



---

## review of Gödel's class theory

In Gödel's class theory, every thing is a class. Sets are a special kind of class. A class  $x$  is a set if it belongs to some class.

```
In[2]:= assert[exists[y, member[x, y]]]
```

```
out[2]= member[x, V]
```

The primitives of Gödel's class theory are: the universal class  $V$ , the membership relation  $E$ , the class constructors **pairset**, **complement**, **intersection**, **domain**, **flip**, **rotate** and **cart**, the predicates **equal** and **member**, subject to various axioms, including the axiom of regularity and the axiom of choice. There is no general class constructor of the form **class[x, p[x]]**, but instead a general existence metatheorem is provided that permits one to construct expressions involving the primitives by syntactically analyzing the statement **p[x]**, which is presumed to be built up by logical connectives and quantifiers from the primitives. The **GOEDEL** program contains a set of rewrite rules for **class** that amount to a modification of Gödel's proof of his metatheorem. The main departure from Gödel's proof is that the Kuratowski construction of ordered pairs is not assumed. Instead, **pair** is adopted as an additional primitive, and his algorithm is modified by having statements about **pair** be reduced to ones involving **equal** rather than **member**.

---

## examples of class rules

```
In[3]:= Begin["Goedel`Private`"];
```

The following **class** rule converts **not** to **complement**. The expression **class[x,True]** is needed here because **x** need not be an atomic symbol, but could involve **pair**.

```
In[4]:= FirstMatch[class[x_, HoldPattern[not[p_]]]]
```

```
Out[4]= class[x_, not[p_]] := intersection[
    complement[class[x, p]], class[x, True]]
```

Another **class** rule converts **and** to **intersection**.

```
In[5]:= FirstMatch[class[x_, HoldPattern[and[p_, q_]]]]
```

```
Out[5]= class[x_, and[p_, q_]] := intersection[class[x, p], class[x, q]]
```

Quantifiers are eliminated by using the primitive **domain**:

```
In[6]:= FirstMatch[class[x_, HoldPattern[exists[y_, p_]]]]
```

```
Out[6]= class[x_, exists[y_, p_]] := domain[class[pair[x, y], p]]
```

---

## definitions

Classes are defined in the **GOEDEL** program, by specifying a membership rule. If the rule involves quantifiers, it is wrapped with **class**. For example, the rule that defines the intersection  $\mathbf{A}[\mathbf{x}]$  of a collection of sets is:

```
In[7]:= FirstMatch[class[x_, HoldPattern[member[y_, A[z_]]]]]]
```

```
Out[7]= class[w_, member[u_, A[x_]]] := Module[
  {v = Unique[]}, class[w, and[member[u, V],
  forall[v, implies[member[v, x], member[u, v]]]]]]]
```

The class  $\mathbf{A}[\mathbf{x}]$  is a set if  $\mathbf{x}$  is not empty. The intersection of the empty collection is  $\mathbf{V}$ . The corresponding function is consequently not total:

```
In[8]:= lambda[x, A[x]]
```

```
Out[8]= BIGCAP
```

```
In[9]:= domain[BIGCAP]
```

```
Out[9]= complement[singleton[0]]
```

---

## constructors are not functions

The constructor **A** should not be confused with the function **BIGCAP**. Functions are classes of ordered pairs, but constructors are not. The class **A[x]** gives the intersection of any class **x**, not just for sets. Applying **BIGCAP** only yields the intersection for sets:

```
In[10]:= equal[A[x], APPLY[BIGCAP, x]]
```

```
Out[10]= member[x, V]
```

The connection between the constructor and the corresponding function is:

```
In[11]:= class[pair[x, y], equal[y, A[x]]]
```

```
Out[11]= BIGCAP
```

Non-primitive predicates, such as **subclass**, whose definitions involve quantifiers are likewise wrapped with **class**.

```
In[12]:= FirstMatch[class[x_, HoldPattern[subclass[y_, z_]]]]
```

```
Out[12]= class[w_, subclass[x_, y_]] := Module[{u = Unique[]}, class[
  w, forall[u, implies[member[u, x], member[u, y]]]]]
```

Complicated concepts are often defined by rules wrapped with **class** just to avoid having their definitions expanded out unnecessarily.

