True
```

Corollary.

```
In[27]:= Map[not, SubstTest[and, implies[and[p1, p3], p4],
           implies[p2, p3], implies[and[p2, p4], p5], not[implies[and[p1, p2], p5]],
           {p1 -> monotone[x], p2 -> total[x], p3 -> hereditary[domain[x]],
            p4 -> subcommute[x, S], p5 -> reversemonotone[inverse[x]]}]]

Out[27]= or[not[equal[V, domain[x]]], not[subclass[composite[x, S, inverse[x]], S]],
           subclass[S, composite[inverse[x], S, x]]] == True

In[28]:= or[not[equal[V, domain[x_]]], not[subclass[composite[x_, S, inverse[x_]], S]],
           subclass[S, composite[inverse[x_], S, x_]]] := True
```

Restatement:

```
In[29]:= implies[and[monotone[x], total[x]], reversemonotone[inverse[x]]]

Out[29]= True
```

---

theorem: subcommute implies monotone for functions

Lemma 1.

```
In[30]:= SubstTest[implies, subclass[u, v],
          subclass[composite[S, u], composite[S, v]], {u -> composite[x, inverse[x]], v -> Id}]

Out[30]= or[not[FUNCTION[composite[Id, x]]], subclass[composite[x, inverse[x]], S]] == True

In[31]:= or[not[FUNCTION[composite[Id, x_]]], subclass[composite[x_, inverse[x_]], S]] := True
```

Lemma 2.

```
In[32]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
                 {p1 -> FUNCTION[x], p2 -> FUNCTION[composite[Id, x]],
            p3 -> subclass[composite[x, inverse[x]], S]}]]

Out[32]= or[not[FUNCTION[x]], subclass[composite[x, inverse[x]], S]] == True

In[33]:= or[not[FUNCTION[x]], subclass[composite[x, inverse[x]], S]] := True
```

Lemma 3.

```
In[34]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
          {u -> composite[x, S, inverse[x]], v -> composite[S, x, inverse[x]], w -> S}]

Out[34]= or[not[subclass[composite[x, inverse[x]], S]],
           not[subclass[composite[x, S, inverse[x]], composite[S, x, inverse[x]]]],
           subclass[composite[x, S, inverse[x]], S]] == True

In[35]:= (% /. x -> x_) /. Equal -> SetDelayed
```

Theorem.

```
In[36]:= Map[not, SubstTest[and, implies[p1, p3], implies[p2, p4], implies[and[p3, p4], p5],
           not[implies[and[p1, p2], p5]], {p1 -> subcommute[x, S], p2 -> FUNCTION[x],
            p3 -> subclass[composite[x, S, inverse[x]], composite[S, x, inverse[x]]],
            p4 -> subclass[composite[x, inverse[x]], S],
          p5 -> monotone[x]}]]

Out[36]= or[not[FUNCTION[x]], not[subclass[composite[x, S], composite[S, x]]],
           subclass[composite[x, S, inverse[x]], S]] == True

In[37]:= or[not[FUNCTION[x_]], not[subclass[composite[x_, S], composite[S, x_]]],
           subclass[composite[x_, S, inverse[x_]], S]] := True
```

Restatement:

```
In[38]:= implies[and[FUNCTION[x], subcommute[x, S]], monotone[x]]

Out[38]= True
```

theorem: reverse−monotone−inverse implies subcommute for functions

Lemma.

```
In[39]:= SubstTest[implies, subclass[u, v], subclass[composite[u, w], composite[v, w]],
             {u -> composite[x, inverse[x]], v -> S, w -> composite[S, x]}]

Out[39]= or[not[subclass[composite[x, inverse[x]], S]],
           subclass[composite[x, inverse[x], S, x], composite[S, x]]] == True

In[40]:= (% /. x -> x_) /. Equal -> SetDelayed
```

Theorem.

```
In[41]:= Map[not, SubstTest[and, implies[p1, p3], implies[p2, p4],
           implies[p4, p5], implies[and[p3, p5], p6], not[implies[and[p1, p2], p6]],
           {p1 -> subclass[S, composite[inverse[x], S, x]], p2 -> FUNCTION[x],
            p3 -> subclass[composite[x, S], composite[x, inverse[x], S, x]],
                 p4 -> subclass[composite[x, inverse[x]], S], p5 ->
              subclass[composite[x, inverse[x], S, x], composite[S, x]], p6 -> subcommute[x, S]}]]

Out[41]= or[not[FUNCTION[x]], not[subclass[S, composite[inverse[x], S, x]]],
           subclass[composite[x, S], composite[S, x]]] == True

