selective information

Johan G. F. Belinfante
2003 November 16

In[1]:= << goedel52.t17; << tools.m

It is now: 2003 Nov 17 at 9:31

Loading Simplification Rules

TOOLS.M Revised 2003 November 15

weightlimit = 40

summary

Mathematica has a built-in method for obtaining information about its rewrite rules. For example:

In[2]:= ?? Information

Information[symbol] prints information about a symbol. More...

Attributes[Information] = {HoldAll, Protected, ReadProtected}

Options[Information] = {LongForm -> True}

The number of rewrite rules differs greatly for various constructors:

In[3]:= Map[# , Length[DownValues[#]] & ,
Union[{UnaryFunctors, BinaryFunctors, {and, or, not, forall, exists}}]]

{cart, 15}, {cliques, 30}, {complement, 49}, {composite, 2502}, {CORE, 28},
{cross, 24}, {dif, 1}, {domain, 168}, {exists, 3}, {first, 6}, {fix, 550}, {flip, 1},
{forall, 1}, {funpart, 42}, {GLB, 14}, {GREATEST, 24}, {H, 8}, {HULL, 30},
{id, 6}, {image, 954}, {IMAGE, 44}, {intadd, 5}, {intersection, 865}, {invar, 34},
{inverse, 39}, {iterate, 34}, {LB, 37}, {LEAST, 24}, {LEFT, 4}, {LUB, 13}, {map, 3},
{MAXIMAL, 12}, {MINIMAL, 11}, {natadd, 7}, {natmul, 8}, {natsub, 14}, {not, 5},
{or, 635}, {P, 10}, {PAIR, 2}, {pairset, 4}, {plus, 5}, {power, 13}, {range, 131},
{rank, 11}, {RC, 4}, {RIGHT, 4}, {rotate, 97}, {second, 6}, {singleton, 46},
{subcommutant, 6}, {subvar, 42}, {succ, 3}, {symdif, 1}, {tc, 10}, {thinpart, 1},
{trv, 30}, {twist, 44}, {U, 156}, {UB, 38}, {Uclosure, 56}, {union, 241}, {VERTSECT, 62}]

When seeking information about rewrite rules in the GOEDEL program, it is advisable to first switch to the Goedel'Private' context.

In[4]:= Begin["Goedel'Private"]


If there are not many rewrite rules, one can quickly obtain information using Mathematica’s built-in Information[] or ??.
In[5]:= ?? ADJOIN

ADJOIN[x] is the function that takes y to union[x,y].

ADJOIN[0] := Id
ADJOIN[V] := 0
ADJOIN[complement[image[V, x_]]] := id[image[V, x]]
ADJOIN[image[V, x_]] := id[complement[image[V, x]]]
ADJOIN[x_] := 0 ; not[member[x, V]]

Since there are many rules for some constructors, such as composite, it helps to select those matching a given pattern. The tools.m file contains a tool for this.

In[6]:= ?? InfoMatch

Goedel 'InfoMatch

InfoMatch[x_] := TableForm[Module[{xstr = HoldPattern[<> ToString[Head[x]], xpatt = List @@ x], 
{StringReplace[ToString[#1], (HoldPattern[->] :> :[]=] &}],
Select[DownValues[Evaluate[Head[x]]], MatchQ[ToExpression[StringReplace[
ToString[First[#1]], xstr -> HoldPattern[List]] /. HoldPattern -> Identity, xpatt] &]]

examples

In this section some examples are presented to illustrate the usage of InfoMatch.

In[7]:= InfoMatch[Uclosure[image[inverse[x_], y_]]]

Out[7]//TableForm=

Uclosure[image[inverse[DORA], S]] := image[inverse[DORA], S]
Uclosure[image[inverse[DORA], inverse[S]]] := image[inverse[DORA], inverse[S]]
Uclosure[image[inverse[S], x_]] := P[U[x]]

In[8]:= InfoMatch[class[pair[u_, v_], w_]]

