Thesis Overview

A Compiler Technology Based Approach to Simulation of Basic Forms of Automata

Ph.D. Thesis, May 2012

Pinaki Chakraborty
School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi 110067, India.
E-mail: pinaki_chakraborty_163@yahoo.com

An automaton is an abstract model of a computer. Various types of automata have been developed to date to model computer systems and computing processes. Theory of automata includes the study of these automata along with their properties and applications. Theory of automata is often taught, learned and researched using tools known as automata simulators [1]. Automata simulators can be used to model large and complex automata, simulate them for any input string and study their properties. The automata simulators may be classified as language based automata simulators, visualization centric automata simulators and hardware automata simulators [2]. A language based automata simulator receives the definition of an automaton in a formal language, which may be notational, assembly-like, procedural or descriptive, and simulates it using some technique for processing formal languages. A visualization centric automata simulator receives the specification of an automaton in either a tabular or a diagrammatic form and visually simulates it. A hardware automata simulator is implemented fully or partially in hardware and simulates an automaton using actual hardware manipulations.

This dissertation formalizes a comprehensive approach based on compiler technology [3-5] to simulate some basic forms of automata *viz*., finite accepters, finite transducers, pushdown accepters and Turing machines. These basic forms of automata are known for their simplicity and generality, hence widely studied and used in research. An Automata Description Language has been defined for modeling these basic forms of automata. The language has three sublanguages *viz*., descriptive sublanguage, pseudo-assembly sublanguage and procedural sublanguage, implementing three paradigms of programming. The sublanguages have the same scope and can be used to define any automata howsoever large and complex. The language supports both deterministic and nondeterministic forms of automata. In this language, an automaton may be composed of one or more submachines that may be developed even by a third party. All automata modeled in the Automata Description Language are required to be compiled before they can be simulated or their properties can be studied, hence the name of the approach.

A reference compiler for the Automata Description Language has been developed. The compiler is a single-pass compiler comprising of lexical analysis, syntax analysis, semantic analysis and code generation phases, and bookkeeping and error handling modules. The compiler translates an automaton modeled in the Automata Description Language into an object program in an intermediate language, provided that the source program is correct. The same compiler can be used for all the three sublanguages of the Automata Description Language.

The object program generated by the compiler can be simulated for any input string by a suitably developed simulator. The simulator starts from the initial configuration of the automaton and try to reach an accepting configuration following the rules specified by the object program. Nondeterminism has been implemented in the simulator using backtracking. Whenever the simulator finds multiple possible paths for simulation, it pushes all but one of them onto a stack. Alternatively, when the simulator finds no path ahead it pops off a configuration from the stack. The backtracking mechanism works internally and without concerning the user.

A toolkit has been proposed and partially implemented to add utility to the Automata Description Language. The toolkit includes tools to analyze the properties of an automaton, convert one form of automata into another, minimize the number of internal states in an automaton, display a graphical representation of an automaton and generate the language accepted by an automaton. All these tools use the object programs generated by the compiler.

Some example automata have been modeled in the Automata Description Language. The contents of their object programs were analyzed and the time required to compile and simulate them were also noted. It was observed that the descriptive sublanguage is best suited for modeling small and simple automata. Alternatively, the procedural sublanguage is the most efficient for large and complex automata followed by the pseudo-assembly sublanguage.

This study is expected to be helpful in further research on simulation of basic forms of automata and encourage the use of automata simulators in both academic and research environments. Additionally, the concepts used to implement nondeterminism in the Automata Description Language may also be used to develop nondeterministic general purpose procedural languages.

ACKNOWLEDGEMENTS

Prof. C. P. Katti and Prof. P. C. Saxena supervised the research leading to this thesis. The research was partly supported by a research fellowship from University Grants Commission, New Delhi.

REFERENCES

[1] P. Chakraborty, P. C. Saxena and C. P. Katti, "Automata simulators: classic tools for computer science education," *British Journal of Educational Technology*, 2012, **43**(1): E11-E13.

[2] P. Chakraborty, P. C. Saxena and C. P. Katti, "Fifty years of automata simulation: a review," *ACM Inroads*, 2011, **2**(4): 59-70.

[3] P. Chakraborty, P. C. Saxena and C. P. Katti, "A compiler-based toolkit to teach and learn finite automata," *Computer Applications in Engineering Education*, in press.

[4] P. Chakraborty, "A language for easy and efficient modeling of Turing machines," *Progress in Natural Science*, 2007, **17**(7): 867-871.

[5] P. Chakraborty, S. Taneja, P. C. Saxena and C. P. Katti, "Teaching purpose compilers – an exercise and its feedback," *ACM Inroads*, 2011, **2**(2): 47-51.