a parabolic function

Johan G. F. Belinfante
2003 May 17

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

It is now: 2003 May 22 at 11:12

Loading Simplification Rules

weightlimit = 40


\section*{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], \ldots\} there is a corresponding sequence of partial sums, \{0, n[0], n[0]+n[1], n[0]+n[1]+n[2], \ldots\}. In particular, for the special case of the sequence of natural numbers \text{id}[\omega] = \{0, 1, 2, 3, \ldots\}, the \textit{n}th partial sum is a parabolic function of \textit{n}. It is shown how this function \texttt{PARABOLA} can be defined using iteration. From the definition of \texttt{PARABOLA} in terms of \texttt{iterate}, some simple properties of this function are derived, including a recursion relation for \texttt{PARABOLA}.

\section*{reduction to an iteration problem}

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

\begin{verbatim}
In[2]:= singleton[pair[0, 0]] == id[singleton[0]]
\end{verbatim}

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

\begin{verbatim}
In[3]:= APPLY[composite[cross[composite[SUCC, FIRST], NATADD], DUP], PAIR[x, y]]
Out[3]= PAIR[succ[x], natadd[x, y]]
\end{verbatim}

The sequence of ordered pairs is therefore given by the following expression for which the temporary abbreviation \texttt{ITER} is introduced.

\begin{verbatim}
In[4]:= ITER := iterate[composite[cross[composite[SUCC, FIRST], NATADD], DUP], id[singleton[0]]]
\end{verbatim}
The basic strategy is to first derive a formula expressing this sequence in terms of the sequence \textsc{parabola} of partial sums of the sequence of natural numbers, and then to turn around and use that formula to derive the properties of \textsc{parabola}.

\textbf{footnote: a comment about PAIR[x,y] versus pair[x,y]}

The \textsc{goedel} program actually has two ordered pairs, \textsc{pair[x,y]} and \textsc{PAIR[x,y]}, the latter defined by:

\begin{verbatim}
In[5]:= A[cart[singleton[x], singleton[y]]]
Out[5]= PAIR[x, y]
\end{verbatim}

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

\textbf{some properties of ITER}

The class \textsc{iter} is a function:

\begin{verbatim}
In[6]:= SubstTest[FUNCTION, iterate[funpart[s], singleton[t]],
{\textbf{s} \to \textbf{composite}[\textbf{cross}[\textbf{composite}[\textbf{SUCC}, \textbf{FIRST}], \textbf{NATADD}], \textbf{DUP}], \textbf{t} \to \textbf{PAIR}[0, 0]}]
\textbf{composite}[\textbf{inverse}[\textbf{FIRST}], \textbf{SUCC}, \textbf{FIRST}], \textbf{cart}[\textbf{singleton}[0], \textbf{singleton}[0]]]] == True
In[7]:= FUNCTION[iterate[intersection[\textbf{composite}[\textbf{inverse}[\textbf{SECOND}], \textbf{NATADD}],
\textbf{composite}[\textbf{inverse}[\textbf{FIRST}], \textbf{SUCC}, \textbf{FIRST}], \textbf{cart}[\textbf{singleton}[0], \textbf{singleton}[0]]]] := True
\end{verbatim}

This can be rewritten as:

\begin{verbatim}
In[8]:= \textsc{FUNCTION}[\textsc{iter}]
Out[8]= True
\end{verbatim}

