On the Big-R Notation for Describing Iterative and Recursive Behaviors

Yingxu Wang, University of Calgary, Canada

ABSTRACT

Iterative and recursive control structures are the most fundamental mechanisms of computing that make programming more effective and expressive. However, these constructs are perhaps the most diverse and confusable instructions in programming languages at both syntactic and semantic levels. This article introduces the big-R notation that provides a unifying mathematical treatment of iterations and recursions in computing. Mathematical models of iterations and recursions are developed using logical inductions. Based on the mathematical model of the big-R notation, fundamental properties of iterative and recursive behaviors of software are comparatively analyzed. The big-R notation has been adopted and implemented in Real-Time Process Algebra (RTPA) and its supporting tools. Case studies demonstrate that a convenient notation may dramatically reduce the difficulty and complexity in expressing a frequently used and highly recurring concept and notion in computing and software engineering.

Keywords: basic control structures; cognitive informatics; computing; formal methods; iteration; loop; mathematical notations; RTPA; recursion; semantics; software engineering; syntax; the big-R notation

INTRODUCTION

A repetitive and efficient treatment of recurrent behaviors and architectures is one of the most premier needs in computing. Iterative and recursive constructs and behaviors are most fundamental to computing because they enable programming to be more effective and expressive. However, unlike the high commonality in branch structures among programming languages, the syntaxes of loops are far more than unified. There is even a lack of common semantics of all forms of loops in modern programming languages (Louden, 1993; Wang, 2006a; Wilson and Clark, 1988).

When analyzing the syntactic and semantic problems inherited in iterations in programming languages, B. L. Meek concluded that: “There are some who argue that this demonstrates that the procedural approach to programming languages must be inadequate and fatally flawed, and that coping with something so fundamental as looping must therefore entail looking at computation in a different way rather than try-
The Basic Control Structures of Computing

Before the big-R notation is introduced, a survey of essential basic control structures in computing is summarized and reviewed below.

Definition 1. Basic control structures (BCS’s) are a set of essential flow control mechanisms that are used for modeling logical architectures of software.

The most commonly identified BCS’s in computing are known as the sequential, branch, case (switch), iterations (three types), procedure call, recursion, parallel, and interrupt structures (Backhouse, 1968; Dijkstra, 1976; Wirth, 1976; Backus, 1978; de Bakker, 1980; Jones, 1980; Cries, 1981; Hehner, 1984; Hoare, 1985, Hoare et al., 1987; Wilson & Clark, 1988; Louden, 1993; Wang, 2002; Horstmann & Budd, 2004). The 10 BCS’s as formally modeled in RTPA (Wang, 2002) are shown in Table 1. These BCS’s provide essential compositional rules for programming. Based on them, complex computing functions and processes can be composed.

As shown in Table 1, the iterative and recursive BCS’s play a very important role in programming. The following theorem explains why iteration and recursion are inherently vital in determining the basic expressive power of computing.

Theorem 1. The need for software is necessarily and sufficiently determined by the following three conditions:

a. The repeatability: Software is required when one needs to do something for more than once.

b. The flexibility or programmability: Software is required when one needs to repeatedly do something not exactly the same.

c. The run-time determinability: Software is required when one needs to flexibly do something by a series of choices on the basis of varying sequences of events determinable only at run-time.