In[42]:= or[not[FUNCTION[x_]], not[subclass[S, composite[inverse[x_], S, x_]]],
           subclass[composite[x_, S], composite[S, x_]]] := True
```

Restatement:

```
In[43]:= implies[and[FUNCTION[x], reversemonotone[inverse[x]]], subcommute[x, S]]

Out[43]= True
```

total + monotone

For the total functions, a common proof technique works. The following lemmas are required:

```
In[44]:= Map[Print[VSRenormality[dif[S, composite[inverse[#], S, #]]]] &,
           {ACLOSURE, BIGCUP, CLIQUES, CORE[x], HC, SUBVAR, UCLOSURE}];

         intersection[S, composite[inverse[ACLOSURE], complement[S], ACLOSURE]] == 0

         intersection[S, composite[inverse[BIGCUP], inverse[POWER], complement[S]]] == 0

         intersection[S, composite[inverse[CLIQUES], complement[S], CLIQUES]] == 0

         intersection[S, composite[inverse[CORE[x]], complement[S], CORE[x]]] == 0

         intersection[S, composite[complement[S], HC]] == 0

         intersection[S, composite[inverse[SUBVAR], complement[S], SUBVAR]] == 0

         intersection[S, composite[inverse[UCLOSURE], complement[S], UCLOSURE]] == 0
```

All these results are added as temporary rules.

```
In[45]:= intersection[S, composite[inverse[ACLOSURE], complement[S], ACLOSURE]] := 0

In[46]:= intersection[S, composite[inverse[BIGCUP], inverse[POWER], complement[S]]] := 0

In[47]:= intersection[S, composite[inverse[CLIQUES], complement[S], CLIQUES]] := 0

In[48]:= intersection[S, composite[inverse[CORE[x_]], complement[S], CORE[x_]]] := 0

In[49]:= intersection[S, composite[complement[S], HC]] := 0

In[50]:= intersection[S, composite[inverse[SUBVAR], complement[S], SUBVAR]] := 0

In[51]:= intersection[S, composite[inverse[UCLOSURE], complement[S], UCLOSURE]] := 0
```

Examples: some total monotone functions.

```
In[52]:= Map[Reverse[SubstTest[equal, 0, dif[u, v], {u -> S, v -> composite[inverse[#], S, #]}]] &,
           {ACLOSURE, BIGCUP, CLIQUES, CORE[x], HC, SUBVAR, UCLOSURE}] // TableForm

Out[52]//TableForm=
        subclass[S, composite[inverse[ACLOSURE], S, ACLOSURE]] == True
        subclass[S, composite[inverse[BIGCUP], inverse[POWER], S]] == True
        subclass[S, composite[inverse[CLIQUES], S, CLIQUES]] == True
        subclass[S, composite[inverse[CORE[x]], S, CORE[x]]] == True
        subclass[S, composite[S, HC]] == True
        subclass[S, composite[inverse[SUBVAR], S, SUBVAR]] == True
        subclass[S, composite[inverse[UCLOSURE], S, UCLOSURE]] == True
```

These are suitable to be made into permanent rules:

```
In[53]:= subclass[S, composite[inverse[ACLOSURE], S, ACLOSURE]] := True

In[54]:= subclass[S, composite[inverse[BIGCUP], inverse[POWER], S]] := True
```

```
In[55]:= subclass[S, composite[inverse[CLIQUES], S, CLIQUES]] := True

In[56]:= subclass[S, composite[inverse[CORE[x_]], S, CORE[x_]]] := True

In[57]:= subclass[S, composite[S, HC]] := True

In[58]:= subclass[S, composite[inverse[SUBVAR], S, SUBVAR]] := True

In[59]:= subclass[S, composite[inverse[UCLOSURE], S, UCLOSURE]] := True
```

ADJOIN[x]

The function **ADJOIN[x]** is total when **x** is a set.

```
In[60]:= domain[ADJOIN[x]]

Out[60]= image[V, singleton[x]]
```

Lemma.

```
In[61]:= dif[S, composite[S, IMAGE[id[x]]]] // VSNormality

Out[61]= intersection[S, composite[complement[S], IMAGE[id[x]]]] == 0

In[62]:= (% /. x -> x_) /. Equal -> SetDelayed
```

New permanent rule:

```
In[63]:= SubstTest[equal, 0, dif[u, v], {u -> S, v -> composite[S, IMAGE[id[x]]]}] // Reverse

Out[63]= subclass[S, composite[S, IMAGE[id[x]]]] == True

In[64]:= subclass[S, composite[S, IMAGE[id[x_]]]] := True
```

Corollary:

```
In[65]:= subclass[S, composite[inverse[ADJOIN[x]], S, ADJOIN[x]]]

Out[65]= member[x, V]
```

subcommute[x,S] corollaries

In a few cases, the subcommute corollary is already known.