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

\begin{verbatim}
In[9]:= SubstTest[subclass, range[iterate[s, t]], union[range[s], t],
{\textbf{s} \to \textbf{composite}[\textbf{cross}[\textbf{composite}[\textbf{SUCC}, \textbf{FIRST}], \textbf{NATADD}], \textbf{DUP}], \textbf{t} \to \textbf{id}[\textbf{singleton}[0]]}]
\textbf{composite}[\textbf{inverse}[\textbf{FIRST}], \textbf{SUCC}, \textbf{FIRST}], \textbf{cart}[\textbf{singleton}[0], \textbf{singleton}[0]]]],
\textbf{union}[\textbf{cart}[\textbf{singleton}[0], \textbf{singleton}[0]], \textbf{composite}[\textbf{id}[\textbf{omega}],
\textbf{S}, \textbf{inverse}[\textbf{SUCC}], \textbf{id}[\textbf{omega}]]]] == True
In[10]:= subclass[range[iterate[intersection[\textbf{composite}[\textbf{inverse}[\textbf{SECOND}], \textbf{NATADD}],
\textbf{composite}[\textbf{inverse}[\textbf{FIRST}], \textbf{SUCC}, \textbf{FIRST}], \textbf{cart}[\textbf{singleton}[0], \textbf{singleton}[0]]]],
\textbf{union}[\textbf{cart}[\textbf{singleton}[0], \textbf{singleton}[0]], \textbf{composite}[\textbf{id}[\textbf{omega}],
\textbf{S}, \textbf{inverse}[\textbf{SUCC}], \textbf{id}[\textbf{omega}]]]] := True
\end{verbatim}

The second application just uses the transitive property of \textsc{subclass}.
In[11]:=  SubstTest[implies, andsubclass[x, s], subclass[s, t]], subclass[x, t],
  \{x -> range[ITER], s -> union[cart[singleton[0], singleton[0]],
  composite[id[omega], S, inverse[SUCC], id[omega]]], t -> cart[omega, omega]\]

  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]]


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
\section*{composite[FIRST,ITER]}

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

\begin{verbatim}
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]]]]\}
\end{verbatim}

This observation suggests the conjecture:

\begin{verbatim}
In[22]:= composite[FIRST, ITER] == id[omega];
\end{verbatim}

This conjecture will now be proved by using iteration uniqueness:

\begin{verbatim}
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
\end{verbatim}

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

\begin{verbatim}
In[24]:= SubstTest[implies, and[equal[composite[s, r], composite[r, SUCC]],
  equal[image[r, singleton[0]], t]],
  {r -> composite[FIRST, ITER], s -> SUCC, t -> singleton[0]}]
  composite[inverse[SECOND], NATADD], composite[inverse[FIRST], SUCC, FIRST]],
  cart[singleton[0], singleton[0]]], id[omega]] == True
\end{verbatim}

This justifies adding a rewrite rule:

\begin{verbatim}
In[25]:= composite[FIRST, iterate[intersection[
  composite[inverse[SECOND], NATADD], composite[inverse[FIRST], SUCC, FIRST]],
  cart[singleton[0], singleton[0]]]] := id[omega]
\end{verbatim}

This verifies the conjecture:

\begin{verbatim}
In[26]:= composite[FIRST, ITER]
Out[26]= id[omega]
\end{verbatim}

\section*{composite[SECOND,ITER]}

The function \texttt{composite[SECOND,ITER]} adds up the sum of the first \( n \) natural numbers, yielding \( 0 + 1 + \ldots + (n-1) = n(n+1)/2 \). This is easily verified for the first few numbers:
The name **PARABOLA** is suggested by these results:

\[\text{composite[SECOND, ITER]} \equiv \text{PARABOLA}\]

This definition is made into a rewrite rule:

\[\text{composite[SECOND, iterate[intersection[ composite[inverse[SECOND]], \text{NATADD}], composite[inverse[FIRST], \text{SUCC}, \text{FIRST}], cart[\text{singleton}[0], \text{singleton}[0]]]} \equiv \text{PARABOLA}\]

That is:

\[\text{composite[SECOND, ITER]} \equiv \text{PARABOLA}\]

The class **PARABOLA** is a function:

\[\text{SubstTest[FUNCTION, composite[SECOND, iterate[funpart[x], \text{singleton}[y]]], }\{x \mapsto \text{composite[cross[composite[SUCC, FIRST], \text{NATADD}], DUP], y \mapsto \text{PAIR}[0, 0]}\}\]

