appendix

E

The Equivalence of Iterative and Recursive Structures

In this appendix, we use our Bare Bones language of Chapter 12 as a tool to answer the question posed in Chapter 5 regarding the relative power of iterative and recursive structures. Recall that Bare Bones contains only three assignment statements (`clear`, `incr`, and `decr`) and one control structure (constructed from a `while` statement). This simple language has the same computing power as a Turing machine; thus, if we accept the Church-Turing thesis, we might conclude that any problem with an algorithmic solution has a solution expressible in Bare Bones.

The first step in the comparison of iterative and recursive structures is to replace the iterative structure of Bare Bones with a recursive structure. We do this by removing the `while` statement from the language and in its place providing the ability to divide a Bare Bones program into units along with the ability to call one of these units from another location in the program. More precisely, we propose that each program in the modified language consist of a number of syntactically disjoint program units. We suppose that each program must contain exactly one unit called `MAIN` having the syntactic structure of

```
def MAIN():
  .
  .
  .
```

(where the dots represent other indented Bare Bones statements) and perhaps other units (semantically subordinate to `MAIN`) that have the structure

```
def unit():
  .
  .
  .
```

(where `unit` represents the unit's name that has the same syntax as variable names). The semantics of this partitioned structure is that the program always begins execution at the beginning of the unit `MAIN` and halts when that unit's indented body is completed. Program units other than `MAIN` can be called as functions by means of the conditional statement

```
if name not 0:
  unit()
```

(where name represents any variable name and unit represents any of the program unit names other than MAIN). Moreover, we allow the units other than MAIN to call themselves recursively.

With these added features, we can simulate the while structure found in the original Bare Bones. For example, a Bare Bones program of the form

```
while X not 0:
  S
```

(where S represents any sequence of Bare Bones statements) can be replaced by the unit structure

```
def MAIN():
  if X not 0:
    unitA()
def unitA():
  S
  if X not 0:
    unitA()
```

Consequently, we conclude that the modified language has all the capabilities of the original Bare Bones.

It can also be shown that any problem that can be solved using the modified language can be solved using Bare Bones. One method of doing this is to show how any algorithm expressed in the modified language could be written in the original Bare Bones. However, this involves an explicit description of how recursive structures can be simulated with the while structure of Bare Bones.

For our purpose, it is simpler to rely on the Church-Turing thesis as presented in Chapter 12. In particular, the Church-Turing thesis, combined with the fact that Bare Bones has the same power as Turing machines, dictates that no language can be more powerful than our original Bare Bones. Therefore, any problem solvable in our modified language can also be solved using Bare Bones.

We conclude that the power of the modified language is the same as that of the original Bare Bones. The only distinction between the two languages is that one provides an iterative control structure and the other provides recursion. Thus the two control structures are in fact equivalent in terms of computing power.