Chapter 20

Case study: A paragraph problem

20.1 Even paragraphs

This case study, the first of three major studies with which we conclude, is based on [Bir86]; like the two to follow, it is quite a lot more ambitious than our earlier examples.

The problem itself seems simple: it is just the laying out of words into lines and paragraphs. Compare the paragraphs of Figures 20.1 and 20.2. In *simple* paragraphs, like Figure 20.1, each line is filled as much as possible before moving on to the next. As a consequence, the minimum number of lines is used; but a long word arriving near the end of a line can cause a large gap there.

In *even* paragraphs, like Figure 20.2, such gaps are reduced: space is distributed over earlier lines in order to increase the length of a later line which would otherwise be very short. We will develop a program that produces even paragraphs.

To start, we forget the actual words and just consider their lengths. Let the sequence of word lengths to be laid out be ws, of type $\mathbf{seq}_N\,\mathbb{N}$. We have a maximum line width of M (characters), and we assume that all (word) lengths are non-zero and no more than M:

$$(\forall\, w : ws \cdot 0 < w \le M)\ .$$

A *paragraph* of ws is a sequence of lines, and each line is a sequence of words. For paragraph $pss : \mathbf{seq}\,\mathbf{seq}\,\mathbb{N}$ (remember we are considering only lengths, not actual words), we have these conditions:

1. The paragraph pss contains exactly the lengths in ws, in their original order: $\mathsf{fl}\,pss = ws$. The function fl, *flatten*, is defined as follows:

   $$\mathsf{fl}\,pss \;\widehat{=}\; (+\!\!+\, ls : pss)\ .$$

2. Each line of the paragraph contains at least one but no more than M characters: $(\forall\, ls : pss \cdot 0 < \sum ls \le M)$. We abbreviate that $\mathsf{ok}\,pss$. The function

```
⊣Compare the paragraphs of Figure 20.1 and    ⊢
⊣Figure 20.2. In simple paragraphs, like Figure⊢
⊣20.1, each line is filled as much as possible ⊢
⊣before moving on to the next. As a           ⊢
⊣consequence, the minimum number of lines is   ⊢
⊣used; but a long word arriving near the end of⊢
⊣a line can cause a large gap there.           ⊢
```

Figure 20.1 Simple paragraph

```
⊣Compare the paragraphs of Figure 20.1 and    ⊢
⊣Figure 20.2. In simple paragraphs, like       ⊢
⊣Figure 20.1, each line is filled as much as   ⊢
⊣possible before moving on to the next. As a   ⊢
⊣consequence, the minimum number of lines is   ⊢
⊣used; but a long word arriving near the end of⊢
⊣a line can cause a large gap there.           ⊢
```

Figure 20.2 Even paragraph

$\sum$ is defined as follows:

$$\sum ls \;\widehat{=}\; (+w : ls) \; .$$

Suppose for example that $ws = \langle 2, 1, 1, 3 \rangle$, and $M = 3$. Here are three paragraphs of ws:

$$\begin{array}{rl} \langle\langle 2, 1\rangle, \langle 1\rangle, \langle 3\rangle\rangle & \text{(simple)} \\ \langle\langle 2\rangle, \langle 1, 1\rangle, \langle 3\rangle\rangle & \text{(even)} \\ \langle\langle 2\rangle, \langle 1\rangle, \langle 1\rangle, \langle 3\rangle\rangle & \text{(neither simple nor even).} \end{array}$$

Figures 20.1 and 20.2 are paragraphs of the same words. But Figure 20.2 minimises the waste of the paragraph, where the *waste* is the size of the largest gap left in any of its lines except the last:

$$\mathsf{wt}\, pss \;\widehat{=}\; (\sqcup ls : \mathsf{fr}\, pss \cdot M - \sum ls) \; .$$

(Recall that fr takes the front of a sequence.) Now the *minimum waste* mw of a sequence of word lengths is the least waste found in any of its paragraphs:

$$\mathsf{mw}\, ws \;\widehat{=}\; (\sqcap pss \mid \mathsf{fl}\, pss = ws \wedge \mathsf{ok}\, pss \cdot \mathsf{wt}\, pss) \; .$$

The paragraph of Figure 20.1 has waste 12; the waste of Figure 20.2 is only 7, which is minimal in that width for those words.

