rec.1 Sequences

The set of primitive recursive functions is remarkably robust. But we will be able to do even more once we have developed a adequate means of handling sequences. We will identify finite sequences of natural numbers with natural numbers in the following way: the sequence \( \langle a_0, a_1, a_2, \ldots, a_k \rangle \) corresponds to the number

\[
p_0^{a_0+1} \cdot p_1^{a_1+1} \cdot p_2^{a_2+1} \cdots p_k^{a_k+1}.
\]

We add one to the exponents to guarantee that, for example, the sequences \( \langle 2, 7, 3 \rangle \) and \( \langle 2, 7, 3, 0, 0 \rangle \) have distinct numeric codes. We can take both 0 and 1 to code the empty sequence; for concreteness, let \( \Lambda \) denote 0.

The reason that this coding of sequences works is the so-called Fundamental Theorem of Arithmetic: every natural number \( n \geq 2 \) can be written in one and only one way in the form

\[
n = p_0^{a_0} \cdot p_1^{a_1} \cdots p_k^{a_k}
\]

with \( a_k \geq 1 \). This guarantees that the mapping \( \langle a_0, \ldots, a_k \rangle = \langle a_0, \ldots, a_k \rangle \) is injective: different sequences are mapped to different numbers; to each number only at most one sequence corresponds.

We’ll now show that the operations of determining the length of a sequence, determining its \( i \)th element, appending an element to a sequence, and concatenating two sequences, are all primitive recursive.

**Proposition rec.1.** The function \( \text{len}(s) \), which returns the length of the sequence \( s \), is primitive recursive.

**Proof.** Let \( R(i, s) \) be the relation defined by

\[
R(i, s) \text{ iff } p_i \mid s \land p_{i+1} \nmid s.
\]

\( R \) is clearly primitive recursive. Whenever \( s \) is the code of a non-empty sequence, i.e.,

\[
s = p_0^{a_0+1} \cdots p_k^{a_k+1},
\]

\( R(i, s) \) holds if \( p_i \) is the largest prime such that \( p_i \mid s \), i.e., \( i = k \). The length of \( s \) thus is \( i + 1 \) iff \( p_i \) is the largest prime that divides \( s \), so we can let

\[
\text{len}(s) = \begin{cases} 
0 & \text{if } s = 0 \text{ or } s = 1 \\
1 + (\min i < s) \ R(i, s) & \text{otherwise}
\end{cases}
\]

We can use bounded minimization, since there is only one \( i \) that satisfies \( R(s, i) \) when \( s \) is a code of a sequence, and if \( i \) exists it is less than \( s \) itself. \( \square \)

**Proposition rec.2.** The function \( \text{append}(s, a) \), which returns the result of appending \( a \) to the sequence \( s \), is primitive recursive.
Proof. append can be defined by:

\[
\text{append}(s, a) = \begin{cases} 
2a + 1 & \text{if } s = 0 \text{ or } s = 1 \\
sp_{\text{len}(s)} & \text{otherwise.}
\end{cases}
\]

Proposition rec.3. The function \(\text{element}(s, i)\), which returns the \(i\)th element of \(s\) (where the initial element is called the 0th), or 0 if \(i\) is greater than or equal to the length of \(s\), is primitive recursive.

Proof. Note that \(a\) is the \(i\)th element of \(s\) iff \(p_i^{a+1} \mid s\) but \(p_i^{a+2} \nmid s\). So:

\[
\text{element}(s, i) = \begin{cases} 
0 & \text{if } i \geq \text{len}(s) \\
\text{(min } a < s \text{) } (p_i^{a+2} \nmid s) & \text{otherwise.}
\end{cases}
\]

Instead of using the official names for the functions defined above, we introduce a more compact notation. We will use \((s)_i\) instead of \(\text{element}(s, i)\), and \(<s_0, \ldots, s_k>\) to abbreviate \(\text{append}(\ldots \text{append}(A, s_0)\ldots), s_k)\).

Note that if \(s\) has length \(k\), the elements of \(s\) are \((s)_0, \ldots, (s)_{k-1}\).

Proposition rec.4. The function \(\text{concat}(s, t)\), which concatenates two sequences, is primitive recursive.

Proof. We want a function \(\text{concat}\) with the property that

\[
\text{concat}(<a_0, \ldots, a_k>, <b_0, \ldots, b_l>) = <a_0, \ldots, a_k, b_0, \ldots, b_l>.
\]

We’ll use a “helper” function \(\text{hconcat}(s, t, n)\) which concatenates the first \(n\) symbols of \(t\) to \(s\). This function can be defined by primitive recursion as follows:

\[
\text{hconcat}(s, t, 0) = s \\
\text{hconcat}(s, t, n + 1) = \text{append}(\text{hconcat}(s, t, n), (t)_n)
\]

Then we can define \(\text{concat}\) by

\[
\text{concat}(s, t) = \text{hconcat}(s, t, \text{len}(t)).
\]
most $k$ prime factors, each at most $p_{k-1}$, and each raised to at most $x + 1$ in
the prime factorization of $s$. In other words, if we define
\[
\text{sequenceBound}(x, k) = p_{k-1}^{(x+1)},
\]
then the numeric code of the sequence $s$ described above is at most sequenceBound($x, k$).

Having such a bound on sequences gives us a way of defining new functions
using bounded search. For example, we can define concat using bounded search.
All we need to do is write down a primitive recursive \emph{specification} of the object
(number of the concatenated sequence) we are looking for, and a bound on how
far to look. The following works:
\[
\text{concat}(s, t) = (\min v < \text{sequenceBound}(s + t, \text{len}(s) + \text{len}(t)))
\]
\[
(\text{len}(v) = \text{len}(s) + \text{len}(t) \land
(\forall i < \text{len}(s)) \ ((v)_i = (s)_i) \land
(\forall j < \text{len}(t)) \ ((v)_{\text{len}(s)+j} = (t)_j))
\]

**Problem rec.1.** Show that there is a primitive recursive function $\text{sconcat}(s)$
with the property that
\[
\text{sconcat}((s_0, \ldots, s_k)) = s_0 \bowtie \ldots \bowtie s_k.
\]

**Problem rec.2.** Show that there is a primitive recursive function $\text{tail}(s)$ with
the property that
\[
\text{tail}(A) = 0 \text{ and} \text{tail}((s_0, \ldots, s_k)) = (s_1, \ldots, s_k).
\]

**Proposition rec.5.** The function $\text{subseq}(s, i, n)$ which returns the subsequence
of $s$ of length $n$ beginning at the $i$th element, is primitive recursive.

**Proof.** Exercise. \hfill \square

**Problem rec.3.** Prove Proposition rec.5.

---

**Photo Credits**

**Bibliography**