Some elementary properties of this function are readily obtained:

\[\text{Assoc[SECOND, ITER, \text{id}[\omega]]} // \text{Reverse}\]

\[\text{composite[PARABOLA, \text{id}[\omega]]} \equiv \text{PARABOLA}\]

\[\text{composite[PARABOLA, \text{id}[\omega]]} \equiv \text{PARABOLA}\]

\[\text{ImageComp[SECOND, ITER, \text{singleton}[0]]}\]

\[\text{image[PARABOLA, \text{singleton}[0]]} \equiv \text{singleton}[0]\]

\[\text{image[PARABOLA, \text{singleton}[0]]} \equiv \text{singleton}[0]\]

**a recursion relation for PARABOLA**

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

\[\text{a recursion relation for PARABOLA}\]

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

\[\text{a recursion relation for PARABOLA}\]

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

\[\text{a recursion relation for PARABOLA}\]
The sequence \( \text{ITER} \) of ordered pairs is determined by the sequences of its first and second coordinates:

\[
\text{In[39]} := \text{Assoc}[\text{cross}[\text{FIRST}, \text{SECOND}], \text{cross}[\text{ITER}, \text{ITER}], \text{DUP}]
\]

\[
\text{Out[39]} = \text{iterate}[\text{intersection}[\text{composite}[\text{inverse}[\text{SECOND}], \text{NATADD}], \text{composite}[\text{inverse}[\text{FIRST}], \text{SUCC}, \text{FIRST}]], \text{cart}[\text{singleton}[0], \text{singleton}[0]]] ::= \text{composite}[\text{PARABOLA}, \text{SUCC}]
\]

This formula expresses \( \text{ITER} \) in terms of \( \text{PARABOLA} \).

\[
\text{In[41]} := \text{ITER}
\]

\[
\text{Out[41]} = \text{composite}[\text{id}[\text{PARABOLA}], \text{inverse}[\text{FIRST}]]
\]

From the recursion for relation for \( \text{ITER} \) one deduces one for \( \text{PARABOLA} \):

\[
\text{In[43]} := \text{Map}[\text{composite}[\text{SECOND}, #] & & , \text{SubstTest}[\text{composite}, \text{iterate}[x, y], \text{SUCC},
\{x \rightarrow \text{composite}[\text{cross}[\text{composite}[\text{SUCC}, \text{FIRST}], \text{NATADD}], \text{DUP}],
\ y \rightarrow \text{id}[\text{singleton}[0]]\}]]
\]

\[
\text{Out[42]} = \text{composite}[\text{PARABOLA}, \text{SUCC}] ::= \text{composite}[\text{NATADD}, \text{id}[\text{PARABOLA}], \text{inverse}[\text{FIRST}]]
\]

\[
\text{In[43]} := \text{composite}[\text{PARABOLA}, \text{SUCC}] ::= \text{composite}[\text{NATADD}, \text{id}[\text{PARABOLA}], \text{inverse}[\text{FIRST}]]
\]

## Illustration of the recursion relation for \( \text{PARABOLA} \)

To illustrate that this recursion relation suffices to compute the values of the \( \text{PARABOLA} \) function, some examples are provided:

\[
\text{In[44]} := \text{ImageComp}[\text{PARABOLA}, \text{SUCC}, \text{singleton}[0]] // \text{Reverse}
\]

\[
\text{Out[44]} = \text{image}[\text{PARABOLA}, \text{singleton}[\text{singleton}[0]]] ::= \text{singleton}[0]
\]

\[
\text{In[45]} := \text{image}[\text{PARABOLA}, \text{singleton}[\text{singleton}[0]]] ::= \text{singleton}[0]
\]

\[
\text{In[46]} := \text{ImageComp}[\text{PARABOLA}, \text{SUCC}, \text{singleton}[\text{singleton}[0]]] // \text{Reverse}
\]

\[
\text{Out[46]} = \text{image}[\text{PARABOLA}, \text{singleton}[\text{succ}[\text{singleton}[0]]]] ::= \text{singleton}[\text{singleton}[0]]
\]

\[
\text{In[47]} := \text{image}[\text{PARABOLA}, \text{singleton}[\text{succ}[\text{singleton}[0]]]] ::= \text{singleton}[\text{singleton}[0]]
\]
The range of PARABOLA

