RATPLUS

Johan G. F. Belinfante
2012 November 22

```
In[1]:= SetDirectory["l:"]; << goedel.12nov21a
        :Package Title: goedel.12nov21a              2012 November 21 at 7:00 a.m.
        Loading takes about sixteen minutes, half that time due to builtin pauses.
        It is now:  2012 Nov 22 at 16:18
        Loading Simplification Rules
        TOOLS.M is now incorporated in the GOEDEL program as of 2010 September 3
        weightlimit = 40
        Loading completed.
        It is now:  2012 Nov 22 at 16:33
```

summary

The function **RATPLUS** is defined and some of its properties are derived.

rules for RATADD

In this preliminary section some new rewrite rules for **RATADD** are derived, some of which are needed later.

Theorem. Vertical section rule for **RATADD**.

```
In[2]:= SubstTest[image, funpart[t], set[w], {t → RATADD, w→ PAIR[x, y]}] // Reverse
Out[2]= image[RATADD, cart[set[x], set[y]]] == set[ratadd[x, y]]
```

```
In[3]:= image[RATADD, cart[set[x_], set[y_]]] := set[ratadd[x, y]]
```

Theorem. Simplification rule.

```
In[4]:= composite[SWAP, inverse[RATADD]] // DoubleInverse
Out[4]= composite[SWAP, inverse[RATADD]] == inverse[RATADD]
```

```
In[5]:= composite[SWAP, inverse[RATADD]] := inverse[RATADD]
```

Theorem. Simplification rule.

```
In[6]:= composite[SWAP, cross[x, y], inverse[RATADD]] // DoubleInverse

Out[6]= composite[SWAP, cross[x, y], inverse[RATADD]] == composite[cross[y, x], inverse[RATADD]]

In[7]:= composite[SWAP, cross[x_, y_], inverse[RATADD]] :=
          composite[cross[y, x], inverse[RATADD]]
```

definition and basic properties of RATPLUS

Definition.

```
In[8]:= APPLY[CURRY, RATADD] := RATPLUS
```

The general theory of Curryied binary operations can be applied to derive many of the basic properties of **RATPLUS**.

Theorem. The class **RATPLUS** is a set.

```
In[9]:= SubstTest[member, APPLY[CURRY, binop[t]], V, t → RATADD] // Reverse

Out[9]= member[RATPLUS, V] == True

In[10]:= member[RATPLUS, V] := True
```

Theorem. The set **RATPLUS** is a function.

```
In[11]:= SubstTest[FUNCTION, APPLY[CURRY, binop[t]], t → RATADD] // Reverse

Out[11]= FUNCTION[RATPLUS] == True

In[12]:= FUNCTION[RATPLUS] := True
```

Theorem. Domain rule.

```
In[13]:= SubstTest[domain, APPLY[CURRY, binop[t]], t → RATADD] // Reverse

Out[13]= domain[RATPLUS] == RATS

In[14]:= domain[RATPLUS] := RATS
```

Corollary. The set **RATPLUS** is not empty.

```
In[15]:= SubstTest[empty, domain[funpart[x]], x → RATPLUS]

Out[15]= equal[0, RATPLUS] == False

In[16]:= equal[0, RATPLUS] := False
```

Theorem. Mapping rule. **RATPLUS: RATS ⟶ (RATS ⟶ RATS)**.

```
In[17]:= SubstTest[member, APPLY[CURRY, binop[x]], map[fix[domain[binop[x]]],
           map[fix[domain[binop[x]]], fix[domain[binop[x]]]]], x → RATADD] // Reverse

Out[17]= member[RATPLUS, map[RATS, map[RATS, RATS]]] == True

In[18]:= member[RATPLUS, map[RATS, map[RATS, RATS]]] := True
```

Theorem. Function application rule.

```
In[19]:= SubstTest[APPLY, APPLY[CURRY, binop[t]], x, t → RATADD] // Reverse

Out[19]= APPLY[RATPLUS, x] == union[complement[image[V, intersection[RATS, set[x]]]], ratplus[x]]

In[20]:= APPLY[RATPLUS, x_] :=
          union[complement[image[V, intersection[RATS, set[x]]]], ratplus[x]]
```

Theorem. Vertical section rule.

```
In[21]:= SubstTest[image, funpart[t], set[x], t → RATPLUS] // Reverse

Out[21]= image[RATPLUS, set[x]] ==
          intersection[image[V, intersection[RATS, set[x]]], set[ratplus[x]]]

In[22]:= image[RATPLUS, set[x_]] :=
          intersection[image[V, intersection[RATS, set[x]]], set[ratplus[x]]]
```

Theorem. Inverse image rule.

```
In[23]:= (member[x, image[inverse[funpart[t]], y]] // AssertTest) /. t → RATPLUS

Out[23]= member[x, image[inverse[RATPLUS], y]] == and[member[x, RATS], member[ratplus[x], y]]

In[24]:= member[x_, image[inverse[RATPLUS], y_]] := and[member[x, RATS], member[ratplus[x], y]]
```

