HULL[fix[UCHAINS]]

Johan G. F. Belinfante
2007 June 25

```
In[1]:= SetDirectory["l:"]; << goedel94.24a; << tools.m

        :Package Title: goedel94.24a      2007 June 24 at 9:00 a.m.

        It is now:  2007 Jun 25 at 4:9

        Loading Simplification Rules

        TOOLS.M                                      Revised 2007 June 23

        weightlimit = 40
```

summary

The class **Uchains[x]** is the class of all unions of chains $\mathbf{a} \subset \mathbf{b} \subset \cdots$ in a class **x**. In this notebook, the function **UCHAINS** which takes any set **x** to the set **Uchains[x]** is studied. This function resembles the function **UCLOSURE**, and satisfies many similar properties. Much of the theory about unions of chains can be restated in variable-free form using this function. An open question is whether **UCHAINS** is idempotent. It is shown that this question is equivalent to the question whether **UCHAINS** is the same as the idempotent function **HULL[fix[UCHAINS]]**. Both of these functions are total, both are subclasses of the subset relation **S** and both subcommute with **S**.

definition and normalization of UCHAINS

The function **UCHAINS** is defined by the membership rule:

```
In[2]:= member[x_, UCHAINS] := and[equal[second[x], Uchains[first[x]]], member[first[x], V]]
```

Theorem.

```
In[3]:= FUNCTION[UCHAINS] // AssertTest
Out[3]= FUNCTION[UCHAINS] == True

In[4]:= FUNCTION[UCHAINS] := True
```

Theorem. (Normalization rule.)

```
In[5]:= UCHAINS // VSNormality // Reverse
Out[5]= composite[IMAGE[BIGCUP], IMAGE[id[chains[S]]], POWER] == UCHAINS
```

```
In[6]:= composite[IMAGE[BIGCUP], IMAGE[id[chains[S]]], POWER] := UCHAINS
```

Theorem. The function **UCHAINS** is total.

```
In[7]:= IminComp[composite[IMAGE[BIGCUP], IMAGE[id[chains[S]]]], POWER, V]
Out[7]= domain[UCHAINS] == V
```

```
In[8]:= domain[UCHAINS] := V
```

Theorem. The function **UCHAINS** is increasing.

```
In[9]:= subclass[UCHAINS, S] // AssertTest
Out[9]= subclass[UCHAINS, S] == True
```

```
In[10]:= subclass[UCHAINS, S] := True
```

---

composites involving UCHAINS

Theorem.

```
In[11]:= composite[inverse[E], UCHAINS] // RelnRenormality // Reverse
Out[11]= composite[BIGCUP, id[chains[S]], inverse[S]] == composite[inverse[E], UCHAINS]
```

```
In[12]:= composite[BIGCUP, id[chains[S]], inverse[S]] := composite[inverse[E], UCHAINS]
```

Corollary.

```
In[13]:= Assoc[composite[BIGCUP, id[chains[S]]], inverse[S], inverse[S]] // Reverse
Out[13]= composite[inverse[E], UCHAINS, inverse[S]] == composite[inverse[E], UCHAINS]
```

```
In[14]:= composite[inverse[E], UCHAINS, inverse[S]] := composite[inverse[E], UCHAINS]
```

Theorem.

```
In[15]:= composite[UCHAINS, CLIQUES] // VSNormality
Out[15]= composite[UCHAINS, CLIQUES] == CLIQUES
```

```
In[16]:= composite[UCHAINS, CLIQUES] := CLIQUES
```

Theorem.

```
In[17]:= composite[UCHAINS, POWER] // RelnNormality
Out[17]= composite[UCHAINS, POWER] == POWER
```

```
In[18]:= composite[UCHAINS, POWER] := POWER
```

Theorem.

```
In[19]:= composite[UCHAINS, UCLOSURE] // VSNormality
```

```
Out[19]= composite[UCHAINS, UCLOSURE] == UCLOSURE
```

```
In[20]:= composite[UCHAINS, UCLOSURE] := UCLOSURE
```