Our first step is to derive a program that calculates $\mathsf{mw}\, ws$.

20.2 The minimum waste

Here is the abstract program that calculates the waste that an even paragraph would have.

$$
\begin{array}{l}
\textbf{var } ws : \mathbf{seq}_N\,\mathbb{N};\\
\quad mw, M : \mathbb{N};\\[1em]
\textbf{and } ws \neq \langle\rangle \wedge (\forall\, w : ws \cdot 0 < w \leq M) \cdot\\[1em]
mw := \mathsf{mw}\; ws\ .
\end{array}
$$

The invariant expresses the conditions on ws and M: that ws is non-empty, no word length in ws is 0, and no word length in ws exceeds M.

Rather than consider larger and larger prefixes of ws, we consider suffixes. That is because the last line is treated specially: it does not contribute to the waste. So we introduce a sequence sf (*suffixes*) to contain the minimum wastes of all suffixes of ws; the needed value will be $sf[0]$ finally.

$$
\begin{array}{ll}
\sqsubseteq\ \textbf{var } sf : \mathbf{seq}_N\,\mathbb{N}\cdot & \\
\quad sf\colon [(\forall\, i \mid 0 \leq i < N \cdot sf[i] = \mathsf{mw}(ws{\downarrow}i))]\,; & \triangleleft\\
\quad mw := sf[0]\ . &
\end{array}
$$

The next few steps are the usual ones for developing a iteration. Note however that the initialisation is $j := N - 1$; that is because $\mathsf{mw}\ \langle\rangle$ is not defined.

$$
\begin{array}{ll}
\sqsubseteq\ I \ \widehat{=}\ (\forall\, i \mid j \leq i < N \cdot sf[i] = \mathsf{mw}(ws{\downarrow}i)) \wedge j < N \cdot & \\
\quad sf\colon [I[j\backslash 0]] & \\
\sqsubseteq\ \textbf{var } j : \mathbb{N}\cdot & \\
\quad j, sf[N-1] := N-1, 0; & \\
\quad j, sf\colon [j = N-1\ ,\ I\ ,\ j = 0] & \triangleleft\\
\sqsubseteq\ \text{“iterate down”} & \\
\quad \textbf{do } j \neq 0 \rightarrow & \\
\qquad j := j - 1; & \\
\qquad sf\colon [I[j\backslash j+1]\ ,\ I] & \text{(i)}\\
\quad \textbf{od}\ . &
\end{array}
$$

Now we must change sf, but it is clear that we need change it only at index j. So we introduce a new variable x to be finally assigned to $sf[j]$. That allows the first command below to leave sf out of the frame.

$$
\begin{array}{ll}
\text{(i)}\ \sqsubseteq\ \textbf{var } x : \mathbb{N}\cdot & \\
\quad x\colon [I[j\backslash j+1]\ ,\ x = \mathsf{mw}(ws{\downarrow}j)]\,; & \text{(ii)}\\
\quad sf[j] := x\ . &
\end{array}
$$

To make progress now, we must look more closely at $\mathsf{mw}(ws{\downarrow}j)$. From the precondition we have $j+1<N$, and we proceed

$$
\begin{array}{ll}
 & \mathsf{mw}(ws{\downarrow}j) \\
= & (\sqcap pss \mid \mathsf{fl}\, pss = ws{\downarrow}j \wedge \mathsf{ok}\, pss \cdot \mathsf{wt}\, pss) \\
= & \text{“because } j<N\text{, and so } ws{\downarrow}j \text{ cannot be empty”} \\
 & (\sqcap ls, pss' \mid \ \mathsf{fl}(\langle ls\rangle +\!\!+ pss') = ws{\downarrow}j \wedge \mathsf{ok}(\langle ls\rangle +\!\!+ pss') \\
 & \qquad\qquad \cdot\ \ \mathsf{wt}(\langle ls\rangle +\!\!+ pss'))\ .
\end{array}
$$

In the last step, we replaced the bound variable pss by a concatenation $\langle ls\rangle +\!\!+ pss'$, using as justification that pss was not empty. The step is valid because the non-empty lists pss and the pairs ls, pss' can be put into 1-to-1 correspondence.