---

one-one property

Lemma.

```
In[25]:= SubstTest[implies, equal[u, v], equal[APPLY[u, w], APPLY[v, w]],
           {u → ratplus[rat[x]], v → ratplus[rat[y]], w → cart[Z, set[id[omega]]]}] // Reverse

Out[25]= or[equal[rat[x], rat[y]], not[equal[ratplus[rat[x]], ratplus[rat[y]]]]] == True

In[26]:= or[equal[rat[x_], rat[y_]], not[equal[ratplus[rat[x_]], ratplus[rat[y_]]]]] := True
```

Theorem. (Remove the **rat** wrappers.)

```
In[27]:= SubstTest[implies, and[equal[x, rat[u]], equal[y, rat[v]],
            equal[ratplus[x], ratplus[y]]], equal[x, y], {u → x, v → y}] // Reverse

Out[27]= or[equal[x, y], not[equal[ratplus[x], ratplus[y]]],
           not[member[x, RATS]], not[member[y, RATS]]] == True
```

```
In[28]:= or[equal[x_, y_], not[equal[ratplus[x_], ratplus[y_]]],
           not[member[x_, RATS]], not[member[y_, RATS]]] := True
```

Theorem. The function **RATPLUS** is one-to-one.

```
In[29]:= Map[equal[0, composite[Id, complement[class[pair[x, y], implies[#, equal[x, y]]]]]] &,
          member[pair[x, y], composite[inverse[RATPLUS], RATPLUS]] // AssertTest]

Out[29]= FUNCTION[inverse[RATPLUS]] == True

In[30]:= FUNCTION[inverse[RATPLUS]] := True
```

---

eval rule

Theorem.

```
In[31]:= SubstTest[composite, eval[x], APPLY[CURRY, binop[t]], t → RATADD] // Reverse

Out[31]= composite[eval[x], RATPLUS] == ratplus[x]

In[32]:= composite[eval[x_], RATPLUS] := ratplus[x]
```

Theorem. Reification rule for the function constructor **ratplus**.

```
In[33]:= SubstTest[reify, x, composite[t, LEFT[f[x]]], t → RATADD] // Reverse

Out[33]= reify[x, ratplus[f[x]]] ==
          composite[cross[inv[RATADD], Id], inverse[RATADD], VERTSECT[reify[x, f[x]]]]

In[34]:= reify[x_, ratplus[y_]] :=
          composite[cross[inv[RATADD], Id], inverse[RATADD], VERTSECT[reify[x, y]]]
```

Corollary. Uncurry rule.

```
In[35]:= Map[flip[rotate[inverse[#]]] &, SubstTest[reify, x, composite[eval[x], t], t → RATPLUS]]

Out[35]= composite[inverse[SINGLETON], IMG, cross[RATPLUS, SINGLETON]] == RATADD

In[36]:= composite[inverse[SINGLETON], IMG, cross[RATPLUS, SINGLETON]] := RATADD
```

Theorem. Simplification rule.

```
In[37]:= SubstTest[composite, FUNPART, APPLY[CURRY, binop[x]], x → RATADD] // Reverse

Out[37]= composite[FUNPART, RATPLUS] == RATPLUS

In[38]:= composite[FUNPART, RATPLUS] := RATPLUS
```

Theorem. Serendipity.

```
In[39]:= image[inverse[RATPLUS], set[0]] // Renormality
Out[39]= image[inverse[RATPLUS], set[0]] == 0

In[40]:= image[inverse[RATPLUS], set[0]] := 0
```

Theorem.

```
In[41]:= SubstTest[composite, inverse[E], APPLY[CURRY, binop[t]], t → RATADD] // Reverse
Out[41]= composite[inverse[E], RATPLUS] == composite[cross[inv[RATADD], Id], inverse[RATADD]]

In[42]:= composite[inverse[E], RATPLUS] := composite[cross[inv[RATADD], Id], inverse[RATADD]]
```

serendipity: normalization rules for ratplus[x]

The following rewrite rules were also discovered in the course of this study.

Theorem.

```
In[43]:= ratplus[x] // FastReifNormality// Reverse
Out[43]= composite[id[image[V, intersection[RATS, set[x]]]], ratplus[x]] == ratplus[x]

In[44]:= composite[id[image[V, intersection[RATS, set[x_]]]], ratplus[x_]] := ratplus[x]
```

Corollary.

```
In[45]:= composite[ratplus[x], id[image[V, intersection[RATS, set[x]]]]] // FastReifNormality
Out[45]= composite[ratplus[x], id[image[V, intersection[RATS, set[x]]]]] == ratplus[x]

In[46]:= composite[ratplus[x_], id[image[V, intersection[RATS, set[x_]]]]] := ratplus[x]
```