Corollary.

```
In[21]:= Map[subclass[fix[#], fix[UCHAINS]] &,
           Assoc[UCHAINS, UCLOSURE, id[fix[UCLOSURE]]]] // Reverse
```

```
Out[21]= subclass[fix[UCLOSURE], fix[UCHAINS]] == True
```

```
In[22]:= subclass[fix[UCLOSURE], fix[UCHAINS]] := True
```

Theorem.

```
In[23]:= Map[assert, SubstTest[implies, subclass[x, y], subclass[image[w, x], image[w, y]],
             {w → inverse[S], x → fix[UCLOSURE], y → fix[UCHAINS]}]] // Reverse
```

```
Out[23]= equal[V, image[inverse[S], fix[composite[S, UCHAINS]]]] == True
```

```
In[24]:= image[inverse[S], fix[UCHAINS]] := V
```

---

vertical section and APPLY rules

Lemma.

```
In[25]:= Uchains[union[x, complement[image[V, y]]]] // Normality
```

```
Out[25]= Uchains[union[x, complement[image[V, y]]]] == union[complement[image[V, y]], Uchains[x]]
```

```
In[26]:= Uchains[union[x_, complement[image[V, y_]]]] :=
          union[complement[image[V, y]], Uchains[x]]
```

Theorem. (Vertical section rule.)

```
In[27]:= image[UCHAINS, set[x]] // Normality
```

```
Out[27]= image[UCHAINS, set[x]] == set[Uchains[x]]
```

```
In[28]:= image[UCHAINS, set[x_]] := set[Uchains[x]]
```

Lemma. (Corollary of the sethood rule.)

```
In[29]:= image[V, set[Uchains[x]]] // Normality
```

```
Out[29]= image[V, set[Uchains[x]]] == image[V, set[x]]
```

```
In[30]:= image[V, set[Uchains[x_]]] := image[V, set[x]]
```

Theorem. (Function application rule.)

```
In[31]:= SubstTest[A, image[t, set[x]], t → UCHAINS]

Out[31]= APPLY[UCHAINS, x] == union[complement[image[V, set[x]]], Uchains[x]]

In[32]:= APPLY[UCHAINS, x_] := union[complement[image[V, set[x]]], Uchains[x]]
```

---

monotonicity

Lemma.

```
In[33]:= Map[composite[Id, complement[#]] &,
          dif[S, composite[inverse[UCHAINS], S, UCHAINS]] // complement // RelnRenormality]

Out[33]= intersection[S, composite[inverse[UCHAINS], complement[S], UCHAINS]] == 0

In[34]:= % /. Equal → SetDelayed
```

Theorem. Monotonicity property.

```
In[35]:= SubstTest[empty, dif[u, v], {u → S, v -> composite[inverse[UCHAINS], S, UCHAINS]}]

Out[35]= subclass[S, composite[inverse[UCHAINS], S, UCHAINS]] == True

In[36]:= subclass[S, composite[inverse[UCHAINS], S, UCHAINS]] := True
```

Corollary. The function **UCHAINS** subcommutes with **S**.

```
In[37]:= SubstTest[implies, subclass[u, v], subclass[composite[t, u], composite[t, v]],
           {t → UCHAINS, u -> S, v -> composite[inverse[UCHAINS], S, UCHAINS]}] // Reverse

Out[37]= subclass[composite[UCHAINS, S], composite[S, UCHAINS]] == True

In[38]:= subclass[composite[UCHAINS, S], composite[S, UCHAINS]] := True
```

Corollary.

```
In[39]:= SubstTest[implies, subclass[u, v], subclass[composite[u, w], composite[v, w]],
           {u -> composite[UCHAINS, S], v -> composite[S, UCHAINS], w → inverse[UCHAINS]}] //
          Reverse

Out[39]= subclass[composite[UCHAINS, S, inverse[UCHAINS]], S] == True

In[40]:= subclass[composite[UCHAINS, S, inverse[UCHAINS]], S] := True
```

use of reify

The vertical section rule derived in the preceding section can be used to derive many additional rules for **UCHAINS**. In particular, the vertical section rules make it easy to use **reify**, which is often much faster than other techniques. This dramatic speed-up is illustrated by the basic formula **UCHAINS = lambda[x, Uchains[x]]**.

```
In[43]:= VERTSECT[reify[x, Uchains[x]]] // Timing
Out[43]= {0. Second, UCHAINS}

In[44]:= lambda[x, Uchains[x]] // Timing
Out[44]= {3.25 Second, UCHAINS}
```

