# JOIN: concatenating lists

Johan G. F. Belinfante
2008 October 21

```
In[1]:= SetDirectory["l:"]; << goedel.08oct21a;<< tools.m

        :Package Title: goedel.08oct21a          2008 October 21 at 12:40 p.m.

        It is now:  2008 Oct 22 at 6:55

        Loading Simplification Rules

        TOOLS.M                                    Revised 2008 October 21

        weightlimit = 40
```

## summary

The function **JOIN** that concatenates lists is defined, and is shown to be associative.

## definition

Definition.

```
In[2]:= composite[CUP, intersection[composite[inverse[SECOND], SECOND], composite[
            inverse[FIRST], COMPOSE, cross[Id, composite[INVERSE, PLUS, IMAGE[FIRST]]]]],
          SWAP, id[cart[LISTS, LISTS]]] := JOIN
```

## FUNCTION rule

```
In[3]:= SubstTest[FUNCTION,
          composite[funpart[t], intersection[composite[inverse[SECOND], SECOND], composite[
             inverse[FIRST], COMPOSE, cross[Id, composite[INVERSE, PLUS, IMAGE[FIRST]]]]],
           SWAP, id[cart[LISTS, LISTS]], t → CUP] // Reverse

Out[3]= FUNCTION[JOIN] == True

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

## domain

Theorem.

```
In[5]:= SubstTest[domain,
          composite[t, intersection[composite[inverse[SECOND], SECOND], composite[
             inverse[FIRST], COMPOSE, cross[Id, composite[INVERSE, PLUS, IMAGE[FIRST]]]]],
           SWAP, id[cart[LISTS, LISTS]], t → CUP] // Reverse

Out[5]= domain[JOIN] == cart[LISTS, LISTS]

In[6]:= domain[JOIN] := cart[LISTS, LISTS]
```

## APPLY rules

Theorem.

```
In[7]:= SubstTest[image, funpart[t], set[PAIR[x, y]], t → JOIN] // Reverse

Out[7]= image[JOIN, cart[set[x], set[y]]] == set[APPLY[JOIN, PAIR[x, y]]]

In[8]:= image[JOIN, cart[set[x_], set[y_]]] := set[APPLY[JOIN, PAIR[x, y]]]
```

Theorem.

```
In[9]:= SubstTest[APPLY,
          composite[t, intersection[composite[inverse[SECOND], SECOND], composite[
             inverse[FIRST], COMPOSE, cross[Id, composite[INVERSE, PLUS, IMAGE[FIRST]]]]],
           SWAP, id[cart[LISTS, LISTS]], PAIR[list[x], list[y]], t → CUP]

Out[9]= union[composite[list[y], inverse[plus[domain[list[x]]]]], list[x]] ==
          APPLY[JOIN, PAIR[list[x], list[y]]]

In[10]:= union[composite[list[y_], inverse[plus[domain[list[x_]]]]], list[x_]] :=
           APPLY[JOIN, PAIR[list[x], list[y]]]
```

Corollary.

```
In[11]:= SubstTest[union, composite[list[x], inverse[plus[domain[list[t]]]]], list[t], t → 0]

Out[11]= APPLY[JOIN, PAIR[0, list[x]]] == list[x]

In[12]:= APPLY[JOIN, PAIR[0, list[x_]]] := list[x]
```

Corollary.

```
In[13]:= SubstTest[union, composite[list[y], inverse[plus[domain[list[x]]]]], list[x], y → 0]

Out[13]= APPLY[JOIN, PAIR[list[x], 0]] == list[x]
```

```
In[14]:= APPLY[JOIN, PAIR[list[x_], 0]] := list[x]
```

Corollary.

```
In[15]:= SubstTest[APPLY, JOIN, PAIR[list[x], 0], x → 0] // Reverse

Out[15]= APPLY[JOIN, PAIR[0, 0]] == 0

In[16]:= APPLY[JOIN, PAIR[0, 0]] := 0
```

## LEFT[0] and RIGHT[0] rules

Corollary.

```
In[17]:= Map[composite[VERTSECT[#], id[LISTS]] &,
          SubstTest[reify, x, APPLY[JOIN, PAIR[t, list[x]]], t → 0]]

Out[17]= composite[JOIN, LEFT[0]] == id[LISTS]

In[18]:= composite[JOIN, LEFT[0]] := id[LISTS]
```

Corollary.