---

## avoiding ordered pairs

To help avoid explicit mention of ordered pairs, the **GOEDEL** program provides the constructors **first** and **second** which permit one to define relations without **class** wrappers. The classes **first[x]** and **second[x]** are sets when **x** is an ordered pair. When **x** is not an ordered pair, both of these are equal to **V**.

```
In[13]:= A[domain[singleton[x]]]
```

```
Out[13]= first[x]
```

```
In[14]:= A[range[singleton[x]]]
```

```
Out[14]= second[x]
```

For example, the membership rule for cartesian products is:

```
In[15]:= member[x, cart[y, z]]
```

```
Out[15]= and[member[first[x], y], member[second[x], z]]
```

Additional examples are provided by the membership relation **E**, the subset relation **S** and the diversity relation **Di**:

```
In[16]:= member[x, E]
```

```
Out[16]= member[first[x], second[x]]
```

```
In[17]:= member[x, S]
```

```
Out[17]= and[member[first[x], V], subclass[first[x], second[x]]]
```

```
In[18]:= member[x, Di]
```

```
Out[18]= and[member[first[x], V], not[equal[first[x], second[x]]]]
```

---

## U[x] and P[x]

If a membership rule does not involve quantifiers, it need not be not wrapped with **class**. For example, the power class constructor **P[x]** is defined by:

```
In[19]:= member[x, P[y]]
Out[19]= and[member[x, V], subclass[x, y]]
```

The rule that defines the sum class constructor **U[x]** is wrapped:

```
In[20]:= FirstMatch[class[x_, HoldPattern[member[y_, U[z_]]]]]
Out[20]= class[w_, member[x_, U[z_]]] := Module[{y = Unique[]},
  class[w, exists[y, and[member[x, y], member[y, z]]]]]
```

These constructors are connected by the following basic rewrite rule:

```
In[21]:= subclass[x, P[y]]
Out[21]= subclass[U[x], y]
```

There are corresponding functions:

```
In[22]:= lambda[x, P[x]]
Out[22]= POWER

In[23]:= lambda[x, U[x]]
Out[23]= BIGCUP
```

---

## core[x,y] and CORE[x]

The class **core[x,y]** is the union of all subsets of  $y$  that belong to  $x$ .

```
In[24]:= class[u,
           exists[v, and[member[u, v], member[v, x], subclass[v, y]]]]
```

```
Out[24]= core[x, y]
```

The function **CORE[x]** is

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

```
Out[25]= CORE[x]
```

These are total functions:

```
In[26]:= domain[CORE[x]]
```

```
Out[26]= V
```

Using the **GOEDEL** program, it was discovered that the following properties characterize these functions:

```
In[27]:= implies[and[FUNCTION[x], idempotent[x],
                    equal[domain[x], V], subclass[x, inverse[S]],
                    subcommute[x, S]], equal[x, CORE[fix[x]]]]
```

```
Out[27]= True
```

The meanings of **idempotent** and **subcommute** are

```
In[28]:= idempotent[x]
```

```
Out[28]= equal[x, composite[x, x]]
```

```
In[29]:= subcommute[x, y]
```

```
Out[29]= subclass[composite[x, y], composite[y, x]]
```

For a total function, the latter statement is equivalent to monotonicity.

