a wrapper for partitions

Johan G. F. Belinfante
2010 January 7

```
In[1]:= SetDirectory["l:"]; << goedel.10jan06a; << tools.m

        :Package Title: goedel.10jan06a              2010 January 6 at 7:55 p.m.

        It is now:  2010 Jan 7 at 11:50

        Loading Simplification Rules

        TOOLS.M                                    Revised 2009 December 17

        weightlimit = 40

In[2]:= Begin["Goedel`Private`"];
```

summary

A wrapper **ptn[x]** for partitions has been introduced in the **GOEDEL** program, defined by this **class**-wrapped membership rule:

```
In[3]:= FirstMatch[class[s_, member[t_, HoldPattern[ptn[x_]]]]]

Out[3]= class[s_, member[t_, ptn[x_]]] := class[s, and[member[t, x],
          not[member[0, x]], subclass[x, union[P[complement[t]], set[t]]]]]
```

Some basic properties of this wrapper are derived in this notebook.

simplification rules for cliques and spine

Some general simplification rules for cliques and spines of restrictions are derived in this section.

Theorem.

```
In[4]:= SubstTest[subclass, cart[x, x],
          intersection[u, v], {u → union[Id, z], v → cart[y, y]}] // Reverse

Out[4]= subclass[cart[x, x], union[composite[id[y], z, id[y]], id[y]]] ==
         and[subclass[x, y], subclass[cart[x, x], union[Id, z]]]

In[5]:= subclass[cart[x_, x_], union[composite[id[y_], z_, id[y_]], id[y_]]] :=
         and[subclass[x, y], subclass[cart[x, x], union[Id, z]]]
```

Theorem.

```
In[6]:= SubstTest[cliques, intersection[t, cart[x, x]], t → union[Id, y]] // Reverse

Out[6]= cliques[union[composite[id[x], y, id[x]], id[x]]] ==
         intersection[cliques[union[Id, y]], P[x]]

In[7]:= cliques[union[composite[id[x_], y_, id[x_]], id[x_]]] :=
         intersection[cliques[union[Id, y]], P[x]]
```

Comment. For the following theorem, a slightly better result is obtained using **Renormality** than with **Normality**. The latter yields almost the same formula, but with an extra unneeded intersection with **x**.

Theorem.

```
In[9]:= spine[restrict[y, x, x], z] // Renormality

Out[9]= spine[composite[id[x], y, id[x]], z] ==
         intersection[complement[image[V, intersection[z, complement[x]]]], spine[y, z]]

In[10]:= spine[composite[id[x_], y_, id[x_]], z_] :=
          intersection[complement[image[V, intersection[z, complement[x]]]], spine[y, z]]
```

Corollary.

```
In[12]:= SubstTest[spine, restrict[t, x, x], z, t → union[Id, y]] // Reverse

Out[12]= spine[union[composite[id[x], y, id[x]], id[x]], z] == intersection[
           complement[image[V, intersection[z, complement[x]]]], spine[union[Id, y], z]]

In[13]:= spine[union[composite[id[x_], y_, id[x_]], id[x_]], z_] := intersection[
           complement[image[V, intersection[z, complement[x]]]], spine[union[Id, y], z]]
```

---

normalization

In this section a normalization rule for **ptn[x]** is derived, and some connections with **spine** that make it easy to derive properties of the **ptn** wrapper.

Lemma. Simplification rule.

```
In[15]:= spine[union[DISJOINT, Id], x] // Normality // Reverse

Out[15]= intersection[x, complement[fix[composite[Di, id[x], E, inverse[E]]]]] ==
          spine[union[DISJOINT, Id], x]

In[16]:= intersection[x_, complement[fix[composite[Di, id[x_], E, inverse[E]]]]] :=
          spine[union[DISJOINT, Id], x]
```

Theorem. Normalization rule.

```
In[17]:= ptn[x] // Normality // Reverse

Out[17]= intersection[complement[image[V, intersection[x, set[0]]]],
           spine[union[DISJOINT, Id], x]] == ptn[x]

In[18]:= intersection[complement[image[V, intersection[x_, set[0]]]],
           spine[union[DISJOINT, Id], x_]] := ptn[x]
```