To avoid proliferating names, we now rename pss' to pss again, and continue:

$$
\begin{array}{ll}
= & (\sqcap ls, pss \mid \ \mathsf{fl}(\langle ls\rangle +\!\!+ pss) = ws{\downarrow}j \wedge \mathsf{ok}(\langle ls\rangle +\!\!+ pss) \\
 & \qquad\qquad \cdot\ \ \mathsf{wt}(\langle ls\rangle +\!\!+ pss)) \\
= & \text{“by definition of } \mathsf{fl} \text{ and } \mathsf{ok}\text{”} \\
 & (\sqcap ls, pss \mid \left\{ \begin{array}{l} ls +\!\!+ \mathsf{fl}\, pss = ws{\downarrow}j \\ \mathsf{ok}\, pss \\ 0 < \sum ls \le M \end{array} \right. \quad \cdot\, \mathsf{wt}(\langle ls\rangle +\!\!+ pss))\ .
\end{array}
$$

Now we replace ls by its length k. Again, there is a 1-to-1 correspondence, since for any k, there is only one ls of length k satisfying $ls +\!\!+ \mathsf{fl}\, pss = ws{\downarrow}j$. That gives

$$
\begin{array}{ll}
= & \text{“replacing } ls \text{ by its length } k\text{”} \\
 & (\sqcap k, pss \mid \left\{ \begin{array}{l} 0 < k \le N-j \\ \mathsf{fl}\, pss = ws{\downarrow}(j+k) \\ \mathsf{ok}\, pss \\ \sum ws{\downarrow}j{\uparrow}k \le M \end{array} \right. \quad \cdot\, \mathsf{wt}(\langle ws{\downarrow}j{\uparrow}k\rangle +\!\!+ pss))\ .
\end{array}
$$

Now we will use the definition of wt, but it is defined only for non-empty sequences! That means the case $k = N-j$, which makes pss empty, must be handled carefully.

If $\sum ws{\downarrow}j \le M$, then k can take the value $N-j$ in the distributed minimum above: $\sum ws{\downarrow}j{\uparrow}k \le M$ will be true even when $k = N-j$. But then pss will be empty, and the waste $\mathsf{wt}(\langle ws{\downarrow}j{\uparrow}k\rangle +\!\!+ pss)$ will be 0. Since all wastes are at least 0, the whole expression simplifies to that value:

$$
\begin{array}{ll}
= & \textbf{Case } \sum ws{\downarrow}j \le M \\
 & 0\ .
\end{array}
$$

In the other case, we can of course exclude $k = N-j$ since it is dealt with in the first case. We have then, by definition of wt,

$$
\begin{array}{ll}
= & \textbf{Case } \sum ws{\downarrow}j > M \\
 & (\sqcap k, pss \mid \left\{ \begin{array}{l} 0 < k < N-j \\ \mathsf{fl}\, pss = ws{\downarrow}(j+k) \\ \mathsf{ok}\, pss \\ \sum ws{\downarrow}j{\uparrow}k \le M \end{array} \right. \quad \cdot \left\{ \begin{array}{ll} & M - \sum ws{\downarrow}j{\uparrow}k \\ \sqcup & \mathsf{wt}\, pss \end{array} \right.)
\end{array}
$$

$$
\begin{array}{ll}
= & \text{“nesting the minima”} \\
& (\sqcap k \mid 0 < k < N - j \wedge \sum ws{\downarrow}j{\uparrow}k \leq M \bullet \\
& \quad (\sqcap pss \mid \mathsf{fl}\, pss = ws{\downarrow}(j+k) \wedge \mathsf{ok}\, pss \bullet \\
& \quad\quad (M - \sum ws{\downarrow}j{\uparrow}k) \sqcup \mathsf{wt}\, pss)) \\
= & \text{“distributing } \sqcup \text{ out of } \sqcap \text{”} \\
& (\sqcap k \mid 0 < k < N - j \wedge \sum ws{\downarrow}j{\uparrow}k \leq M \bullet \\
& \quad \left. \begin{array}{ll} & M - \sum ws{\downarrow}j{\uparrow}k \\ \sqcup & (\sqcup pss \mid \mathsf{fl}\, pss = ws{\downarrow}(j+k) \wedge \mathsf{ok}\, pss \bullet \mathsf{wt}\, pss) \end{array} \right) \\
= & \text{“definition of } \mathsf{mw} \text{”} \\
& (\sqcap k \mid 0 < k < N - j \wedge \sum ws{\downarrow}j{\uparrow}k \leq M \bullet \\
& \quad (M - \sum ws{\downarrow}j{\uparrow}k) \sqcup \mathsf{mw}\; ws{\downarrow}(j+k)) \\
= & \text{“replace } k \text{ by } k - j \text{”} \\
& (\sqcap k \mid j < k < N \wedge \sum ws[j{\rightarrow}k] \leq M \bullet \\
& \quad (M - \sum ws[j{\rightarrow}k]) \sqcup \mathsf{mw}(ws{\downarrow}k)) \;.
\end{array}
$$

