a parabolic function

Johan G. F. Belinfante
2003 May 17

```
In[1]:= << goedel52.r74; << tools.m

        :Package Title: goedel52.r74        2003 May 16 at 9:55 p.m.

        It is now:  2003 May 22 at 11:12

        Loading Simplification Rules

        TOOLS.M                       Revised 2003 May 21

        weightlimit = 40
```

■ summary

This notebook contains a detailed illustration of how a recursive definition of a function can be replaced by an iterative definition. For any infinite sequence of numbers **{n[0], n[1], n[2], n[3], ... }** there is a corresponding sequence of partial sums, **{0, n[0], n[0]+n[1], n[0]+n[1]+n[2], ... }**. In particular, for the special case of the sequence of natural numbers **id[omega] = {0, 1, 2, 3, ... }**, the **n**th partial sum is a parabolic function of **n**. It is shown how this function **PARABOLA** can be defined using iteration. From the definition of **PARABOLA** in terms of **iterate**, some simple properties of this function are derived, including a recursion relation for **PARABOLA**.

■ reduction to an iteration problem

The **n**th partial sum **p[n]** for the sequence **{0, 1, 2, 3, ... }** satisfies the recursion relation **p[n+1] = p[n] + n**, which involves both **n** and **p[n]**. To convert this recursion relation to an iteration problem, one first forms the sequence of ordered pairs **pair[n, p[n]]**. The set whose only member is the initial pair for this sequence is

```
In[2]:= singleton[pair[0, 0]] == id[singleton[0]]

Out[2]= True
```

The next pair **pair[n+1, p[n+1]]** in this sequence of pairs is obtained by applying the function that takes **pair[x,y]** to **pair[x+1,x+y]**. This function is easily constructed from **SUCC** and **NATADD**:

```
In[3]:= APPLY[composite[cross[composite[SUCC, FIRST], NATADD], DUP], PAIR[x, y]]

Out[3]= PAIR[succ[x], natadd[x, y]]
```

The sequence of ordered pairs is therefore given by the following expression for which the temporary abbreviation **ITER** is introduced.

```
In[4]:= ITER := iterate[composite[cross[composite[SUCC, FIRST], NATADD], DUP], id[singleton[0]]]
```

The basic strategy is to first derive a formula expressing this sequence in terms of the sequence **PARABOLA** of partial sums of the sequence of natural numbers, and then to turn around and use that formula to derive the properties of **PARABOLA**.

■ footnote: a comment about PAIR[x,y] versus pair[x,y]

The **GOEDEL** program actually has two ordered pairs, **pair[x,y]** and **PAIR[x,y]**, the latter defined by:

```
In[5]:= A[cart[singleton[x], singleton[y]]]

Out[5]= PAIR[x, y]
```

These pairs agree when **x** and **y** are sets, but behave differently for proper classes. To apply binary functions only to bonafide pairs of sets, one should use **PAIR**. The constructor **pair[x,y]** in the **GOEDEL** program is a primitive notion, whose postulated properties are compatible with the ordered pair in our **Otter** work, which uses Quaife's modification of Kuratowski's ordered pair.

■ some properties of ITER

The class **ITER** is a function:

```
In[6]:= SubstTest[FUNCTION, iterate[funpart[s], singleton[t]],
             {s -> composite[cross[composite[SUCC, FIRST], NATADD], DUP], t -> PAIR[0, 0]}]

Out[6]= FUNCTION[iterate[intersection[composite[inverse[SECOND], NATADD],
             composite[inverse[FIRST], SUCC, FIRST]], cart[singleton[0], singleton[0]]]] == True

In[7]:= FUNCTION[iterate[intersection[composite[inverse[SECOND], NATADD],
             composite[inverse[FIRST], SUCC, FIRST]], cart[singleton[0], singleton[0]]]] := True
```

