wrappers for constant functions

Johan G. F. Belinfante
2009 December 17

```
In[1]:= SetDirectory["l:"]; << goedel.09dec16a; << tools.m

        :Package Title: goedel.09dec16a              2009 December 16 at 1:10 p.m.

        It is now:  2009 Dec 17 at 14:4

        Loading Simplification Rules

        TOOLS.M                                    Revised 2009 November 2

        weightlimit = 40
```

summary

Wrappers for constant functions and for small constant functions are introduced in this notebook. The latter is used to derive a variable-free rewrite rule for the fact that there is no maximal constant function.

wrapper for constant functions

The following abbreviation is introduced for a constant function wrapper.

```
In[2]:= const[x_] := cart[domain[x], set[U[range[x]]]]
```

Theorem. Wrapper removal rule for constant functions.

```
In[3]:= equiv[and[FUNCTION[x], equal[x, cart[domain[x], range[x]]]],
           equal[x, cart[domain[x], set[U[range[x]]]]]] // not // not

Out[3]= True

In[4]:= equal[x_, cart[domain[x_], set[U[range[x_]]]]] :=
         and[equal[x, cart[domain[x], range[x]]], FUNCTION[x]]
```

wrapper for small constant functions

The compound wrapper **const[setpart[x]]** is a wrapper for small constant functions. A simplification of this wrapper leads to better rewrite rules. One does not have to be aware of this simplification however to use this compound wrapper.

Lemma.

```
In[5]:= or[equal[0, domain[setpart[x]]], equal[U[range[x]], U[range[setpart[x]]]]] // AssertTest

Out[5]= or[equal[0, domain[setpart[x]]], equal[U[range[x]], U[range[setpart[x]]]]] == True

In[6]:= or[equal[0, domain[setpart[x_]]], equal[U[range[x_]], U[range[setpart[x_]]]]] := True
```

Theorem. Simplification rule for the compound wrapper **const[setpart[x]]**, eliminating one of the two occurrences of **setpart**.

```
In[7]:= equal[cart[domain[setpart[x]], set[U[range[setpart[x]]]]],
          cart[domain[setpart[x]], set[U[range[x]]]]]

Out[7]= True

In[8]:= cart[domain[setpart[x_]], set[U[range[setpart[x_]]]]] :=
          cart[domain[setpart[x]], set[U[range[x]]]]
```

Lemma.

```
In[11]:= Map[implies[member[setpart[x], CONST], #] &,
            SubstTest[equal, t, const[t], t → setpart[x]]] // MapNotNot // Reverse

Out[11]= or[equal[cart[domain[setpart[x]], set[U[range[x]]]], setpart[x]],
           not[member[setpart[x], CONST]]] == True

In[12]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[14]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
            implies[and[p1, p3], p4], not[implies[p1, p4]], {p1 → member[x, CONST],
             p2 → member[setpart[x], CONST], p3 → equal[setpart[x], const[setpart[x]]],
             p4 -> equal[x, const[setpart[x]]]}]] // Reverse

Out[14]= or[equal[x, cart[domain[setpart[x]], set[U[range[x]]]]], not[member[x, CONST]]] == True

In[15]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. Wrapper removal rule for the compound wrapper **const[setpart[x]]**.

```
In[16]:= equiv[equal[x, const[setpart[x]]], member[x, CONST]] // not // not

Out[16]= True

In[17]:= equal[x_, cart[domain[setpart[x_]], set[U[range[x_]]]]] := member[x, CONST]
```

there is no maximal constant function

Lemma. Extending a constant function by adding a new point.

```
In[18]:= SubstTest[implies, equal[t, const[setpart[x]]],
           member[union[t, cart[set[u], set[U[range[x]]]]], CONST], t → x] // Reverse

Out[18]= or[member[union[x, cart[set[u], set[U[range[x]]]]], CONST],
           not[member[x, CONST]]] == True

In[19]:= (% /. x → x_) /. Equal→ SetDelayed
```

Lemma.

```
In[20]:= SubstTest[implies, and[member[x, z], member[y, z], subclass[x, y], not[equal[x, y]]],
           member[x, image[inverse[PS], z]],
           {y → union[x, cart[set[u], set[U[range[x]]]]], z → CONST}] // Reverse

Out[20]= or[member[x, image[inverse[PS], CONST]],
           member[pair[u, U[range[x]]], x], not[member[u, V]], not[member[x, CONST]],
           not[member[union[x, cart[set[u], set[U[range[x]]]]], CONST]]] == True

In[21]:= (% /. {u → u_, x → x_}) /. Equal→ SetDelayed
```

This result has a redundant literal that can be eliminated.

Lemma. (Removal of the redundant literal..)

```
In[22]:= Map[not, SubstTest[and, implies[and[p1, p2, p3], p4], implies[p1, p3],
            not[implies[and[p1, p2], p4]], {p1 → member[x, CONST], p2 → member[u, V],
             p3 -> member[union[x, cart[set[u], set[U[range[x]]]]], CONST],
             p4 -> or[member[x, image[inverse[PS], CONST]],
               member[pair[u, U[range[x]]], x]]}]] // Reverse

Out[22]= or[member[x, image[inverse[PS], CONST]],
           member[pair[u, U[range[x]]], x], not[member[u, V]], not[member[x, CONST]]] == True

In[23]:= (% /. {u → u_, x → x_}) /. Equal→ SetDelayed
```

Lemma. (Eliminating the variable **u**.)

```
In[24]:= (Map[equal[V, #] &, SubstTest[class, u, implies[and[member[x, y], member[u, V]],
              or[member[pair[u, t], x], member[x, z]]], {t → U[range[x]], y → CONST,
              z → image[inverse[PS], CONST]}]] // MapNotNot) /. x → setpart[t]

Out[24]= or[member[setpart[t], image[inverse[PS], CONST]],
           not[member[setpart[t], CONST]]] == True

In[25]:= (% /. t → t_) /. Equal→ SetDelayed
```

Lemma. (Eliminating the variable **t**.)

```
In[26]:= Map[equal[V, #] &,
          SubstTest[class, t, or[member[setpart[t], v], not[member[setpart[t], u]]],
           {u → CONST, v → image[inverse[PS], CONST]}]]

Out[26]= subclass[CONST, image[inverse[PS], CONST]] == True
```

```
In[27]:= % /. Equal → SetDelayed
```

Restatement. There is no maximal constant function.

```
In[28]:= empty[maximal[S, CONST]]

Out[28]= True
```

Lemma. (A reverse inclusion.)

```
In[29]:= SubstTest[subclass, image[inverse[PS], t], image[inverse[S], t], t → CONST] // Reverse

Out[29]= subclass[image[inverse[PS], CONST], CONST] == True

In[30]:= % /. Equal → SetDelayed
```

The two inclusions in opposite directions can be combined into an equation and made into a rewrite rule.

Theorem. A variable-free rewrite rule.

```
In[31]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> image[inverse[PS], CONST], v -> CONST}]

Out[31]= equal[CONST, image[inverse[PS], CONST]] == True

In[32]:= image[inverse[PS], CONST] := CONST
```