Chapter 1
Why Schematic Functional Programming?

Though I speak with the tongues of men and of angels, and have not charity, I am become as sounding brass, or a tinkling cymbal.

Corinthians 13, 1.

Introduction

Schematic functional programming uses diagrams, or more precisely schema, to specify a functional program. We will describe a particular example of such a programming environment called Clarity. At the time of writing, and as far as we know, this is the only example of a professional programming language that is based on a combination of schema and functions.

Clarity was originally written by the authors simply because we were tired of struggling with computer coding when all we wanted to do was to create computer programs that solved our problems. The problems we had to solve were hard enough without being worried by the difficulties of getting the coding right. We could spend days hunting for some minor error such as a missing or misplaced bracket while the real issue we were trying to solve was held up. This seemed a tremendous waste of time.

Having had some experience of electronic engineering we had found the drawing and design of electronic circuits not only fun but also very productive. The translation of a drawing into an actual implementation was never a problem, so it did seem a good idea to try for something equivalent in programming. We needed the clarity of diagrammatic representation so that we could cope with the complexity of large programs. However, we were not the first to consider this and there are a few examples of successful electronic style diagrams that generate computer programs (e.g. Prograph and Matlab). Unfortunately they hit several problems with their visual

---

1 For meaning of the word ‘charity’ we prefer ‘generosity of heart’.
approach as has been shown by those who study computer psychology (Addis and Addis 2001). We will discuss this in the next section.

It would seem that with the complexity of professional programming there was a danger of getting lost in the same way as a child might get lost in the puzzle “Who is linked with which shape?” (Fig. 1.1). This and the view that visual programming languages were only toy languages and not worthy of consideration are the two main reasons for such languages not being adopted by serious or professional programmers. On the other hand many of those developing such languages have done so with the explicit intention of opening up programming only to novices.

The exceptions are those languages that were specifically designed to create interfaces to programs, such as Visual C++ or Visual Basic. This class of visual languages still has the difficult job of programming the processes that lie behind the constructed interfaces. The buttons or sliders still need to do something and to describe this means delving into the details of C++, Fortran, Pascal or Basic programming. So if we were to do anything serious with a truly visual language and assuming that such a language would help us cope with the difficulties of constructing large complex programs then the question arises as to what needs to be done to take advantage of a schematic representation.

We will explore these and other questions while we introduce the schematic functional language Clarity. It is possible to skip these explorations and just jump to the sections concerned with the practicality of programming using diagrams (or schema, etc.). Whether you skip or not we recommend that the exercises be attempted. We will indicate those sections that can be skipped with an arrow followed by a page number for those who want to just learn the practice of programming with diagrams. An example of this is shown at the end of the next heading below. The start of the practical headings on the page will be prefixed by *. Exercises and projects that are designed to give the reader practical experience are provided in most chapters.

For the reader, what will eventually emerge from these explorations, tests and a practical experience of Clarity is an elegant view of programming, a view, we believe, that will benefit his or her designing ability with any style of programming language.

Fig. 1.1 The problem with pictures