This can be rewritten as:

```
In[8]:= FUNCTION[ITER]

Out[8]= True
```

It will now be shown that **range[ITER]** is a subclass of **cart[omega,omega]**. This is accomplished by two applications of **SubstTest**. The first one uses a general fact about the range of **iterate[x,y]**.

```
In[9]:= SubstTest[subclass, range[iterate[s, t]], union[range[s], t],
             {s -> composite[cross[composite[SUCC, FIRST], NATADD], DUP], t -> id[singleton[0]]}]

Out[9]= subclass[range[iterate[intersection[composite[inverse[SECOND], NATADD],
              composite[inverse[FIRST], SUCC, FIRST]], cart[singleton[0], singleton[0]]]],
           union[cart[singleton[0], singleton[0]], composite[id[omega],
             S, inverse[SUCC], id[omega]]]]] == True

In[10]:= subclass[range[iterate[intersection[composite[inverse[SECOND], NATADD],
              composite[inverse[FIRST], SUCC, FIRST]], cart[singleton[0], singleton[0]]]],
           union[cart[singleton[0], singleton[0]], composite[id[omega],
             S, inverse[SUCC], id[omega]]]]] := True
```

The second application just uses the transitive property of **subclass**.

```
In[11]:= SubstTest[implies, and[subclass[r, s], subclass[s, t]], subclass[r, t],
          {r -> range[ITER], s -> union[cart[singleton[0], singleton[0]],
          composite[id[omega], S, inverse[SUCC], id[omega]]], t -> cart[omega, omega]}]
```

```
Out[11]= subclass[range[iterate[intersection[
           composite[inverse[SECOND], NATADD], composite[inverse[FIRST], SUCC, FIRST]],
          cart[singleton[0], singleton[0]]]]], cart[omega, omega]] == True
```

```
In[12]:= subclass[range[iterate[intersection[
           composite[inverse[SECOND], NATADD], composite[inverse[FIRST], SUCC, FIRST]],
          cart[singleton[0], singleton[0]]]]], cart[omega, omega]] := True
```

This can be rewritten:

```
In[13]:= subclass[range[ITER], cart[omega, omega]]
```

```
Out[13]= True
```

The following fact follows immediately:

```
In[14]:= equal[composite[id[cart[omega, omega]], ITER], ITER]
```

```
Out[14]= True
```

Replacing **equal** with **Equal** yields:

```
In[15]:= Equal[composite[id[cart[omega, omega]], ITER], ITER]
```

```
Out[15]= composite[id[cart[omega, omega]],
           iterate[intersection[composite[inverse[SECOND], NATADD],
             composite[inverse[FIRST], SUCC, FIRST]], cart[singleton[0], singleton[0]]]]] ==
          iterate[intersection[composite[inverse[SECOND], NATADD],
            composite[inverse[FIRST], SUCC, FIRST]], cart[singleton[0], singleton[0]]]
```

This fact will be added as a rewrite rule:

```
In[16]:= composite[id[cart[omega, omega]],
           iterate[intersection[composite[inverse[SECOND], NATADD],
             composite[inverse[FIRST], SUCC, FIRST]], cart[singleton[0], singleton[0]]]]] :=
          iterate[intersection[composite[inverse[SECOND], NATADD],
            composite[inverse[FIRST], SUCC, FIRST]], cart[singleton[0], singleton[0]]]
```

A closely related rewrite rule is also needed:

```
In[17]:= Assoc[id[cart[V, V]], id[cart[omega, omega]], ITER]
```

```
Out[17]= composite[id[cart[V, V]], iterate[intersection[composite[inverse[SECOND], NATADD],
             composite[inverse[FIRST], SUCC, FIRST]], cart[singleton[0], singleton[0]]]]] ==
          iterate[intersection[composite[inverse[SECOND], NATADD],
            composite[inverse[FIRST], SUCC, FIRST]], cart[singleton[0], singleton[0]]]
```

```
In[18]:= composite[id[cart[V, V]], iterate[intersection[composite[inverse[SECOND], NATADD],
             composite[inverse[FIRST], SUCC, FIRST]], cart[singleton[0], singleton[0]]]]] :=
          iterate[intersection[composite[inverse[SECOND], NATADD],
            composite[inverse[FIRST], SUCC, FIRST]], cart[singleton[0], singleton[0]]]
```

