An Executable Formal Semantics For A Functional Actor Language

Xiaohong Chen, Lucas Peña

University of Illinois at Urbana-Champaign,
{xc3, lpena7}@illinios.edu

Abstract. A simple reference actor language has been proposed in [], serving as a sound mathematical calculus foundation for various actor languages used in distributed applications that involves concurrency. Even though the authors of [] provided in the paper an operational semantics of the actor language, an executable implementation was not given. In this project, we will give the reference actor language an implementation that by construction conforms to its formal operational semantics using the K framework [], a rule-based semantic framework. We will show that our executable formal semantics are capable of executing some meaningful actor systems examples. We hope our executable formal semantics can be served as a starting point for future work in developing automatic equivalence provers using the K framework.

1 Introduction

The K framework [] is a framework for defining executable operational semantics of a programming language. K is based on rewriting logic, organizing the state of a program into cells and rewriting the particular parts of the configuration. K also has access the entire computation at any particular time. These features and more allow for difficult language constructs to be designed with ease in K. On top of that, K provides backend tooling for additional features based on the semantics of a programming language, such as a semantic debugger and a full formal verifier. See Figure 1 for a complete view of the tools one gets from K based off an operational semantics.

1.1 Related Work

To demonstrate the power of K, languages as complex as C, Java, and JavaScript have been fully defined using the K framework [**?**,**?**,**?**]. On top of that, in all of these languages and more, nontrivial programs have been formally verified using K's prover, such as proofs involving AVL trees and red-black trees. We would like to add an actor-based language to the growing list of languages defined in K, so we can take advantage of K's formal verification engine and more.

Semantics of actor-based languages have been defined on paper, for example in []. Additionally, there are many implementations of actor-based systems, including Erlang [], Akka [], Scheme [], and much more. However, there is currently

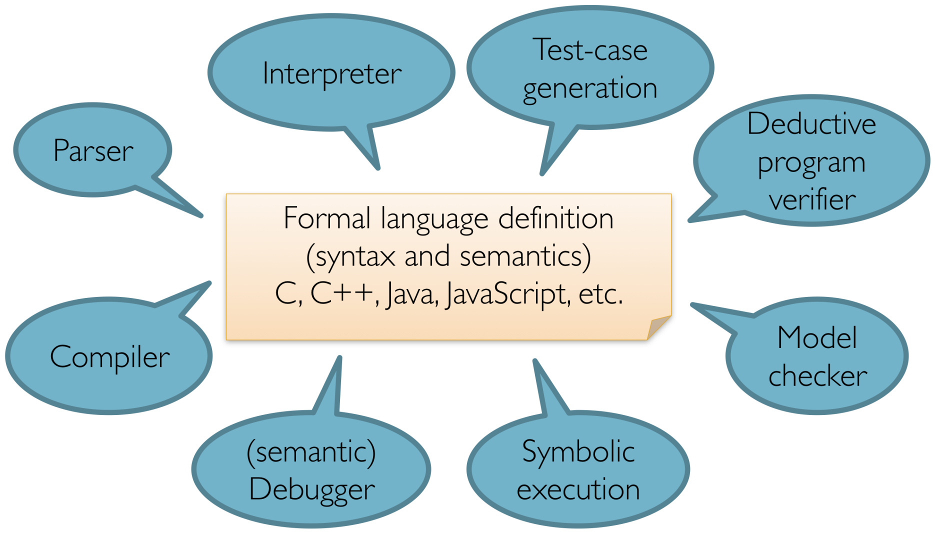
Interpreter
Test-case generation
Parser
Deductive program verifier
Formal language definition (syntax and semantics) C, C++, Java, JavaScript, etc.
Compiler
Model checker
(semantic) Debugger
Symbolic execution

Fig. 1. Backend tools K provides from just an operational semantics. Some tools such as a semantics-based compiler and test-case generation are in progress.

no formally defined *executable* operational semantics of an actor-based language. In K, there was previous work done to define the semantics of Orc, a concurrent programming language [], though that work is incomplete.

In this paper, we aim to define the semantics of the actor language proposed in []. We choose this language as the semantics are clearly defined in the paper, and it is one of the more simple actor frameworks. Defining this language in K will give assurance to the correctness of the language - as K is executable this can be tested much easier than defining a language on paper. It will also provide more assurance on K's language-defining ability, as well as open up to formal analysis using the tools seen in Figure 1.

2 The Actor Language in K

2.1 Configuration

2.2 Syntax and Evaluation Contexts

2.3 Semantics

2.4 In-depth Example

3 Conclusion and Future Work

References