# 11. 'while' Programs

The third programming language that we consider is the $\mathcal{W}$ language. It is similar to $\mathcal{L}$, except that instead of the loop-end instruction it has the instruction

$$\begin{array}{l}\textsf{while } V \neq 0 \textsf{ do}\\ \quad\vdots\\ \textsf{end.}\end{array}$$

We also need '$V--$' as a primitive instruction (for technical reasons).

Clearly, in contrast to $\mathcal{L}$-programs, $\mathcal{W}$-programs ***can diverge***.

We will clarify the relationship between $\mathcal{W}$-COMP and $\mathcal{G}$-COMP.

**Lemma 11.1**. $\mathcal{W}$-COMP $\subseteq$ $\mathcal{G}$-COMP.

**Proof:** $\mathcal{W}$-programs $\mathcal{P}$ can be translated (or "compiled") into $\mathcal{G}$-programs $\mathcal{P}'$, using

$$\boxed{\begin{array}{l}\textsf{while } V \neq 0 \textsf{ do}\\ \quad \mathcal{Q}\\ \textsf{end}\end{array}} \longmapsto \boxed{\begin{array}{ll}[A] & \textsf{if } V = 0 \textsf{ goto } E\\ & \mathcal{Q}'\\ & \textsf{goto } A\end{array}} \qquad \square$$

For the converse direction, it is very difficult to translate $\mathcal{G}$-COMP directly into $\mathcal{W}$-COMP. We therefore proceed by way of $\mu$PR.

**Lemma 11.2**. $\mu$PR $\subseteq$ $\mathcal{W}$-COMP.

**Proof:** EXERCISE. (*Hint:* First show that $\mathcal{W}$-COMP is closed under composition, primitive recursion and the $\mu$-operator.)

(Cf. proofs that PR $\subseteq$ $\mathcal{L}$-COMP, p. 10-3, Lemma 10.2, and
$\mu$PR $\subseteq$ $\mathcal{G}$-COMP, p.9-3, Thm 9.3, using Lemma 9.1.) $\square$

**Theorem 11.3**. $\mathcal{W}$-COMP $=$ $\mathcal{G}$-COMP ($= \mu$PR).

**Proof:** From Lemmas 11.1 and 11.2 and Thm 9.3. $\square$

NOTES:

1. This provides further confirmation for CT!

2. Again, there is a ***relativised*** notion of 'while' computability, and a relativised version of Thm 11.3:

$$\mathcal{W}\text{-COMP}(\vec{g}) = \mathcal{G}\text{-COMP}(\vec{g}).$$

This brings us to our ***final display***, in which all the questions about proper inclusions, raised in the previous pages, have been answered:

$$\boxed{\begin{array}{ccccccc} & & \text{COMP} & \overset{\text{CT}}{=} & \text{EFF} & \subset & \text{FN} \\ & & \cup & & \cup & & \cup \\ \text{PR} = \mathcal{L}\text{-COMP} & \subset & \text{TCOMP} & \overset{\text{CT}}{=} & \text{TEFF} & \subset & \text{TFN} \end{array}}$$

where 'COMP' represents *any one of* $\mathcal{G}$-COMP, $\mathcal{W}$-COMP and $\mu$PR, and 'TCOMP' means *any one of* $\mathcal{G}$-TCOMP, $\mathcal{W}$-TCOMP and T$\mu$PR (= the class of *total* $\mu$PR functions).

EXERCISES:

1. Give details of the proof of Lemma 11.2.

2. Let $\mathcal{WC}$ be the programming language for 'while' and the ***conditional*** instruction, i.e. the language $\mathcal{W}$ together with the construct

```
if V = 0
   then
      P_1
   else
      P_2
fi.
```

   Prove or disprove: $\mathcal{WC}$-COMP = $\mathcal{W}$-COMP. Do *not* use CT.

*3. Show that Ackermann's function is $\mathcal{WC}$-computable. (Write a program for Ackermann's function in $\mathcal{WC}$.)