In other words:

```
In[19]:= composite[id[cart[V, V]], ITER] == ITER
```

```
Out[19]= True
```

■ composite[FIRST,ITER]

Note what happens when one applies the function **composite[FIRST,ITER]** to the first few natural numbers:

```
In[20]:= NestList[succ, 0, 4]

Out[20]= {0, singleton[0], succ[singleton[0]],
          succ[succ[singleton[0]]], succ[succ[succ[singleton[0]]]]}

In[21]:= Map[APPLY[composite[FIRST, ITER], #] &, %]

Out[21]= {0, singleton[0], succ[singleton[0]],
          succ[succ[singleton[0]]], succ[succ[succ[singleton[0]]]]}
```

This observation suggests the conjecture:

```
In[22]:= composite[FIRST, ITER] == id[omega];
```

This conjecture will now be proved by using iteration uniqueness:

```
In[23]:= implies[and[equal[composite[s, r], composite[r, SUCC]],
                 equal[image[r, singleton[0]], t]],
          equal[composite[r, id[omega]], iterate[s, t]]]

Out[23]= True
```

The idea is to compare two solutions of the same iteration problem:

```
In[24]:= SubstTest[implies, and[equal[composite[s, r], composite[r, SUCC]],
                 equal[image[r, singleton[0]], t]],
          equal[composite[r, id[omega]], iterate[s, t]],
             {r -> composite[FIRST, ITER], s -> SUCC, t -> singleton[0]}]

Out[24]= equal[composite[FIRST, iterate[intersection[
              composite[inverse[SECOND], NATADD], composite[inverse[FIRST], SUCC, FIRST]],
             cart[singleton[0], singleton[0]]]]], id[omega]] == True
```

This justifies adding a rewrite rule:

```
In[25]:= composite[FIRST, iterate[intersection[
             composite[inverse[SECOND], NATADD], composite[inverse[FIRST], SUCC, FIRST]],
            cart[singleton[0], singleton[0]]]]] := id[omega]
```

This verifies the conjecture:

```
In[26]:= composite[FIRST, ITER]

Out[26]= id[omega]
```

■ composite[SECOND,ITER]

The function **composite[SECOND,ITER]** adds up the sum of the first **n** natural numbers, yielding **0 + 1 + ... + (n−1)= n(n+1)/2**. This is easily verified for the first few numbers:

```
In[27]:= Map[APPLY[composite[SECOND, ITER], #] &, NestList[succ, 0, 4]]

Out[27]= {0, 0, singleton[0], succ[succ[singleton[0]]],
          succ[succ[succ[succ[succ[singleton[0]]]]]]}
```

The name **PARABOLA** is suggested by these results:

```
In[28]:= composite[SECOND, ITER] == PARABOLA

Out[28]= composite[SECOND, iterate[intersection[
              composite[inverse[SECOND], NATADD], composite[inverse[FIRST], SUCC, FIRST]],
            cart[singleton[0], singleton[0]]]] == PARABOLA
```

This definition is made into a rewrite rule:

```
In[29]:= composite[SECOND, iterate[intersection[
              composite[inverse[SECOND], NATADD], composite[inverse[FIRST], SUCC, FIRST]],
            cart[singleton[0], singleton[0]]]] := PARABOLA
```

That is:

```
In[30]:= composite[SECOND, ITER]

Out[30]= PARABOLA
```

The class **PARABOLA** is a function:

```
In[31]:= SubstTest[FUNCTION, composite[SECOND, iterate[funpart[x], singleton[y]]],
              {x -> composite[cross[composite[SUCC, FIRST], NATADD], DUP], y -> PAIR[0, 0]}]

Out[31]= FUNCTION[PARABOLA] == True

In[32]:= FUNCTION[PARABOLA] := True
```