---

## Uclosure and Aclosure

Since the **CORE[x]** functions are idempotent, their ranges coincide with their fixed point sets:

```
In[30]:= implies[and[idempotent[x], FUNCTION[x]], equal[range[x], fix[x]]]
```

```
Out[30]= True
```

The range of **CORE[x]** is the class **Uclosure[x]** of all unions of subsets of **x**.

```
In[31]:= fix[CORE[x]]
```

```
Out[31]= Uclosure[x]
```

```
In[32]:= class[u, exists[s, and[equal[u, U[s]], subclass[s, x]]]]
```

```
Out[32]= Uclosure[x]
```

Similarly, **Aclosure[x]** is the class of all intersections of nonempty subsets of **x**.

```
In[33]:= class[u, exists[s, and[equal[u, A[s]], subclass[s, x]]]]
```

```
Out[33]= Aclosure[x]
```

The corresponding functions are:

```
In[34]:= lambda[x, Aclosure[x]]
```

```
Out[34]= ACLOSURE
```

```
In[35]:= lambda[x, Uclosure[x]]
```

```
Out[35]= UCLOSURE
```

---

## assert

Any statement in Gödel's class theory can be rewritten as an equation. In the process, all quantifiers in the original statement are eliminated. To do this, one uses **assert**, which is defined in terms of **class** as follows:

```
In[36]:= ?? assert
```

```
assert[p] is a statement equivalent to p
  obtained by applying Goedel's algorithm to class[w,p].
  Applying assert repeatedly sometimes simplifies a statement.

assert[p_] := Module[{w = Unique[]}, equal[V, class[w, p]]]
```

As an illustration, consider the statement that a collection of sets is closed under binary intersections:

```
In[37]:= assert[forall[u, v, implies[and[member[u, x], member[v, x]],
  member[intersection[u, v], x]]]]

Out[37]= subclass[image[CAP, cart[x, x]], x]
```

The function **CAP** corresponds to binary intersections:

```
In[38]:= lambda[pair[x, y], intersection[x, y]]

Out[38]= CAP
```

---

## topology

A **topology** is a set  $\mathbf{t}$  that is closed under binary intersections and arbitrary unions. The members of  $\mathbf{t}$  are called the **open** sets for that topology. The **topological space** for a topology  $\mathbf{t}$  is the union  $\mathbf{U}[\mathbf{t}]$  of all the open sets. The class **TOPS** of all topologies is

```
In[39]:= class[t,  
          and[equal[t, Uclosure[t]], subclass[image[CAP, cart[t, t]], t]]
```

```
Out[39]= TOPS
```

The **GOEDEL** program has been used to derive a number of theorems in topology, but this enterprise is still in its infancy, and so it is not hard to find simple facts in topology that have not yet been added to the program. This makes it rather easy to illustrate how the **GOEDEL** program can be used to discover and derive new facts in this area of mathematics.

---

## example: interiors are open

If  $\mathbf{t}$  is a topology and  $\mathbf{x}$  is a subset of  $\mathbf{U}[\mathbf{t}]$ , then  $\mathbf{core}[\mathbf{t},\mathbf{x}]$  is the **interior** of  $\mathbf{x}$ . It is an elementary fact in topology that the interior of any set is an open set. This theorem happens to be one that has not yet been added to the **GOEDEL** program. Deriving this theorem only requires very elementary reasoning, and can be done in a single step. The reasoning is done using **SubstTest**, which is the workhorse for derivations in the **GOEDEL** program::

```
In[40]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
    not[implies[p1, p3]], {p1 → member[t, TOPS],
    p2 → member[t, fix[UCLOSURE]], p3 → member[core[t, x], t}}]]
```

```
out[40]= or[member[core[t, x], t], not[member[t, TOPS]]] == True
```

This fact can be made into a new rewrite rule:

```
In[41]:= or[member[core[t_, x_], t_], not[member[t_, TOPS]]] := True
```

---

# SubstTest

The definition of **SubstTest** is very simple:

```
In[42]:= ?? SubstTest
```

```
SubstTest compares two different orders of evaluation
```

```
SubstTest[f_, x_, r_] := f @@ ({x} /. r) == (f @@ {x} /. r)
```

The idea of comparing different orders of evaluation is not only useful for deriving theorems, but has many other applications as well. For the purpose of deriving theorems, the head **f** is taken to be **and**, the various steps of the derivation are schematically indicated by a sequence of generic clauses **x**, and the replacement rules **r** replace the generic clauses with specific clauses that make all but the last item in **x** true. The idea is to arrange it so that  $(\{x\} /. r)$  becomes a list of the form  $\{\mathbf{True}, \mathbf{True}, \dots, \mathbf{not}[p]\}$  where  $p/.r$  is the theorem to be proved, and the expression **and[x]** is false. In one order of evaluation, one obtains **not[p]** and the other yields **False**, thereby disproving the negation of the desired theorem. As a practical matter, the main contribution to execution time here is the reduction of **and[x]** to **False**, which amounts to a check that the form of the argument is valid.