Theorem.

```
In[45]:= composite[UCLOSURE, UCHAINS] // ReifNormality
Out[45]= composite[UCLOSURE, UCHAINS] == UCLOSURE

In[46]:= composite[UCLOSURE, UCHAINS] := UCLOSURE
```

Other examples:

```
In[47]:= composite[UCHAINS, ADJOIN[set[0]]] // ReifNormality
Out[47]= composite[UCHAINS, ADJOIN[set[0]]] == UCHAINS

In[48]:= composite[UCHAINS, ADJOIN[set[0]]] := UCHAINS

In[49]:= composite[BIGCAP, UCHAINS] // ReifNormality
Out[49]= composite[BIGCAP, UCHAINS] == cart[V, set[0]]

In[50]:= composite[BIGCAP, UCHAINS] := cart[V, set[0]]

In[51]:= composite[BIGCUP, UCHAINS] // ReifNormality
Out[51]= composite[BIGCUP, UCHAINS] == BIGCUP

In[52]:= composite[BIGCUP, UCHAINS] := BIGCUP

In[53]:= composite[UCHAINS, CHAINS] // ReifNormality
Out[53]= composite[UCHAINS, CHAINS] == CHAINS

In[54]:= composite[UCHAINS, CHAINS] := CHAINS

In[55]:= composite[UCHAINS, SUBVAR] // ReifNormality
Out[55]= composite[UCHAINS, SUBVAR] == SUBVAR

In[56]:= composite[UCHAINS, SUBVAR] := SUBVAR
```

allclosed formula

Lemma.

```
In[57]:= fix[UCHAINS] // Normality // Reverse
Out[57]= fix[composite[S, UCHAINS]] == fix[UCHAINS]

In[58]:= fix[composite[S, UCHAINS]] := fix[UCHAINS]
```

Theorem. The class of sets closed under the infinitary operation **composite[BIGCUP, id[chains[S]]]** is **fix[UCHAINS]**.

```
In[59]:= SubstTest[fix, composite[S, IMAGE[t], POWER], t -> composite[BIGCUP, id[chains[S]]]]
Out[59]= allclosed[composite[BIGCUP, id[chains[S]]]] == fix[UCHAINS]

In[60]:= allclosed[composite[BIGCUP, id[chains[S]]]] := fix[UCHAINS]
```

Corollary.

```
In[61]:= SubstTest[fix, HULL[allclosed[t]], t -> composite[BIGCUP, id[chains[S]]]] // Reverse
Out[61]= fix[HULL[fix[UCHAINS]]] == fix[UCHAINS]

In[62]:= fix[HULL[fix[UCHAINS]]] := fix[UCHAINS]
```

Corollary.

```
In[63]:= SubstTest[Aclosure, allclosed[t], t -> composite[BIGCUP, id[chains[S]]]] // Reverse
Out[63]= Aclosure[fix[UCHAINS]] == fix[UCHAINS]

In[64]:= Aclosure[fix[UCHAINS]] := fix[UCHAINS]
```

Theorem.

```
In[65]:= Assoc[UCHAINS, id[fix[UCHAINS]], HULL[fix[UCHAINS]]]
Out[65]= composite[UCHAINS, HULL[fix[UCHAINS]]] == HULL[fix[UCHAINS]]

In[66]:= composite[UCHAINS, HULL[fix[UCHAINS]]] := HULL[fix[UCHAINS]]
```

Uchains and hull[fix[UCHAINS],x]

Lemma.

```
In[67]:= Map[equal[#, hull[fix[UCHAINS], x]] &, ApComp[UCHAINS, HULL[fix[UCHAINS]], x]]
Out[67]= or[equal[hull[fix[UCHAINS], x], Uchains[hull[fix[UCHAINS], x]]], not[member[x, V]]] ==
          True
```

```
In[68]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[69]:= SubstTest[implies, equal[t, V],
           equal[Uchains[t], t], t -> hull[fix[UCHAINS], x]] // Reverse
```

```
Out[69]= or[equal[hull[fix[UCHAINS], x], Uchains[hull[fix[UCHAINS], x]]], member[x, V]] == True
```

```
In[70]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem.