Some elementary properties of this function are readily obtained:

```
In[33]:= Assoc[SECOND, ITER, id[omega]] // Reverse

Out[33]= composite[PARABOLA, id[omega]] == PARABOLA

In[34]:= composite[PARABOLA, id[omega]] := PARABOLA

In[35]:= ImageComp[SECOND, ITER, singleton[0]]

Out[35]= image[PARABOLA, singleton[0]] == singleton[0]

In[36]:= image[PARABOLA, singleton[0]] := singleton[0]
```

■ a recursion relation for PARABOLA

The following fact is needed to derive a recursion relation for **PARABOLA**:

```
In[37]:= Assoc[SECOND, ITER, SUCC]

Out[37]= composite[NATADD, iterate[intersection[
             composite[inverse[SECOND], NATADD], composite[inverse[FIRST], SUCC, FIRST]],
           cart[singleton[0], singleton[0]]]] == composite[PARABOLA, SUCC]

In[38]:= composite[NATADD, iterate[intersection[
             composite[inverse[SECOND], NATADD], composite[inverse[FIRST], SUCC, FIRST]],
           cart[singleton[0], singleton[0]]]] := composite[PARABOLA, SUCC]
```

The sequence **ITER** of ordered pairs is determined by the sequences of its first and second coordinates:

```
In[39]:= Assoc[cross[FIRST, SECOND], cross[ITER, ITER], DUP]

Out[39]= iterate[intersection[composite[inverse[SECOND], NATADD],
           composite[inverse[FIRST], SUCC, FIRST]], cart[singleton[0], singleton[0]]] ==
         composite[id[PARABOLA], inverse[FIRST]]

In[40]:= iterate[intersection[composite[inverse[SECOND], NATADD],
           composite[inverse[FIRST], SUCC, FIRST]], cart[singleton[0], singleton[0]]] :=
         composite[id[PARABOLA], inverse[FIRST]]
```

This formula expresses **ITER** in terms of **PARABOLA**.

```
In[41]:= ITER

Out[41]= composite[id[PARABOLA], inverse[FIRST]]
```

From the recursion for relation for **ITER** one deduces one for **PARABOLA**:

```
In[42]:= Map[composite[SECOND, #] &, SubstTest[composite, iterate[x, y], SUCC,
             {x -> composite[cross[composite[SUCC, FIRST], NATADD], DUP],
           y -> id[singleton[0]]}]]

Out[42]= composite[PARABOLA, SUCC] == composite[NATADD, id[PARABOLA], inverse[FIRST]]

In[43]:= composite[PARABOLA, SUCC] := composite[NATADD, id[PARABOLA], inverse[FIRST]]
```

■ illustration of the recursion relation for PARABOLA

To illustrate that this recursion relation suffices to compute the values of the **PARABOLA** function, some examples are provided:

```
In[44]:= ImageComp[PARABOLA, SUCC, singleton[0]] // Reverse

Out[44]= image[PARABOLA, singleton[singleton[0]]] == singleton[0]

In[45]:= image[PARABOLA, singleton[singleton[0]]] := singleton[0]

In[46]:= ImageComp[PARABOLA, SUCC, singleton[singleton[0]]] // Reverse

Out[46]= image[PARABOLA, singleton[succ[singleton[0]]]] == singleton[singleton[0]]

In[47]:= image[PARABOLA, singleton[succ[singleton[0]]]] := singleton[singleton[0]]
```

```
In[48]:= ImageComp[PARABOLA, SUCC, singleton[succ[singleton[0]]]] // Reverse

Out[48]= image[PARABOLA, singleton[succ[succ[singleton[0]]]]] ==
          singleton[succ[succ[singleton[0]]]]

In[49]:= image[PARABOLA, singleton[succ[succ[singleton[0]]]]] :=
          singleton[succ[succ[singleton[0]]]]

In[50]:= ImageComp[PARABOLA, SUCC, singleton[succ[succ[singleton[0]]]]] // Reverse

Out[50]= image[PARABOLA, singleton[succ[succ[succ[singleton[0]]]]]] ==
          singleton[succ[succ[succ[succ[succ[singleton[0]]]]]]]

In[51]:= image[PARABOLA, singleton[succ[succ[succ[singleton[0]]]]]] :=
          singleton[succ[succ[succ[succ[succ[singleton[0]]]]]]]
```

