**summary**

Deriving facts about functions with the **GOEDEL** program often requires using methods markedly different from what one is accustomed to from experience with hand-produced proofs. The goal of this notebook is to derive new rules which hopefully will enable one to use **APPLY** to derive facts about functions in a fashion resembling what one does by hand.

**removing some old rules**

Before getting started, some old rewrite rules are removed for which replacements will shortly be derived.

```math
In[2]:= \text{image[funpart[x_], singleton[y_]] =. }
```

The **GOEDEL** program until now has no membership rule for **APPLY**, but one can be derived using **AssertTest**.

```math
In[3]:= \text{member[w, APPLY[x, y]] // AssertTest}
Out[3]= \text{member[w, A[image[x, singleton[y]]], not[member[y, image[inverse[x], P[complement[singleton[w]]]]]]}
```

The only current rule for **APPLY** is removed:

```math
In[4]:= \text{APPLY[x_, y_] =. }
```

Another rewrite rule scheduled to be replaced is also removed:

```math
In[5]:= \text{member[pair[x_, y_], composite[complement[inverse[E]], z_]] =.}
```

**a new beginning**

The unwrapped membership rule derived in the preceding section will now be used to define **APPLY**.
The old equational definition can be rederived using Normality, but we now turn it around so that APPLY is no longer eliminated. Doing so has the advantage that it makes statements easier to read, but it has the disadvantage that now one will need to add many new rules about APPLY.

The above rule causes existing rewrite rules that involve the expression \( A[\text{image}[x, \text{singleton}[y]]] \) to become inoperative. Replacement rules for these will need to be derived.

### Immediate Corollaries

In this section some immediate consequences of the membership rule are derived. The fact that \( A[w] \) is a set when \( w \) is not empty yields a sethood rule:

\[
\text{In}[9]:= \text{SubstTest}[\text{member}, A[w], V, w \rightarrow \text{image}[x, \text{singleton}[y]]]
\]

\[
\text{Out}[9]= \text{member}[\text{APPLY}[x, y], V] = \text{member}[y, \text{domain}[x]]
\]

\[
\text{In}[10]:= \text{member}[\text{APPLY}[x_, y_], V] := \text{member}[y, \text{domain}[x]]
\]

Another version of the sethood rule for APPLY can be derived using Normality.

\[
\text{In}[11]:= \text{image}[V, \text{singleton}[\text{APPLY}[x, y]]] \quad \text{// Normality}
\]

\[
\text{Out}[11]= \text{image}[V, \text{singleton}[\text{APPLY}[x, y]]] = \text{image}[V, \text{intersection}[\text{domain}[x], \text{singleton}[y]]]
\]

\[
\text{In}[12]:= \text{image}[V, \text{singleton}[\text{APPLY}[x_, y_]]] := \text{image}[V, \text{intersection}[\text{domain}[x], \text{singleton}[y]]]
\]

The following corollary says that if evaluation produces a set, then the argument must have been in the domain.

\[
\text{In}[13]:= \text{SubstTest}[\text{implies}, \text{member}[u, z], \text{member}[u, V], u \rightarrow \text{APPLY}[x, y]]
\]

\[
\text{Out}[13]= \text{or}[\text{member}[y, \text{domain}[x]], \text{not}[\text{member}[\text{APPLY}[x, y], z]]] \quad \text{True}
\]

\[
\text{In}[14]:= \text{or}[\text{member}[y_, \text{domain}[x_]], \text{not}[\text{member}[\text{APPLY}[x_, y_], z_]]] := \text{True}
\]

The class \( A[w] \) is equal to \( V \) when \( w \) is empty.

\[
\text{In}[15]:= \text{SubstTest}[\text{equal}, V, A[w], w \rightarrow \text{image}[x, \text{singleton}[y]]]
\]

\[
\text{Out}[15]= \text{equal}[V, \text{APPLY}[x, y]] = \text{not}[\text{member}[y, \text{domain}[x]]]
\]

\[
\text{In}[16]:= \text{equal}[V, \text{APPLY}[x_, y_]] := \text{not}[\text{member}[y, \text{domain}[x]]]
\]

Here is another version of the same fact:
In[17]:= image[V, complement[APPLY[x, y]]] // Normality
Out[17]= image[V, complement[APPLY[x, y]]] = image[V, intersection[domain[x], singleton[y]]]

In[18]:= image[V, complement[APPLY[x_, y_]]] := image[V, intersection[domain[x], singleton[y]]]

---

**a replacement for the vertical section rule for funpart**

In this section a replacement will be derived for the vertical section rule for funpart that was removed earlier.