The following lemma is needed:

From the recursion relation for PARABOLA one finds:

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

This fact is a corollary:
One is therefore justified in adding the following rewrite rule:

```math
In[60]:= \text{composite}[\text{id}[\omega], \text{PARABOLA}] := \text{PARABOLA}
```

### the domain of PARABOLA

The domain of \text{PARABOLA} is contained in the set \omega of all natural numbers:

```math
In[61]:= \text{Map}[\text{subclass}[\#, \omega] \&., \text{InminComp}[\text{PARABOLA}, \text{id}[\omega], \nu]]
Out[61]= \text{subclass}[\text{domain}[\text{PARABOLA}], \omega] == \text{True}
```

```math
In[62]:= \text{subclass}[\text{domain}[\text{PARABOLA}], \omega] := \text{True}
```

It will now be shown that the recursion relation implies that \text{PARABOLA} is defined for all natural numbers.

```math
In[63]:= \text{Map}[\text{subclass}[\text{domain}[\text{PARABOLA}], \#] \&., \text{InminComp}[\text{PARABOLA}, \text{Succ}, \nu]] // \text{Reverse}
Out[63]= \text{subclass}[\text{image}[\text{Succ}, \text{domain}[\text{PARABOLA}]], \text{domain}[\text{PARABOLA}]] == \text{True}
```

```math
In[64]:= \text{subclass}[\text{image}[\text{Succ}, \text{domain}[\text{PARABOLA}]], \text{domain}[\text{PARABOLA}]] := \text{True}
```

```math
In[65]:= \text{Map}[\text{not}, \text{SubstTest}[\text{equal}, 0, \text{image}[x, \text{singleton}[y]], \{x \rightarrow \text{PARABOLA}, y \rightarrow 0\}] ] // \text{Reverse}
Out[65]= \text{member}[0, \text{domain}[\text{PARABOLA}]] == \text{True}
```

```math
In[66]:= \text{member}[0, \text{domain}[\text{PARABOLA}]] := \text{True}
```

The key step is to use mathematical induction.

```math
In[67]:= \text{INDUCTIVE}[x]
Out[67]= \text{and}[\text{member}[0, x], \text{subclass}[\text{image}[\text{Succ}, x], x]]
```

```math
In[68]:= \text{SubstTest}[\text{implies}, \text{INDUCTIVE}[x], \text{subclass}[\omega, x], x \rightarrow \text{domain}[\text{PARABOLA}]]
Out[68]= \text{subclass}[\omega, \text{domain}[\text{PARABOLA}]] == \text{True}
```

```math
In[69]:= \text{subclass}[\omega, \text{domain}[\text{PARABOLA}]] := \text{True}
```

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

```math
In[70]:= \text{SubstTest}[\text{and}, \text{subclass}[u, v], \text{subclass}[v, u], \{u \rightarrow \text{domain}[\text{PARABOLA}], v \rightarrow \omega\}]
Out[70]= \text{True} == \text{equal}[\omega, \text{domain}[\text{PARABOLA}]]
```

```math
In[71]:= \text{domain}[\text{PARABOLA}] := \omega
```

### the function PARABOLA maps omega into omega

The function \text{PARABOLA} is a set.
\text{In[72]}:= \text{SubstTest[implies, and[subclass[x, y], member[y, V]], member[x, V],}
\{x \rightarrow \text{PARABOLA}, y \rightarrow \text{cart[omega, omega]}\])
\text{Out[72]}= \text{member[PARABOLA, V] == True}
\text{In[73]}:= \text{member[PARABOLA, V] := True}

It follows that \text{PARABOLA} is a mapping from \text{omega} into \text{omega}.

\text{In[74]}:= \text{member[PARABOLA, map[omega, omega]]}
\text{Out[74]}= \text{True}