With the above, we have defined $\mathsf{mw}(ws{\downarrow}j)$ in terms of the minimum waste $\mathsf{mw}(ws{\downarrow}k)$ of shorter suffixes; we can now return to the development of the program. Since the range condition $\sum ws[j{\rightarrow}k] \leq M$ is less likely to be true for greater values of k, we start k at $j+1$ and increase it. In our invariant also will be the sum s of the segment $ws[j{\rightarrow}k]$ considered so far.

The case distinction will be made *after* we have calculated the minimum above, since then the sum $\sum ws{\downarrow}j$ will be available in s. So we continue

$$
\begin{array}{lll}
\text{(ii)} & \sqsubseteq & \mathbf{var}\; n, s : \mathbb{N} \bullet \\
& & x, n, s\colon [I[j \backslash j+1] \;,\; x = \mathsf{mw}(ws{\downarrow}j)] \\
& \sqsubseteq & X \mathrel{\widehat{=}} \left(\sqcap k \mid \left\{ \begin{array}{l} j < k < n \\ \sum ws[j{\rightarrow}k] \leq M \end{array} \right. \bullet \left\{ \begin{array}{ll} & M - \sum ws[j{\rightarrow}k] \\ \sqcup & \mathsf{mw}(ws{\downarrow}k) \end{array} \right. \right) \\
& & J \mathrel{\widehat{=}} I[j \backslash j+1] \wedge x = X \wedge s = \sum ws[j{\rightarrow}n] \wedge j+1 \leq n \leq N \bullet \\
& & n, s, x := j+1, ws[j], M; \\
& & n, s, x\colon [J \;,\; J \wedge (n = N \vee s > M)]\,; \qquad\qquad \triangleleft \\
& & \mathbf{if}\; s \leq M \rightarrow x := 0 \\
& & [\!]\;\; s > M \rightarrow \mathbf{skip} \\
& & \mathbf{fi} \\
& \sqsubseteq & \text{“invariant } J\text{, variant } N - n \text{”} \\
& & \mathbf{do}\; n \neq N \wedge s \leq M \rightarrow \\
& & \quad s, x\colon [n \neq N \wedge s \leq M \wedge J \;,\; J[n \backslash n+1]]\,; \qquad\qquad \triangleleft \\
& & \quad n := n+1 \\
& & \mathbf{od} \\
& \sqsubseteq & s, x := s + ws[n], x \sqcap ((M - s) \sqcup sf[n]) \;.
\end{array}
$$

That completes the development of this section. In the next section we finish the job, finding not only the minimum waste but a paragraph that produces it.

20.3 Producing the even paragraph

Let the predicate ep, *even paragraph*, be defined as follows:

$$\mathsf{ep}\, ws\ pss \ \widehat{=}\ ws = pss = \langle\rangle \vee \left\{\begin{array}{l} \mathsf{fl}\, pss = ws \\ \mathsf{ok}\, pss \\ \mathsf{wt}\, pss = \mathsf{mw}\, ws\ . \end{array}\right.$$

We then have an abstract program for producing a minimum-waste paragraph of a sequence of lengths ws:

$$\begin{array}{l} \mathbf{var}\ ws : \mathbf{seq}_N\, \mathbb{N}; \\ \quad M : \mathbb{N}; \\ \quad pss : \mathbf{seq}\,\mathbf{seq}\, \mathbb{N}; \\ \\ \mathbf{and}\ ws \neq \langle\rangle \wedge (\forall\, w : ws \bullet 0 < w \leq M) \bullet \\ \\ pss\!: [\mathsf{ep}\, ws\ pss]\ . \end{array}$$