---

## removing set variables

The **class** rules in provide a standard procedure that can be used to eliminate set variables. Doing so often leads to new insights. For the particular theorem derived in the preceding section, one can eliminate one or both of the variables **x** and **t**. For example, when the variable **x** is removed, one finds a seemingly strange new result:

```
In[43]:= Map[equal[V, #] &, SubstTest[class, x, implies[
      member[t, z], member[core[t, x], t]], z → TOPS]] // Reverse
Out[43]= or[equal[V, image[inverse[CORE[t]], t]],
      not[member[t, TOPS]]] == True
```

Further investigation of this expression reveals that this new fact can be automatically recognized by the **GOEDEL** program as being true by using **assert**. In the present case the best procedure is to use **assert** to rewrite the conclusion:

```
In[44]:= assert[equiv[equal[V, image[inverse[CORE[t]], t]],
      equal[Uclosure[t], t]]]
Out[44]= True
```

---

## a new rewrite rule

The newly discovered fact can also be added to the **GOEDEL** program. In other words, one has not only proven the original theorem, but one has also discovered a new rewrite rule that one may otherwise have overlooked.

```
In[45]:= equal[V, image[inverse[CORE[t_]], t_]] := equal[t, Uclosure[t]]
```

Once this is done, the result derived above is rewritten to **True**.

```
In[46]:= implies[member[t, TOPS], equal[Uclosure[t], t]]
```

```
Out[46]= True
```

Incidentally, the other variable **t** can also be removed, yielding a variable-free formulation, whose truth is obvious from the definition of the class **TOPS**.

```
In[47]:= subclass[TOPS, fix[UCLOSURE]]
```

```
Out[47]= True
```

---

## generalizing theorems in topology

In deriving theorems of topology, considerable effort has been expended to make sure that results are stated in the most general form possible. In this way, whatever is proved can be applied to areas totally removed from topology, and in particular to algebra and number theory.

---

## an example: the class FULL

A class **x** is **full** if every member is a subclass.

```
In[48]:= assert[forall[w, implies[member[w, x], subclass[w, x]]]]
```

```
Out[48]= subclass[U[x], x]
```

The class of all full sets is called **FULL**.

```
In[49]:= class[x, subclass[U[x], x]]
```

```
Out[49]= FULL
```

---

## FULL as a pseudo-topology

The class **FULL** is closed under arbitrary intersections and unions.

```
In[50]:= Aclosure[FULL]
```

```
Out[50]= FULL
```

```
In[51]:= Uclosure[FULL]
```

```
Out[51]= FULL
```

It fails to be a topology on  $U[\mathbf{FULL}] = \mathbf{V}$  because it is not a set.

```
In[52]:= member[FULL, V]
```

```
Out[52]= False
```

The interior of the class **FINITE** of all finite sets with respect to the **FULL** pseudo-topology is the class **H[FINITE]** of hereditarily finite sets.

```
In[53]:= core[FULL, FINITE]
```

```
Out[53]= H[FINITE]
```

If the axiom of regularity holds, then the interior of **FULL** is the class **OMEGA** of all ordinal numbers.

```
In[54]:= implies[AxReg, equal[core[FULL, FULL], OMEGA]]
```

```
Out[54]= True
```

---

## `hull[x,y]` and `HULL[x]`

The class `hull[x,y]` is the intersection of all sets that contain `y` and belong to the class `x`.

```
In[55]:= class[u, forall[v,  
            implies[and[member[v, x], subclass[y, v]], member[u, v]]]]
```

```
Out[55]= hull[x, y]
```