```
In[71]:= SubstTest[and, implies[p, q], or[p, q],
           {p → member[x, V], q -> equal[hull[fix[UCHAINS], x], Uchains[hull[fix[UCHAINS], x]]]}]
```

```
Out[71]= equal[hull[fix[UCHAINS], x], Uchains[hull[fix[UCHAINS], x]]] == True
```

```
In[72]:= Uchains[hull[fix[UCHAINS], x_]] := hull[fix[UCHAINS], x]
```

Corollary.

```
In[73]:= SubstTest[implies, subclass[x, t],
           subclass[Uchains[x], Uchains[t]], t → hull[fix[UCHAINS], x]] // Reverse
```

```
Out[73]= subclass[Uchains[x], hull[fix[UCHAINS], x]] == True
```

```
In[74]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[75]:= SubstTest[implies, subclass[x, t], subclass[hull[w, x], hull[w, t]],
           {w → fix[UCHAINS], t → Uchains[x]}] // Reverse
```

```
Out[75]= subclass[hull[fix[UCHAINS], x], hull[fix[UCHAINS], Uchains[x]]] == True
```

```
In[76]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[77]:= SubstTest[implies, subclass[u, v], subclass[hull[t, u], hull[t, v]],
           {t → fix[UCHAINS], u → Uchains[x], v → hull[fix[UCHAINS], x]}] // Reverse
```

```
Out[77]= subclass[hull[fix[UCHAINS], Uchains[x]], hull[fix[UCHAINS], x]] == True
```

```
In[78]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem.

```
In[79]:= SubstTest[and, subclass[u, v], subclass[v, u],
           {u -> hull[fix[UCHAINS], Uchains[x]], v -> hull[fix[UCHAINS], x]}]
```

```
Out[79]= equal[hull[fix[UCHAINS], x], hull[fix[UCHAINS], Uchains[x]]] == True
```

```
In[80]:= hull[fix[UCHAINS], Uchains[x_]] := hull[fix[UCHAINS], x]
```

This can be restated in variable-free form as follows:

```
In[81]:= composite[HULL[fix[UCHAINS]], UCHAINS] // ReifNormality
Out[81]= composite[HULL[fix[UCHAINS]], UCHAINS] == HULL[fix[UCHAINS]]

In[82]:= composite[HULL[fix[UCHAINS]], UCHAINS] := HULL[fix[UCHAINS]]
```

Theorem.

```
In[83]:= subclass[UCHAINS, composite[inverse[S], HULL[fix[UCHAINS]]]] // AssertTest
Out[83]= subclass[UCHAINS, composite[inverse[S], HULL[fix[UCHAINS]]]] == True

In[84]:= subclass[UCHAINS, composite[inverse[S], HULL[fix[UCHAINS]]]] := True
```

open questions

Unresolved question: Are the functions **HULL[fix[UCHAINS]]** and **UCHAINS** equal? This reduces to the question whether **UCHAINS** is idempotent:

```
In[85]:= SubstTest[implies, and[FUNCTION[t], idempotent[t], subclass[t, S], subcommute[t, S]],
          equal[t, HULL[fix[t]]], t → UCHAINS]
Out[85]= True == or[equal[UCHAINS, HULL[fix[UCHAINS]]],
            not[equal[UCHAINS, composite[UCHAINS, UCHAINS]]]]
```

It has been shown that **x ⊂ Uchains[x] ⊂ Uchains[Uchains[x]] ⊂ ⋯ ⊂ hull[fix[UCHAINS], x]**. Can this chain be infinite?