And so on... Summarizing:

```
In[52]:= Map[APPLY[PARABOLA, #] &, NestList[succ, 0, 4]]

Out[52]= {0, 0, singleton[0], succ[succ[singleton[0]]],
          succ[succ[succ[succ[succ[singleton[0]]]]]]}
```

■ the range of PARABOLA

The following lemma is needed:

```
In[53]:= ImageComp[PARABOLA, id[omega], x] // Reverse

Out[53]= image[PARABOLA, intersection[omega, x]] == image[PARABOLA, x]

In[54]:= image[PARABOLA, intersection[omega, x_]] := image[PARABOLA, x]
```

From the recursion relation for **PARABOLA** one finds:

```
In[55]:= Map[subclass[#, omega] &, ImageComp[PARABOLA, composite[id[omega], SUCC], V]] // Reverse

Out[55]= subclass[image[PARABOLA, complement[singleton[0]]], omega] == True

In[56]:= subclass[image[PARABOLA, complement[singleton[0]]], omega] := True
```

The value at **0** is also a natural number, so:

```
In[57]:= Map[subclass[#, omega] &, SubstTest[image, PARABOLA,
            union[x, y], {x -> singleton[0], y -> complement[singleton[0]]}]]

Out[57]= subclass[range[PARABOLA], omega] == True

In[58]:= subclass[range[PARABOLA], omega] := True
```

This fact is a corollary:

```
In[59]:= equal[composite[id[omega], PARABOLA], PARABOLA]

Out[59]= True
```

One is therefore justified in adding the following rewrite rule:

```
In[60]:= composite[id[omega], PARABOLA] := PARABOLA
```

■ the domain of PARABOLA

The domain of **PARABOLA** is contained in the set **omega** of all natural numbers:

```
In[61]:= Map[subclass[#, omega] &, IminComp[PARABOLA, id[omega], V]]

Out[61]= subclass[domain[PARABOLA], omega] == True

In[62]:= subclass[domain[PARABOLA], omega] := True
```

It will now be shown that the recursion relation implies that **PARABOLA** is defined for all natural numbers.

```
In[63]:= Map[subclass[domain[PARABOLA], #] &, IminComp[PARABOLA, SUCC, V]] // Reverse

Out[63]= subclass[image[SUCC, domain[PARABOLA]], domain[PARABOLA]] == True

In[64]:= subclass[image[SUCC, domain[PARABOLA]], domain[PARABOLA]] := True

In[65]:= Map[not, SubstTest[equal, 0, image[x, singleton[y]], {x -> PARABOLA, y -> 0}]] // Reverse

Out[65]= member[0, domain[PARABOLA]] == True

In[66]:= member[0, domain[PARABOLA]] := True
```

The key step is to use mathematical induction.

```
In[67]:= INDUCTIVE[x]

Out[67]= and[member[0, x], subclass[image[SUCC, x], x]]

In[68]:= SubstTest[implies, INDUCTIVE[x], subclass[omega, x], x -> domain[PARABOLA]]

Out[68]= subclass[omega, domain[PARABOLA]] == True

In[69]:= subclass[omega, domain[PARABOLA]] := True
```

Since the inclusion holds in both directions, one obtains an equation:

```
In[70]:= SubstTest[and, subclass[u, v], subclass[v, u], {u -> domain[PARABOLA], v -> omega}]

Out[70]= True == equal[omega, domain[PARABOLA]]

In[71]:= domain[PARABOLA] := omega
```

■ the function PARABOLA maps omega into omega

The function **PARABOLA** is a set.

```
In[72]:= SubstTest[implies, and[subclass[x, y], member[y, V]], member[x, V],
           {x -> PARABOLA, y -> cart[omega, omega]}]

Out[72]= member[PARABOLA, V] == True

In[73]:= member[PARABOLA, V] := True
```

It follows that **PARABOLA** is a mapping from **omega** into **omega**.

```
In[74]:= member[PARABOLA, map[omega, omega]]

Out[74]= True
```