If `y` is not a set, then `hull[x,y] = V`. In general, since there is no guarantee that there exists any set that contains `y` and belongs to `x`, the corresponding function need not be total.

```
In[56]:= lambda[y, hull[x, y]]
```

```
Out[56]= HULL[x]
```

The domain of `HULL[x]` is the class of all subsets of members of `x`.

```
In[57]:= domain[HULL[x]]
```

```
Out[57]= image[inverse[S], x]
```

---

## characterization of HULL[x]

The following characterization of **HULL[x]** functions was discovered:

```
In[58]:= implies[and[FUNCTION[x], idempotent[x],
                    subcommute[x, S], subclass[x, S]], equal[x, HULL[fix[x]]]]
```

```
Out[58]= True
```

The functions **ACLOSURE** and **UCLOSURE** satisfy these properties, and are therefore examples:

```
In[59]:= HULL[fix[ACLOSURE]]
```

```
Out[59]= ACLOSURE
```

```
In[60]:= HULL[fix[UCLOSURE]]
```

```
Out[60]= UCLOSURE
```

Another example is the function **IMAGE[inverse[S]]**, and more generally:

```
In[61]:= implies[and[idempotent[x], subclass[Id, x]],
                equal[IMAGE[x], HULL[fix[IMAGE[x]]]]]
```

```
Out[61]= True
```

Here **IMAGE[x]** is the function **lambda[y, image[x, y]]**.

---

## applications to topology

The characterization of **CORE** and **HULL** functions can be used to derive a number of interesting theorems. A simple application of interest in topology is the following formula that relates interiors with closures via relative complements:

```
In[62]:= HULL[image[RC[x], Uclosure[y]]]
```

```
Out[62]= composite[RC[x], CORE[y], RC[x]]
```

Here **RC[x]** is the relative complement function

```
In[63]:= class[pair[u, v], and[disjoint[u, v], equal[x, union[u, v]]]]
```

```
Out[63]= RC[x]
```

A second application to topology is the formula

```
In[64]:= composite[UCLOSURE, HULL[binclosed[CAP]]]
```

```
Out[64]= HULL[TOPS]
```

Here **binclosed[CAP]** is the class of all collections that are closed under binary intersection.

---

## other applications

Applications of these functions are not confined to topology. For example, the function **HULL**[**Z**], where **Z** is the class of all integers enters into the following formula for the function **INTADD** for integer addition.

```
In[65]:= composite[HULL[Z], COMPOSE, id[cart[Z, Z]]]
```

```
Out[65]= INTADD
```

The transitive closure **trv**[**x**] of a relation **x** is the smallest transitive relation that contains it; this is the union of all positive powers of **x**:

```
In[66]:= image[power[x], complement[singleton[0]]]
```

```
Out[66]= trv[x]
```

The class **trv**[**x**] can be used to construct the smallest class that contains a given set, and is invariant under **x**.

```
In[67]:= IMAGE[union[Id, trv[x]]]
```

```
Out[67]= HULL[invar[x]]
```

---

## unsolved problems

The research reported in this paper suggests some interesting mathematical problems, including the following:

1. Is the constructor **Aclosure** idempotent?
2. Do the functions **ACLOSURE** and **UCLOSURE** commute?
3. Is there a proper class with **Aclosure[x]** not equal to **fix[HULL[x]]**?
4. Must a proper class contain an infinite subset?

---

## conclusions

Computers are can be used in mathematical reasoning in many ways:

1. finding proofs
2. finding counterexamples
3. verifying proofs
4. simplifying proofs
5. formulating definitions
- 6 simplifying expressions
7. discovering new facts
8. calculating

Unlike most automated reasoning programs, the **GOEDEL** program does not do searches, neither for proofs nor for counter-examples. The program is not meant to replace automated reasoning programs, but is only intended to serve as a supplement to programs like **Otter** to help with formulating definitions and theorems, and to help in the discovery of new facts, and for verifying proofs. Because many mathematical facts are built into the program in the form of rewrite rules, one can sometimes take a partial proof obtained by hand and use the program to complete the proof.