Observartion. The class **ptn[x]** is the spine of **x** with respect to the restriction of the relation **union[DISJOINT, Id]** to the class of sets that do not hold **0**. It is convenient to introduce the temporary abbreviation **UDJ** for this relation.

```
In[19]:= UDJ := union[composite[id[complement[set[0]]], DISJOINT, id[complement[set[0]]],
           id[complement[set[0]]]]
```

The connection between **ptn** and **spine** is this:

```
In[20]:= spine[UDJ, x]

Out[20]= ptn[x]
```

The complexity of the formula for **UDJ** is not the primary reason for introducing the special wrapper **ptn**. A much more important advantage of **ptn** is that its membership rule is wrapped with **class**, and accordingly membership statements of the form **u ∈ ptn[x]** do not expand out.

Observation. Simplification rule.

```
In[25]:= subclass[cart[x, x], UDJ]

Out[25]= and[not[member[0, x]], subclass[cart[x, x], union[DISJOINT, Id]]]
```

basic properties of ptn[x]

Theorem.

```
In[21]:= Map[empty, dif[ptn[x], x] // Renormality]

Out[21]= subclass[ptn[x], x] == True

In[22]:= subclass[ptn[x_], x_] := True
```

Theorem.

```
In[23]:= member[0, ptn[x]] // AssertTest

Out[23]= member[0, ptn[x]] == False

In[24]:= member[0, ptn[x_]] := False
```

Theorem. The class **ptn[x]** is pairwise disjoint.

```
In[26]:= SubstTest[subclass, cartsq[spine[t, x]], union[t, inverse[t]], t → UDJ] // Reverse

Out[26]= subclass[cart[ptn[x], ptn[x]], union[DISJOINT, Id]] == True

In[27]:= subclass[cart[ptn[x_], ptn[x_]], union[DISJOINT, Id]] := True
```

Corollary.

```
In[28]:= SubstTest[implies, and[member[r, s], subclass[s, t]], member[r, t],
           {r → pair[u, v], s → cart[ptn[x], ptn[x]], t → union[DISJOINT, Id]}] // Reverse

Out[28]= or[equal[0, intersection[u, v]], equal[u, v],
           not[member[u, ptn[x]]], not[member[v, ptn[x]]]] == True

In[29]:= or[equal[0, intersection[u_, v_]], equal[u_, v_],
           not[member[u_, ptn[x_]]], not[member[v_, ptn[x_]]]] := True
```

Theorem. Wrapper removal rule.

```
In[30]:= SubstTest[equal, x, spine[t, x],
           t → union[composite[id[complement[set[0]]], DISJOINT, id[complement[set[0]]]],
             id[complement[set[0]]]]] // Reverse

Out[30]= equal[x, ptn[x]] == and[not[member[0, x]], subclass[cart[x, x], union[DISJOINT, Id]]]

In[31]:= equal[x_, ptn[x_]] := and[not[member[0, x]], subclass[cart[x, x], union[DISJOINT, Id]]]
```

Theorem. Simplification rule.

```
In[32]:= intersection[complement[set[0]], ptn[x]] // Renormality

Out[32]= intersection[complement[set[0]], ptn[x]] == ptn[x]

In[33]:= intersection[complement[set[0]], ptn[x_]] := ptn[x]
```

other properties of ptn[x]

Theorem. To each partition there corresponds an equivalence class.

```
In[34]:= SubstTest[EQUIVALENCE, composite[inverse[funpart[t]], funpart[t]],
           t → composite[id[ptn[x]], E]] // Reverse

Out[34]= EQUIVALENCE[composite[inverse[E], id[ptn[x]], E]] == True

In[35]:= EQUIVALENCE[composite[inverse[E], id[ptn[x_]], E]] := True
```