Lemma:

In[19]:= SubstTest[implies, and[equal[u, v], member[v, w]], member[u, w],
   {u -> 0, v -> image[x, singleton[y]], w -> range[SINGLETON]}]
Out[19]= or[member[y, domain[x]], not[member[image[x, singleton[y]], range[SINGLETON]]]] = True

In[20]:= (% / {x -> x_, y -> y_}) /. Equal -> SetDelayed

In[21]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
   p1 -> member[image[x, singleton[y]], range[SINGLETON]],
   p2 -> member[y, domain[x]], p3 -> member[y, V]]]
Out[21]= or[member[y, V], not[member[image[x, singleton[y]], range[SINGLETON]]]] = True

In[22]:= (% / {x -> x_, y -> y_}) /. Equal -> SetDelayed

To derive the replacement rule, we use a trick:

In[23]:= Map[complement[complement[#]] & , SubstTest[image, VERTSECT[complement[z]],
   singleton[y], z -> composite[complement[inverse[E]], funpart[x]]]]
Out[23]= image[funpart[x], singleton[y]] = singleton[APPLY[funpart[x], y]]

In[24]:= image[funpart[x_], singleton[y_]] := singleton[APPLY[funpart[x], y]]

The following rewrite rule is the reverse of the removed rule for funpart.

In[25]:= SubstTest[image, intersection[u, v], singleton[y],
   {u -> composite[Id, x], v -> complement[composite[Di, x]]}] // Reverse
Out[25]= intersection[complement[image[Di, image[x, singleton[y]]]]], image[x, singleton[y]] =
   singleton[APPLY[funpart[x], y]]

In[26]:= intersection[complement[image[Di, image[x_, singleton[y_]]]]],
   image[x_, singleton[y_]] := singleton[APPLY[funpart[x], y]]

The following is a replacement for another removed rule:

In[27]:= member[pair[x, y], composite[complement[inverse[E]], z]] // AssertTest
Out[27]= member[pair[x, y], composite[complement[inverse[E]], z]] =
   and[member[x, image[inverse[z], P[complement[singleton[y]]]]], member[y, V]]

In[28]:= member[pair[x_, y_], composite[complement[inverse[E]], z_]] :=
   and[member[x, image[inverse[z], P[complement[singleton[y]]]]], member[y, V]]
normalization issues

The following is added to normalize `APPLY[funpart[x], y]`:

\[
\text{In}[29]:= \quad \text{union}[\text{APPLY}[x, y], \text{complement}[\text{image}[V, \text{intersection}[\text{domain}[x], \text{singleton}[y]]]]] // \text{DoubleComplement}
\]

\[
\text{Out}[29]= \quad \text{union}[\text{APPLY}[x, y], \text{complement}[\text{image}[V, \text{intersection}[\text{domain}[x], \text{singleton}[y]]]]] = \text{APPLY}[x, y]
\]

This is a rule of a similar nature:

\[
\text{In}[31]:= \quad \text{union}[\text{APPLY}[x, y], \text{complement}[\text{image}[V, \text{singleton}[y]]]] // \text{DoubleComplement}
\]

\[
\text{Out}[31]= \quad \text{union}[\text{APPLY}[x, y], \text{complement}[\text{image}[V, \text{singleton}[y]]]] = \text{APPLY}[x, y]
\]

equality substitution rules

Lemma:

\[
\text{In}[33]:= \quad \text{SubstTest}[\text{implies}, \text{equal}[u, v], \text{equal}[A[u], A[v]], \{u \rightarrow \text{image}[w, \text{singleton}[x]], v \rightarrow \text{image}[y, \text{singleton}[z]]\}]
\]

\[
\text{Out}[33]= \quad \text{or}[\text{equal}[\text{APPLY}[w, x], \text{APPLY}[y, z]], \text{not}[\text{equal}[\text{image}[w, \text{singleton}[x]], \text{image}[y, \text{singleton}[z]]]]] = \text{True}
\]

\[
\text{In}[34]:= \quad (\% / \{w \rightarrow w, x \rightarrow x, y \rightarrow y, z \rightarrow z\}) / . \text{Equal} \rightarrow \text{SetDelayed}
\]

The rule for substitution on the first argument:

\[
\text{In}[35]:= \quad \text{Map}[\text{not}, \text{SubstTest}[\text{and}, \text{implies}[p1, p2], \text{implies}[p2, p3], \text{not}[[\text{implies}[p1, p3]], \{p1 \rightarrow \text{equal}[x, y], p2 \rightarrow \text{equal}[\text{image}[x, \text{singleton}[z]], \text{image}[y, \text{singleton}[z]]], p3 \rightarrow \text{equal}[\text{APPLY}[x, z], \text{APPLY}[y, z]]\}]]
\]

\[
\text{Out}[35]= \quad \text{or}[\text{equal}[\text{APPLY}[x, z], \text{APPLY}[y, z]], \text{not}[\text{equal}[x, y]]] = \text{True}
\]

\[
\text{In}[36]:= \quad \text{or}[\text{equal}[\text{APPLY}[x, z], \text{APPLY}[y, z]], \text{not}[\text{equal}[x, y]]] := \text{True}
\]

An application given below requires two lemmas. This is the first:

\[
\text{In}[37]:= \quad \text{SubstTest}[\text{implies}, \text{equal}[x, z], \text{equal}[\text{image}[x, \text{singleton}[y]], \text{image}[z, \text{singleton}[y]]], z \rightarrow \text{funpart}[x]]
\]

\[
\text{Out}[37]= \quad \text{or}[\text{equal}[\text{image}[x, \text{singleton}[y]], \text{singleton}[\text{APPLY}[\text{funpart}[x], y]]], \text{not}[\text{FUNCTION}[x]] = \text{True}
\]

\[
\text{In}[38]:= \quad (\% / \{x \rightarrow x, y \rightarrow y\}) / . \text{Equal} \rightarrow \text{SetDelayed}
\]