This time we consider prefixes of ws; the even paragraph will be produced as we go. The invariant is 'if pss were extended by an even paragraph qss of the remaining text $ws{\downarrow}i$, the result $pss +\!\!+\, qss$ would be an even paragraph of the whole text ws'. Here is the first step:

$$\begin{array}{ll} \sqsubseteq & \left\{\begin{array}{l} \mathbf{var}\ i : \mathbb{N} \\ I\ \widehat{=}\ \left\{\begin{array}{l} (\forall\, qss \bullet \mathsf{ep}(ws{\downarrow}i)\ qss \Rightarrow \mathsf{ep}\, ws\ (pss +\!\!+\, qss)) \\ i \leq N \end{array}\right. \end{array}\right. \bullet \\ & i, pss := 0, \langle\rangle; \\ & i, pss\!: [I\ ,\ I \wedge i = N] \qquad \triangleleft \\ \sqsubseteq & \text{“invariant } I\text{, variant } N - i\text{”} \\ & \mathbf{do}\ i \neq N \rightarrow \\ & \quad i, pss\!: [i \neq N\ ,\ I\ ,\ i_0 < i] \qquad \triangleleft \\ & \mathbf{od}\ . \end{array}$$

Unusually, increasing i above will not necessarily just be a matter of adding 1: the next line of pss could be longer than that. Instead, we introduce a variable j to find the new value of i; the next line of pss is then $ws[i{\rightarrow}j]$.

$$\begin{array}{ll} \sqsubseteq & \mathbf{var}\ j : \mathbb{N} \bullet \\ & j\!: \left[\begin{array}{l} i \neq N \wedge I\ , \\ \left(\forall\, qss \bullet \left\{\begin{array}{ll} & \mathsf{ep}(ws{\downarrow}j)\ qss \\ \Rightarrow & \mathsf{ep}\, ws\ (pss +\!\!+\, \langle ws[i{\rightarrow}j]\rangle +\!\!+\, qss) \end{array}\right.\right) \\ \qquad i < j \leq N \end{array}\right]; \qquad \text{(iii)} \\ & i, pss := j, pss +\!\!+\, \langle ws[i{\rightarrow}j]\rangle\ . \end{array}$$

(iii) $\sqsubseteq$ "Exercise 20.1"

$$j\colon \left[i < N\ ,\ \left(\forall\, qss \cdot \left\{ \begin{array}{ll} & \mathsf{ep}(ws{\downarrow}j)\ qss \\ \Rightarrow & \mathsf{ep}(ws{\downarrow}i)\ (\langle ws[i{\rightarrow}j]\rangle +\!\!\!+\ qss) \end{array} \right. \right) \atop i < j \le N \right] .$$

The program above finds a *first* line in a paragraph of $ws{\downarrow}i$ (which the development has shown to be the *next* line in a paragraph of ws). If we know the minimum waste of $ws{\downarrow}i$, and all its suffixes, then that line is easily found: its sum is less than M, and we must have either $j = N$ or

$$\mathsf{mw}(ws{\downarrow}i) \ge (M - \textstyle\sum ws[i{\rightarrow}j]) \sqcup \mathsf{mw}(ws{\downarrow}j) .$$

That leads to

$$\begin{array}{l} \sqsubseteq\ E(j)\ \widehat{=}\ (M - \sum ws[i{\rightarrow}j]) \sqcup \mathsf{mw}(ws{\downarrow}j)\cdot \\ \quad j\colon \left[i < N\ ,\ \begin{array}{c} i < j \le N \\ \sum ws[i{\rightarrow}j] \le M \\ j = N \vee \mathsf{mw}(ws{\downarrow}i) \ge E(j) \end{array} \right] . \end{array}$$

There may be several choices for the next line — but if we take the shortest, we know its length can be no greater than M. Thus we strengthen the postcondition:

$$\sqsubseteq\ j\colon \left[i < N\ ,\ \begin{array}{c} i < j \le N \\ (\forall\, k \cdot i < k < j \Rightarrow \mathsf{mw}(ws{\downarrow}i) < E(k)) \\ j = N \vee \mathsf{mw}(ws{\downarrow}i) \ge E(j) \end{array} \right] .$$