Out[8]//TableForm=

class[pair[u_, v_], True] := cart[class[u, True], class[v, True]]
class[pair[u_, v_], member[u_, v_]] := E ; AtomQ[u] && AtomQ[v]
class[pair[u_, v_], p_] := cart[class[u, p], class[v, True]] ; allfreeQ[p, v]
class[pair[u_, v_], p_] := cart[class[u, True], class[v, p]] ; allfreeQ[p, u]
class[pair[u_, v_], member[v_, u_]] := inverse[E] ; AtomQ[u] && AtomQ[v]
class[pair[pair[u_, v_], w_], p_] := composite[cart[pair[v, w], p], SECOND, id[cart[c.
class[pair[pair[u_, v_], w_], p_] := composite[cart[pair[u, w], p], FIRST, id[cart[V, 
class[pair[w_, pair[u_, v_]], p_] := composite[cart[pair[u, True], V]], inverse[SE]]
class[pair[w_, pair[u_, v_]], p_] := composite[cart[V, class[v, True]], inverse[FI]]
class[pair[u_, v_], equal[u_, v_]] := Id ; AtomQ[u] && AtomQ[v]
class[pair[u_, v_], equal[pair[V, u_], v_]] := LeftPairV
class[pair[u_, v_], equal[pair[u, V], v_]] := RightPairV
class[pair[u_, v_], equal[pair[V, v_], u_]] := inverse[LeftPairV]
class[pair[u_, v_], equal[pair[v, V], u_]] := inverse[RightPairV]
**warnings**

Warning: Careless application of `InfoMatch` sometimes does not produce what one might have expected. The pattern specified is evaluated before being used. For example, note that `equal` and `not` are switched due to preprocessing in the following:

```plaintext
In[9]:= InfoMatch[and[not[x_], equal[y___]]]
```

```
Out[9]//TableForm=
   and[equal[0, x_], not[member[x_, V]]] := False
   and[equal[0, second[x_]], not[member[first[x_], V]]] := False
   and[equal[x_, y_], not[subclass[x_, y_]]] := False
```

If the specified pattern exactly matches a rewrite rule, that rewrite rule will change the pattern, and one may obtain no information. For example:

```plaintext
In[10]:= InfoMatch[UClosure[image[inverse[S], x_]]]
```

```
Out[10]//TableForm=
```

One can get around this problem by placing `HoldPattern` in the body of the expression.

```plaintext
In[11]:= InfoMatch[UClosure[HoldPattern[image[inverse[S], x_]]]]
```

```
Out[11]//TableForm=
   UClosure[image[inverse[S], x_]] := P[U[x]]
```

This is especially useful when one wants to see wrapped membership rules, such as the following one for `DUP`.

```plaintext
In[12]:= InfoMatch[class[w_, HoldPattern[member[x_, DUP]]]]
```

```
Out[12]//TableForm=
   class[z_, member[w_, DUP]] := Module[{u = Unique[]}, class[z, exists[u, equal[pair[u, ]
```

---

**the effect of Blank**

```plaintext
In[13]:= ?? Blank
```

```plaintext
_ or Blank[ ] is a pattern object that can stand for any Mathematica expression. _h or Blank[h] can stand for any expression with head h. More...
```

```plaintext
Attributes[Blank] = {Protected}
```

The presence of `AtomQ` in some of the `class` rules is affected by `Blank`. Compare for example:

```plaintext
In[14]:= class[pair[x, y], member[u, v]]
```

```
Out[14]= cart[V, image[V, intersection[v, singleton[u]]]]
```

```plaintext
In[15]:= class[pair[x_, y_], member[u_, v_]]
```

```
Out[15]= class[pair[x_, y_], member[u_, v_]]
```
In[16]:= ? BlankSequence

__ (two _ characters) or BlankSequence[ ] is a pattern object that can stand for any sequence of one or more Mathematica expressions. __h or BlankSequence[h] can stand for any sequence of one or more expressions, all of which have head h.

More...

Attributes[BlankSequence] = {Protected}

The use of BlankSequence may not work as expected for certain constructors due to preprocessing. For example:

In[17]:= InfoMatch[natadd[x__]]

Out[17]//TableForm=

The cure here is to write at least one variable explicitly:

In[18]:= InfoMatch[natadd[x_, y__]]

Out[18]//TableForm=

be patient!

If there are many rewrite rules for a constructor, the information may take some time to be produced, so one must be patient. For example:

In[19]:= InfoMatch[composite[IMAGE[BIGCUP], x__]]

Out[19]//TableForm=

info.nb