Theorem. Every member of a partition is maximal.

```
In[36]:= SubstTest[or, equal[0, intersection[t, image[PS, t]]], member[0, t],
           not[subclass[cart[t, t], union[DISJOINT, Id]]], t → ptn[x]] // Reverse

Out[36]= equal[0, intersection[image[PS, ptn[x]], ptn[x]]] == True
```

```
In[37]:= intersection[image[PS, ptn[x_]], ptn[x_]] := 0
```

Lemma.

```
In[38]:= SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]],
           {t → id[S], u → cartsq[ptn[x]], v → UDJ}] // Reverse

Out[38]= subclass[composite[id[ptn[x]], S, id[ptn[x]]], id[complement[set[0]]]] == True

In[39]:= subclass[composite[id[ptn[x_]], S, id[ptn[x_]]], id[complement[set[0]]]] := True
```

Theorem. If one member of a partition is a subset of another, then they are equal.

```
In[40]:= SubstTest[implies, and[member[t, w], subclass[w, z]],
            member[t, z], {t → pair[u, v], w -> composite[id[ptn[x]], S, id[ptn[x]]],
             z -> id[complement[set[0]]]}] // MapNotNot // Reverse

Out[40]= or[equal[u, v], not[member[u, ptn[x]]],
           not[member[v, ptn[x]]], not[subclass[u, v]]] == True

In[41]:= or[equal[u_, v_], not[member[u_, ptn[x_]]],
           not[member[v_, ptn[x_]]], not[subclass[u_, v_]]] := True
```

---

reify rule for ptn[x]

Theorem.

```
In[42]:= SubstTest[reify, x, intersection[complement[image[V, intersection[f[x], set[0]]]],
            spine[u, f[x]]], u → union[DISJOINT, Id]] // Reverse

Out[42]= reify[x, ptn[f[x]]] == composite[intersection[
            complement[composite[intersection[Di, composite[E, inverse[E]]], reify[x, f[x]]]],
            reify[x, f[x]]], id[complement[image[inverse[reify[x, f[x]]], set[0]]]]]

In[43]:= reify[x_, ptn[y_]] := composite[intersection[
            complement[composite[intersection[Di, composite[E, inverse[E]]], reify[x, y]]],
            reify[x, y]], id[complement[image[inverse[reify[x, y]], set[0]]]]]
```

Theorem.

```
In[44]:= SubstTest[reify, z, spine[t, z], t → restrict[y, x, x]] /. y -> union[DISJOINT, Id]

Out[44]= GREATEST[union[composite[id[x], DISJOINT, id[x]], id[x]]] ==
          composite[GREATEST[union[DISJOINT, Id]], id[P[x]]]

In[45]:= GREATEST[union[composite[id[x_], DISJOINT, id[x_]], id[x_]]] :=
          composite[GREATEST[union[DISJOINT, Id]], id[P[x]]]
```

Obssrvation. The function that takes **x** to **ptn[x]** is **λx. ptn[x] = VERTSECT[reify[x, ptn[x]]]**.

```
In[46]:= VERTSECT[reify[x, ptn[x]]]

Out[46]= union[cart[complement[P[complement[set[0]]]], set[0]],
          composite[VERTSECT[GREATEST[union[DISJOINT, Id]]], id[P[complement[set[0]]]]]]
```

The function **λx. ptn[x]** is a total function.

```
In[47]:= domain[VERTSECT[reify[x, ptn[x]]]]

Out[47]= V
```

Theorem. A general theorem.

```
In[48]:= Map[equal[#, image[VERTSECT[GREATEST[x]], P[y]]] &,
          SubstTest[range, VERTSECT[GREATEST[t]], t -> restrict[x, y, V]]]

Out[48]= equal[image[VERTSECT[GREATEST[x]], P[y]], intersection[cliques[x], P[y]]] == True
```

```
In[49]:= image[VERTSECT[GREATEST[x_]], P[y_]] := intersection[cliques[x], P[y]]
```

Observation. The range of **λx. ptn[x]** is the class of all partitions: **range[λx. ptn[x]] = PARTNS**.

```
In[50]:= range[VERTSECT[reify[x, ptn[x]]]]

Out[50]= PARTNS
```