```
In[66]:= Select[{ACLOSURE, BIGCUP, CLIQUES, CORE[x], HC, SUBVAR, UCLOSURE}, subcommute[#, S] &]

Out[66]= {ACLOSURE, BIGCUP, CORE[x]}
```

The case of **HC** follows from that of **CORE[x]**:

```
In[67]:= SubstTest[subcommute, CORE[x], S, x -> FULL]

Out[67]= subclass[composite[HC, S], composite[S, HC]] == True

In[68]:= subclass[composite[HC, S], composite[S, HC]] := True
```

In the remaining three cases, a theorem proved earlier suffices:

```
In[69]:= Map[SubstTest[implies, and[FUNCTION[x], subclass[S, composite[inverse[x], S, x]]],
             subcommute[x, S], x -> #] &, {CLIQUES, SUBVAR, UCLOSURE}] // TableForm

Out[69]//TableForm=
        subclass[composite[CLIQUES, S], composite[S, CLIQUES]] == True
        subclass[composite[SUBVAR, S], composite[S, SUBVAR]] == True
        subclass[composite[UCLOSURE, S], composite[S, UCLOSURE]] == True

In[70]:= subclass[composite[CLIQUES, S], composite[S, CLIQUES]] := True

In[71]:= subclass[composite[SUBVAR, S], composite[S, SUBVAR]] := True

In[72]:= subclass[composite[UCLOSURE, S], composite[S, UCLOSURE]] := True
```

monotone corollaries

The monotonicity result is already known in two cases:

```
In[73]:= Select[{ACLOSURE, BIGCUP, CLIQUES, CORE[x], HC, SUBVAR, TC, UCLOSURE}, monotone]

Out[73]= {BIGCUP, CORE[x]}
```

The case of **HC** is best done as a corollary of the **CORE[x]** formula.

```
In[74]:= SubstTest[monotone, CORE[x], x -> FULL]

Out[74]= subclass[composite[HC, S, inverse[HC]], S] == True

In[75]:= subclass[composite[HC, S, inverse[HC]], S] := True
```

The functions **HULL[x]** and **RANK** are not total, but they have hereditary domains, and the subcommute property is known for them.

```
In[76]:= Map[SubstTest[implies, and[FUNCTION[x], subcommute[x, S]], monotone[x], x -> #] &,
           {ACLOSURE, CLIQUES, HULL[x], RANK, SUBVAR, TC, UCLOSURE}] // TableForm

Out[76]//TableForm=
        subclass[composite[ACLOSURE, S, inverse[ACLOSURE]], S] == True
        subclass[composite[CLIQUES, S, inverse[CLIQUES]], S] == True
        subclass[composite[HULL[x], S, inverse[HULL[x]]], S] == True
        subclass[composite[RANK, S, inverse[RANK]], S] == True
        subclass[composite[SUBVAR, S, inverse[SUBVAR]], S] == True
        subclass[composite[id[FULL], S, inverse[TC]], S] == True
        subclass[composite[UCLOSURE, S, inverse[UCLOSURE]], S] == True

In[77]:= subclass[composite[ACLOSURE, S, inverse[ACLOSURE]], S] := True
```

```
In[78]:= subclass[composite[CLIQUES, S, inverse[CLIQUES]], S] := True

In[79]:= subclass[composite[HULL[x_], S, inverse[HULL[x_]]], S] := True

In[80]:= subclass[composite[RANK, S, inverse[RANK]], S] := True

In[81]:= subclass[composite[SUBVAR, S, inverse[SUBVAR]], S] := True

In[82]:= subclass[composite[id[FULL], S, inverse[TC]], S] := True

In[83]:= subclass[composite[UCLOSURE, S, inverse[UCLOSURE]], S] := True
```

monotonicity of plus[x]

Themonotonicity of the function **plus[x]** is established as follows:

```
In[84]:= Assoc[plus[x], composite[S, id[omega]], inverse[plus[x]]]

Out[84]= composite[plus[x], S, inverse[plus[x]]] ==
          composite[id[omega], S, id[intersection[omega,
             image[S, singleton[x]], image[V, intersection[omega, singleton[x]]]]]]

In[85]:= composite[plus[x_], S, inverse[plus[x_]]] :=
          composite[id[omega], S, id[intersection[omega,
             image[S, singleton[x]], image[V, intersection[omega, singleton[x]]]]]]
```

Corollary:

```
In[86]:= monotone[plus[x]]