This is the second lemma:
The following result will be used to derive a simple consequence of the antitone property:

In[47]:= \text{.sequence expression}

Out[47]= \text{result expression}

In[48]:= \text{sequence expression}

Out[48]= \text{result expression}

This is the application:

In[49]:= \text{sequence expression}

Out[49]= \text{result expression}

This rewrite rule will be made permanent:

In[50]:= \text{sequence expression}

Out[50]= \text{result expression}

There is also a rule for substitution involving the second argument:

In[51]:= \text{sequence expression}

Out[51]= \text{result expression}

In[52]:= \text{sequence expression}

Out[52]= \text{result expression}

\textbf{an antitone property}

As an application of the equality substitution rule, an antitone property of APPLY is derived in this section. The starting point is this observation:

In[53]:= \text{sequence expression}

Out[53]= \text{result expression}

In[54]:= \text{sequence expression}

Out[54]= \text{result expression}

From this one fact one can deduce an antitone property:

In[55]:= \text{sequence expression}

Out[55]= \text{result expression}

In[56]:= \text{sequence expression}

Out[56]= \text{result expression}

The following result will be used to derive a simple consequence of the antitone property:
In[49]:= SubstTest[A, image[s, singleton[x]], s -> S] // Reverse

Out[49]= APPLY[S, x] = union[x, complement[image[V, singleton[x]]]]

In[50]:= APPLY[S, x_] := union[x, complement[image[V, singleton[x]]]]

The antitone property implies this:

In[51]:= SubstTest[implies, subclass[x, s], subclass[APPLY[s, y], APPLY[x, y]], s -> S]

Out[51]= or[notsubclass[x, S], subclass[y, APPLY[x, y]]] = True

In[52]:= or[notsubclass[x_, S], subclass[y_, APPLY[x_, y_]]] := True

rules for pairs

For binary functions, the following is useful:

In[53]:= SubstTest[A, image[x, singleton[w]], w -> PAIR[y, z]]

Out[53]= A[image[x, cart[singleton[y], singleton[z]]]] = APPLY[x, PAIR[y, z]]

In[54]:= A[image[x_, cart[singleton[y_], singleton[z_]]]] := APPLY[x, PAIR[y, z]]

The following two rules help to clean up expressions involving this expression.

In[55]:= union[APPLY[x, PAIR[y, z]], complement[image[V, singleton[y]]]] // DoubleComplement

Out[55]= union[APPLY[x, PAIR[y, z]], complement[image[V, singleton[y]]]] = APPLY[x, PAIR[y, z]]

In[56]:= union[APPLY[x_, PAIR[y_, z_]], complement[image[V, singleton[y_]]]] :=
     APPLY[x, PAIR[y, z]]

In[57]:= union[APPLY[x, PAIR[y, z]], complement[image[V, singleton[z]]]] // DoubleComplement

Out[57]= union[APPLY[x, PAIR[y, z]], complement[image[V, singleton[z]]]] = APPLY[x, PAIR[y, z]]

In[58]:= union[APPLY[x_, PAIR[y_, z_]], complement[image[V, singleton[z_]]]] :=
     APPLY[x, PAIR[y, z]]

For normalization one needs a further rule:

In[59]:= APPLY[x, PAIR[y, z]] // Normality // Reverse

Out[59]= union[APPLY[x, pair[y, z]], complement[image[V, singleton[y]]],
     complement[image[V, singleton[z]]]] = APPLY[x, PAIR[y, z]]

In[60]:= union[APPLY[x_, pair[y_, z_]], complement[image[V, singleton[y_]]],
     complement[image[V, singleton[z_]]]] := APPLY[x, PAIR[y, z]]

repeated application

All that one needs is this rule for composites:
The familiar rule for repeated application is a consequence when \( y \) is a function:

```plaintext
In[63]:= APPLY[composite[x, funpart[y]], z]
Out[63]= APPLY[x, APPLY[funpart[y], z]]
```

### reify rule

Whenever a new constructor is introduced, it is a good idea to add a corresponding reification rule.

```plaintext
In[64]:= SubstTest[reify, x, A[image[f[x], h[x]]], h[x] -> singleton[g[x]]]
Out[64]= reify[x, APPLY[f[x], g[x]]] =
    composite[Id, complement[composite[complement[inverse[E]]], SECOND, intersection[
        composite[inverse[FIRST], VERTSECT[reify[x, g[x]]], reify[x, f[x]]]]]]
```

```plaintext
In[65]:= reify[x_, APPLY[y_, z_]] :=
    composite[Id, complement[composite[complement[inverse[E]]], SECOND, intersection[
        composite[inverse[FIRST], VERTSECT[reify[x, z]]], reify[x, y]]]]
```