Abstract

Today's software systems are complex artifacts whose behavior is often extremely difficult to understand. This fact has led to the development of sophisticated formal methodologies for program analysis, comprehension, and debugging.

Trace analysis is concerned with techniques that allow execution traces to be dynamically searched for specific contents. The search can be carried out *forward* or *backward*. While forward analysis results in a form of *impact analysis* that identifies the scope and potential consequences of changing the program input, backward analysis allows *provenance analysis* to be performed; i.e., it shows how (parts of) a program output depends on (parts of) its input and helps estimate which input data need to be modified to accomplish a change in the outcome.

In this thesis, we investigate a number of trace analysis methodologies that are suitable for analyzing complex, textually-large execution traces in rewriting logic (RWL), which is a *logical* and *semantic framework* particularly suitable for formalizing highly concurrent, complex systems.

The first part of the thesis is devoted to develop an incremental, slicing-based backward trace analysis technique that achieves huge reductions in the size of the trace. This methodology favors better analysis and debugging since most tedious and irrelevant inspections that are routinely performed during diagnosis and bug localization can be eliminated automatically. This technique is illustrated by means of several examples that we execute by using the *i*JULIENNE system, an interactive trace slicer that we developed which implements the backward trace analysis technique.

The second part of the thesis formalizes a rich and highly dynamic, parameterized scheme for exploring rewriting logic computations. The scheme implements a generic animation algorithm that allows the nondeterministic execution of a given *conditional* rewrite theory to be followed up by using different modalities, including *incremental stepping* and *automated forward/backward slicing*, which drastically reduce the size and complexity of the traces under examination and allow users to evaluate the effects of a given statement or instruction in isolation, track input change impact, and gain insight into program behavior (or misbehavior). Moreover, cutting down the execution trace can expose opportunities for program optimizations. With this methodology, an analyst can browse, slice, filter, or search the traces as they come to life during the program execution. The generic trace analysis framework has been implemented into the Anima system and we report a thorough experimental evaluation that we conducted which assesses the usefulness of the proposed approach.