Thinking About Algorithms

Sangeet Khemlani (khemlani@princeton.edu)
P.N. Johnson-Laird (phil@princeton.edu)
Department of Psychology, Princeton University
Princeton, NJ 08540 USA

Keywords: algorithms; reasoning; deduction; computer programming; problem solving

Introduction

Computer programming depends on high-level cognitive abilities. While theories of programming exist, they tend to focus on how individuals learn to program (e.g., Soloway, Bonar, & Ehrlich, 1989; Anderson, Pirolli, & Farrell, 1988). Our research aims to answer the following questions: Can naïve individuals solve algorithmic problems? Can they generate descriptions of algorithms?

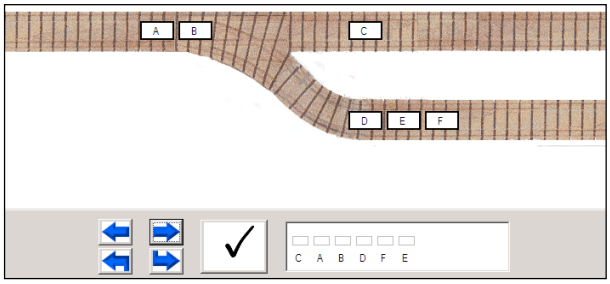

Figure 1. Railway environment for algorithmic reasoning

To answer these questions, we developed an environment that non-programmers can readily grasp, but that allows us to test their ability to reason about recursive processes that are commonplace in programming languages, such as Lisp. The environment concerns railway trains consisting of several cars; tasks call for the cars to be reordered by using a switch leading to a siding (see Figure 1).

This system is equivalent to a push-down automaton, in which the siding acts as a stack-like memory. The system provides a concrete mechanism for any such automaton, and with two extendible sidings, cars of two sorts, and the capacity to add and subtract cars, it has the power of a Universal Turing machine, i.e., it can perform arbitrary computation. Hence, the system allows us to study the reverse engineering of finite and infinite automata.

Experiment 1: List processing

Our first aim was to examine whether naïve individuals were capable of the reasoning required to program simple list-processing operations of the sort that occur in a programming language such as Lisp. Twenty participants were given problems in which they had to rearrange the cars on one side of the track and place them on the other, similar to list processing problems in introductory computer science courses. Naïve individuals were able to solve these problems with ease: they produced no erroneous responses. We also observed the subtle phenomenon that the relational complexity of a move (see Halford et al., 1998) – that is, in our system the number of cars to be moved – affected performance.

Experiment 2: Description of algorithms

The experiment called for the participants to formulate descriptions of how to solve three common list-processing problems (reversals, palindromes, and sortings) in the railway environment. They tackled the problems based on two sorts of trains. The *determinate* trains contained a single set of eight cars, whereas the *indeterminate* trains contained an indeterminate number of cars, signified by an ellipsis (e.g., A…Z). Table 1 presents the percentages of correct descriptions of algorithms: the determinate problems were reliably easier than the indeterminate problems (Wilcoxon test, $z = 2.69$, $p < .005$). A common mistake was for participants to treat the ellipsis as though it were a single car despite clear instructions that it signified an indeterminate number of cars.

Table 1: Percentage of correct algorithms by level of determinacy

Determinacy	Reversals	Palindromes	Sortings
Determinate	100	100	100
Indeterminate	85	42	57

Discussion

Algorithmic reasoning is central to the mental processes of computer programming. We found that naïve individuals with no training in computer science were able both to solve list-processing problems (Experiment 1) and to describe the recursive loops of operations needed to solve these problems (Experiment 2).

References

Anderson, J. R., Pirolli, P., & Farrell, R. (1988). Learning to program recursive functions. In M. Chi, R. Glaser, & M. Farr (Eds.), *The Nature of Expertise*. Hillsdale, NJ: Erlbaum.

Halford, G.S., Wilson, W.H., & Phillips, S. (1998). Processing capacity defined by relational complexity: Implications for comparative, developmental, and cognitive psychology. *Behavioral and Brain Sciences*, 21.

Soloway, E., Bonar, J., & Ehrlich, K. (1989). Cognitive strategies and looping constructs: An empirical study. In E. Soloway and J. C. Spohrer (Eds.), *Studying the Novice Programmer*. Hillsdale, NJ, Lawrence Erlbaum Associates.