```
In[19]:= Map[composite[VERTSECT[#], id[LISTS]] &,
           SubstTest[reify, x, union[composite[t, inverse[plus[domain[list[x]]]]], list[x]],
             t → list[y]] /. y → 0] // Reverse

Out[19]= composite[JOIN, RIGHT[0]] == id[LISTS]

In[20]:= composite[JOIN, RIGHT[0]] := id[LISTS]
```

## range

```
In[21]:= Map[subclass[#, range[JOIN]] &, ImageComp[JOIN, RIGHT[0], V]]

Out[21]= subclass[LISTS, range[JOIN]] == True

In[22]:= % /. Equal→ SetDelayed
```

Theorem.

```
In[23]:= SubstTest[implies,
           and[FUNCTION[u], FUNCTION[v], disjoint[domain[u], domain[v]]], FUNCTION[union[u, v]],
           {u → list[x], v -> composite[list[y], inverse[plus[domain[list[x]]]]]}] // Reverse

Out[23]= FUNCTION[APPLY[JOIN, PAIR[list[x], list[y]]]] == True

In[24]:= FUNCTION[APPLY[JOIN, PAIR[list[x_], list[y_]]]] := True
```

Lemma.

```
In[25]:= SubstTest[domain, union[u, v],
          {u → list[x], v -> composite[list[y], inverse[plus[domain[list[x]]]]]}]

Out[25]= union[domain[list[x]], image[plus[domain[list[x]]], domain[list[y]]]] ==
          domain[APPLY[JOIN, PAIR[list[x], list[y]]]]

In[26]:= union[domain[list[x_]], image[plus[domain[list[x_]]], domain[list[y_]]]] :=
          domain[APPLY[JOIN, PAIR[list[x], list[y]]]]
```

Theorem.

```
In[27]:= SubstTest[implies, and[member[u, omega], member[v, omega]],
           equal[natadd[u, v], union[u, image[plus[u], v]]],
           {u → domain[list[x]], v → domain[list[y]]}] // Reverse

Out[27]= equal[domain[APPLY[JOIN, PAIR[list[x], list[y]]]],
           natadd[domain[list[x]], domain[list[y]]]] == True

In[28]:= domain[APPLY[JOIN, PAIR[list[x_], list[y_]]]] :=
          natadd[domain[list[x]], domain[list[y]]]
```

The following observation motivates the next lemma.

```
In[29]:= member[APPLY[JOIN, PAIR[list[x], list[y]]], LISTS]

Out[29]= True
```

Lemma. (This takes quite a while.)

```
In[30]:= Map[or[subclass[range[JOIN], LISTS], empty[#]] &, SubstTest[reify, x,
           dif[set[APPLY[JOIN, PAIR[list[first[x]], list[second[x]]]]], t], t -> LISTS]]

Out[30]= subclass[range[JOIN], LISTS] == True

In[31]:= % /. Equal→ SetDelayed
```

Theorem.

```
In[32]:= SubstTest[and, subclass[u, v], subclass[v, u], {u → range[JOIN], v → LISTS}]

Out[32]= equal[LISTS, range[JOIN]] == True

In[33]:= range[JOIN] := LISTS
```

## the associative law

Lemma.

```
In[34]:= SubstTest[union, composite[list[z], inverse[plus[domain[list[t]]]]],
           list[t], t → APPLY[JOIN, PAIR[list[x], list[y]]]] // Reverse

Out[34]= union[APPLY[JOIN, PAIR[list[x], list[y]]],
           composite[list[z], inverse[plus[natadd[domain[list[x]], domain[list[y]]]]]] ==
         APPLY[JOIN, PAIR[APPLY[JOIN, PAIR[list[x], list[y]]], list[z]]]

In[35]:= union[APPLY[JOIN, PAIR[list[x_], list[y_]]],
           composite[list[z_], inverse[plus[natadd[domain[list[x_]], domain[list[y_]]]]]] :=
         APPLY[JOIN, PAIR[APPLY[JOIN, PAIR[list[x], list[y]]], list[z]]]
```

Lemma.

```
In[36]:= SubstTest[composite, union[u, v], inverse[plus[domain[list[x]]]],
           {u → list[y], v -> composite[list[z], inverse[plus[domain[list[y]]]]]}] // Reverse

Out[36]= composite[APPLY[JOIN, PAIR[list[y], list[z]]], inverse[plus[domain[list[x]]]]] ==
          union[composite[list[y], inverse[plus[domain[list[x]]]]],
           composite[list[z], inverse[plus[natadd[domain[list[x]], domain[list[y]]]]]]

In[37]:= composite[APPLY[JOIN, PAIR[list[y_], list[z_]]], inverse[plus[domain[list[x_]]]]] :=
          union[composite[list[y], inverse[plus[domain[list[x]]]]],
           composite[list[z], inverse[plus[natadd[domain[list[x]], domain[list[y]]]]]]
```

Theorem.

