Pattern-guided program synthesis

The motivation behind analyzing consistency of execution traces with desired output in Chap. 6 was to identify and promote the programs that contain prospectively useful subprograms. The approach described in this chapter generalizes the trace consistency method in two respects. Firstly, we seek here for a more general relatedness between the intermediate execution states and the desired output, rather than for information-theoretic consistency. Secondly, while evaluation in the trace consistency method depends on a single stage of program execution that maximizes consistency (captured in a particular column of an execution record (6.4)), the evaluation functions proposed here depend on the entire execution record. In this way, the approach presented in this chapter, originally proposed in [101] and then extended in [96], looks for patterns in program behaviors that seem relevant for a given program synthesis task.

7.1 Motivation

The motivations for relying on general ‘relatedness’ between intermediate execution states and program output and for taking into account entire execution records can be illustrated with the following example.

Example 7.1. Assume the task is to synthesize a program that checks if a quadratic polynomial \( ax^2 + bx + c \) has roots in the real domain. The input to the program is a triple of numbers \((a, b, c)\) and the desired output is an appropriate Boolean value. Consider the candidate program for this task, which we present as a function in the Scala programming language in Fig. 7.1a to demonstrate that the formalism of execution record is applicable to conventional programming languages.

Fig. 7.1b shows the corresponding execution record for this program, composed of three traces. The program is clearly very close to being correct; its only deficiency is disregarding the \texttt{delta} variable. Fixing this is straightforward for a human programmer: the \texttt{return} statement in line 4 needs
Fig. 7.1: (a) A partially correct program in Scala for determining if a quadratic polynomial \( ax^2 + bx + c \) has real roots. (b) The execution record for this program for three exemplary inputs. The columns \( p_i \) of the execution record are numbered consistently with the line numbers in the program listing.

to be extended to return hasDegreeTwo && (delta >= 0). The components required for expressing the target concept are calculated at intermediate execution stages in lines 2 and 3. However, this program fails to combine them in the right way.

The conventional objective function \( f_o \) would judge this program only by its output, which would be incorrect for many tests. The trace consistency evaluation function \( f_{tc} \) (6.4) from Chap. 6 should be able to notice that the intermediate execution states reached in lines 2 and 3 (i.e. the values of variables hasDegreeTwo and delta calculated there) have relatively high information-theoretic consistency with the desired output. Depending on the actual tests used, the two-way conditional entropy is likely to be minimized by one of the random variables \( X_k \) associated with these locations (6.4). However, only one of them will be ultimately reflected by \( f_{tc} \). \( f_{tc} \) cannot take into account that it is the combination of these expressions (subprograms) that is particularly promising. As a consequence, the above program may attain the same value of \( f_{tc} \) as its close relatives, e.g., a program that misses line 3.

The lesson learned from this example is that consistency of execution states with the desired output is only one type of potentially useful behaviors that can emerge in candidate programs synthesizes in an iterative framework like GP. There are other, more complex and more subtle behaviors that can be the telltales of prospective performance. The challenge lies in making a program synthesis method capable of detecting such behavioral patterns.

A skilled human programmer may discover behavioral patterns and exploit them to design a program that meets the specification of a program synthesis task. Humans in general are known to be incredibly good at spotting patterns and thinking in patterns when solving all sorts of problems – it is not for no reason that they have been termed informavores [128]. A great deal of AI research is about modeling and mimicking such capabilities [53]. Moreover, humans can anticipate the patterns that are desirable in a given problem and often use domain and common sense knowledge for that sake.