Introducing a variable s to hold the length of the developing line (used in $E(j)$), we continue

$$\sqsubseteq \left\{ \begin{array}{l} \mathbf{var}\ s : \mathbb{N} \\ J\ \widehat{=}\ \left\{ \begin{array}{l} i < j \le N \\ (\forall\, k \cdot i < k < j \Rightarrow \mathsf{mw}(ws{\downarrow}i) < E(k))\ \cdot \\ s = \sum ws[i{\rightarrow}j] \end{array} \right. \end{array} \right.$$

$$\begin{array}{ll} & j, s\colon [i < N\ ,\ J \wedge (j = N \vee \mathsf{mw}(ws{\downarrow}i) \ge E(j))] \\ \sqsubseteq & j, s := i + 1, ws[i]; \\ & j, s\colon [J\ ,\ J \wedge (j = N \vee \mathsf{mw}(ws{\downarrow}i) \ge E(j))] \qquad \triangleleft \\ \sqsubseteq & \text{"invariant } J\text{, variant } N - j\text{"} \\ & \mathbf{do}\ j \ne N \wedge \mathsf{mw}(ws{\downarrow}i) < E(j) \rightarrow \\ & \qquad j, s := j + 1, s + ws[j] \\ & \mathbf{od}\ . \end{array}$$

And that completes the development — nearly. We still have the expressions $\mathsf{mw}(ws{\downarrow}i)$ and $\mathsf{mw}(ws{\downarrow}j)$ (in $E(j)$) in the guard of the iteration. But the program of Section 20.2 establishes

$$(\forall\, i \mid 0 \le i \le N \cdot \mathsf{mw}(ws{\downarrow}i) = sf[i])\ ,$$

and so by sequentially composing the two programs we can replace the guard with

$$j \neq N \wedge sf[i] < (M - s) \sqcup sf[j] \;.$$

And that does complete the development.

20.4 Exercises

Ex. 20.1 ♡ Recall Specification (iii) (p.205). Its postcondition says

> If the paragraph so far pss is extended by the line $ws[i \rightarrow j]$, and then by any even paragraph qss of what remains $ws \downarrow j$, the result is an even paragraph of the entire input ws.

The postcondition of its following refinement says

> The line $ws[i \rightarrow j]$, if extended by an even paragraph qss of what remains, is an even paragraph of the remaining input $ws \downarrow i$.

Explain informally why the refinement is valid, then check it rigorously using *strengthen postcondition* and *weaken precondition*.

Ex. 20.2 Collect the code developed in this chapter. Determine its time complexity in terms of M and N. Is it linear? Quadratic? Worse?
Hint: The sum s must increase on each inner iteration.

Ex. 20.3 ♡ Although there may be several even paragraphs of a single sequence of words, the code we developed is deterministic. Where, in the development, was the nondeterminism removed?

Ex. 20.4 ♡ Modify the code of Exercise 20.2 so that, using the module of Figure 20.3, it actually reads and writes words from input and output. Your final program should not contain any variables of type **seq seq** $\cdots$ (except css within the module $Words$).
Hint: Don't forget to account for spaces between words.

$$
\begin{array}{l}
\textbf{module } \mathit{Words};\\
\quad \textbf{var } css : \textbf{seq}\,\textbf{seq}\,\mathit{CHAR};\\
\\
\quad \textbf{procedure } \mathit{GetWord}\ (\textbf{result } w : \mathbb{N})\ \widehat{=}\\
\quad |[\ \ \textbf{var } cs : \textbf{seq}\,\mathit{CHAR} \bullet\\
\qquad \textbf{if}\ \ \textbf{eof} \rightarrow w := 0\\
\qquad [\!]\ \ \neg\textbf{eof} \rightarrow\\
\qquad\qquad\quad \textbf{input } cs;\\
\qquad\qquad\quad css := css +\!\!+ \langle cs \rangle;\\
\qquad\qquad\quad w := \# cs\\
\qquad \textbf{fi}\\
\quad]|;\\
\\
\quad \textbf{procedure } \mathit{PutWord}\\
\quad \widehat{=}\ \ \textbf{output } \mathsf{hd}\, css;\\
\qquad css := \mathsf{tl}\, css;\\
\\
\quad \textbf{procedure } \mathit{PutLine}\\
\quad \widehat{=}\ \ \textbf{output } \mathsf{nl}\\
\textbf{end}
\end{array}
$$

The character nl takes output to a new line.

Figure 20.3 Input/output module for words