Out[86]= True
```

The inverse of **plus[x]** is monotone as well. To derive this fact, one can proceed as follows. First, a lemma:

```
In[87]:= Assoc[id[image[S, singleton[x]]],
           id[image[V, intersection[omega, singleton[x]]]], plus[x]] // Reverse

Out[87]= composite[id[intersection[image[S, singleton[x]],
             image[V, intersection[omega, singleton[x]]]]], plus[x]] == plus[x]

In[88]:= composite[id[intersection[image[S, singleton[x_]],
             image[V, intersection[omega, singleton[x_]]]]], plus[x_]] := plus[x]
```

The main idea is to use **ImageComp**:

```
In[89]:= ImageComp[cross[inverse[plus[x]], inverse[plus[x]]],
           cross[plus[x], plus[x]], S] // Reverse

Out[89]= composite[inverse[plus[x]], S, plus[x]] == composite[
           id[intersection[omega, image[V, intersection[omega, singleton[x]]]]], S, id[omega]]

In[90]:= composite[inverse[plus[x_]], S, plus[x_]] := composite[
           id[intersection[omega, image[V, intersection[omega, singleton[x]]]]], S, id[omega]]
```

Corollary:

```
In[91]:= monotone[inverse[plus[x]]]

Out[91]= True
```

monotonicity of integers

The monotonicity for non−negative integers is established by:

```
In[92]:= Map[equal[0, #] &,
           composite[SWAP, RIF, id[composite[SWAP, cross[Id, complement[inverse[S]]]]], cross[Id,
             composite[SWAP, cross[S, Id]]], inverse[E], CART, DUP, PLUS] // RelnNormality]

Out[92]= subclass[range[PLUS], cliques[complement[cross[S, complement[S]]]]] == True

In[93]:= subclass[range[PLUS], cliques[complement[cross[S, complement[S]]]]] := True
```

For non−positive integers, a similar derivation works:

```
In[94]:= Map[equal[0, #] &,
           composite[SWAP, RIF, id[composite[SWAP, cross[Id, complement[inverse[S]]]]],
             cross[Id, composite[SWAP, cross[S, Id]]], inverse[E],
             CART, DUP, INVERSE, PLUS] // RelnNormality]

Out[94]= subclass[image[INVERSE, range[PLUS]],
           cliques[complement[cross[S, complement[S]]]]] == True

In[95]:= subclass[image[INVERSE, range[PLUS]],
           cliques[complement[cross[S, complement[S]]]]] := True
```

It remains to put these two results together to obtain a formula that says that all integers are monotone:

```
In[96]:= SubstTest[subclass, union[x, y], z, {x -> range[PLUS],
           y -> image[INVERSE, range[PLUS]], z -> cliques[complement[cross[S, complement[S]]]]}]

Out[96]= subclass[Z, cliques[complement[cross[S, complement[S]]]]] == True

In[97]:= subclass[Z, cliques[complement[cross[S, complement[S]]]]] := True
```

monotonicity of inverse functions

```
In[98]:= Select[Funs[NamedClasses], monotone[inverse[#]] &]

Out[98]= {0, Id, INVERSE, POWER, SINGLETON}
```

This is a degenerate form of monotonicity:

```
In[99]:= composite[inverse[SINGLETON], S, SINGLETON]

Out[99]= Id
```

inverse[RCF]

The monotonicity of **inverse[RCF]** is derived in this section. This is another degenerate form of monotonicity.

```
In[100]:=
Map[equal[V, class[x, #]] &, SubstTest[implies, subclass[u, v],
  subclass[image[u, w], image[v, w]], {u -> RC[x], v -> RC[y], w -> singleton[0]}]]
```

```
Out[100]=
subclass[image[inverse[RCF], P[RC[y]]], singleton[y]] == True
```

```
In[101]:=
subclass[image[inverse[RCF], P[RC[y_]]], singleton[y_]] := True
```

```
In[102]:=
SubstTest[and, subclass[u, v], subclass[v, u],
 {u -> singleton[x], v -> image[inverse[RCF], P[RC[x]]]}]
```

```
Out[102]=
True == equal[image[inverse[RCF], P[RC[x]]], singleton[x]]
```

```
In[103]:=
image[inverse[RCF], P[RC[x_]]] := singleton[x]
```

```
In[104]:=
Map[inverse, SubstTest[reify, x, image[inverse[RCF], P[F[x]]], F -> RC] // Reverse]
```

```
Out[104]=
composite[inverse[RCF], S, RCF] == Id
```

```
In[105]:=
composite[inverse[RCF], S, RCF] := Id
```

```
In[106]:=
monotone[inverse[RCF]]
```

```
Out[106]=
True
```