```
In[38]:= SubstTest[union, composite[list[t], inverse[plus[domain[list[x]]]]],
          list[x], t → APPLY[JOIN, PAIR[list[y], list[z]]]]

Out[38]= APPLY[JOIN, PAIR[list[x], APPLY[JOIN, PAIR[list[y], list[z]]]]] ==
          APPLY[JOIN, PAIR[APPLY[JOIN, PAIR[list[x], list[y]]], list[z]]]

In[39]:= APPLY[JOIN, PAIR[list[x_], APPLY[JOIN, PAIR[list[y_], list[z_]]]]] :=
          APPLY[JOIN, PAIR[APPLY[JOIN, PAIR[list[x], list[y]]], list[z]]]
```

## removing the variables

Lemma. (This takes quite a while.)

```
In[40]:= Map[composite[VERTSECT[#], id[LISTS]] &, Map[composite[#, id[LISTS]] &,
           SubstTest[reify, z, APPLY[t, PAIR[PAIR[list[x], list[y]], list[z]]],
            t → composite[JOIN, cross[Id, JOIN], ASSOC]]]]

Out[40]= composite[JOIN, cross[Id, JOIN], ASSOC,
           id[cart[cart[set[list[x]], set[list[y]]], LISTS]], inverse[SECOND]] ==
         composite[JOIN, LEFT[APPLY[JOIN, PAIR[list[x], list[y]]]]]

In[41]:= composite[JOIN, cross[Id, JOIN], ASSOC,
           id[cart[cart[set[list[x_]], set[list[y_]]], LISTS]], inverse[SECOND]] :=
         composite[JOIN, LEFT[APPLY[JOIN, PAIR[list[x], list[y]]]]]
```

Lemma. (Simplification rule.)

```
In[42]:= Map[inverse, composite[intersection[composite[inverse[FIRST], SECOND],
             composite[inverse[SECOND], JOIN, cross[Id, JOIN], ASSOC]],
           id[cart[V, LISTS]], inverse[FIRST]] // inverse // TripleRotate]

Out[42]= composite[intersection[composite[inverse[FIRST], SECOND],
           composite[inverse[SECOND], JOIN, cross[Id, JOIN], ASSOC]],
         id[cart[V, LISTS]], inverse[FIRST]] ==
        composite[SWAP, inverse[rotate[composite[JOIN, cross[Id, JOIN], ASSOC, SWAP]]]]

In[43]:= % /. Equal → SetDelayed
```

Lemma.

```
In[44]:= Map[composite[#, id[LISTS]] &, SubstTest[reify, y,
           composite[t, id[cart[cart[set[list[x]], set[list[y]]], LISTS]], inverse[SECOND]],
           t → composite[JOIN, cross[Id, JOIN], ASSOC]]]

Out[44]= composite[SWAP,
           inverse[rotate[composite[JOIN, cross[Id, JOIN], ASSOC, SWAP]]], LEFT[list[x]]] ==
         composite[SWAP, inverse[rotate[composite[JOIN, SWAP]]], JOIN, LEFT[list[x]]]

In[45]:= composite[SWAP,
           inverse[rotate[composite[JOIN, cross[Id, JOIN], ASSOC, SWAP]]], LEFT[list[x_]]] :=
         composite[SWAP, inverse[rotate[composite[JOIN, SWAP]]], JOIN, LEFT[list[x]]]
```

Lemma. (Simplification rule.)

```
In[46]:= composite[JOIN, cross[Id, JOIN], ASSOC, id[cart[cart[LISTS, LISTS], V]]] // TripleRotate

Out[46]= composite[JOIN, cross[Id, JOIN], ASSOC, id[cart[cart[LISTS, LISTS], V]]] ==
         composite[JOIN, cross[Id, JOIN], ASSOC]

In[47]:= % /. Equal → SetDelayed
```

Theorem. (Variable-free statement of the associative law.)

```
In[48]:= Map[composite[rotate[inverse[rotate[inverse[#]]]], id[cart[cart[LISTS, LISTS], V]]] &,
          Map[composite[#, id[LISTS]] &, SubstTest[reify, x, composite[t, LEFT[list[x]]], t ->
            composite[SWAP, inverse[rotate[composite[JOIN, cross[Id, JOIN], ASSOC, SWAP]]]]]]]

Out[48]= composite[JOIN, cross[Id, JOIN], ASSOC] == composite[JOIN, cross[JOIN, Id]]

In[49]:= composite[JOIN, cross[Id, JOIN], ASSOC] := composite[JOIN, cross[JOIN, Id]]
```