Computational Logical Frameworsk and Generic Program Analysis Technologies

José Meseguer

Computer Science Department
University of Illinois at Urbana-Champaign

Motivation

The technologies developed to solve the verifying compiler grand challenge should be generic: not tied to a particular language but widely applicable to many languages.

Such technologies should also be semantics-based, that is, based on a rigorous formal semantics of the languages.

For this, a computational logical framework with efficient executability and a spectrum of meta-tools can serve as a computational semantic framework to develop generic program analysis techniques that can generate powerful analysis tools for each language of interest.

Not all logical frameworks can serve such purposes well. I list some specific requirements that I think are important:

Logical Framework Requirements

1. good data representation capabilities,
2. support for concurrency and nondeterminism,
3. simplicity of the formalism,
4. efficient implementability, and efficient meta-tools
5. support for reflection,
6. initial model semantics, to suport inductive reasoning.

The Rewriting Logic/Maude Experience

At UIUC, Grigore Roşu and I, together with several students, are developing semantic definitions of programming languages based on rewriting logic (RWL) and executed in the Maude RWL language to generate analysis tools for free for a wide range of languages such as: Java and the JVM, Scheme, ML, Haskell, and bc.

Rewriting logic meets the requirements mentioned above, and supports semantic definition of programming languages that combine algebraic denotational semantics and SOS semantics is a seamless way. Given a language L its semantics is a rewrite theory

$$\mathcal{R}_L = (\Sigma_L, E_L, R_L)$$

The Rewriting Logic/Maude Experience (II)

When executed in Maude the RWL formal semantics $\mathcal{R}_L$ of language L automatically becomes an efficient interpreter for L: for example, faster than the Linux bc interpreter, and 1/2 the speed of the Scheme interpreter.

Furthermore, Maude's formal tools, such as its inductive theorem prover, LTL model checker, and breadth-first search capability then become meta-tools from which we derive useful program analysis tools for L using $\mathcal{R}_L$.

We are furthermore developing new generic program analysis technologies such as, for example, a generic partial-order reduction technique than can apply to any language L with threads, and does not require any changes to an underlying model checker.

The Rewriting Logic/Maude Experience (III)

The cost of generating tools for a language L this way using its formal semantic definition $\mathcal{R}_L$ is much lower (in the order of weeks) than that of building similar language specific analysis tools (man years). For example, it took Feng Chen at UIUC only three weeks to develop $\mathcal{R}_L$ for a large subset of Java including multithreading, inheritance, polymorphism, object references, and dynamic object allocation.

Furthermore, the formal analysis tools obtained for free from $\mathcal{R}_L$ are competitive with similar language-specific tools such a NASA-Ames' Java Path Finder and Stanford's Java model checker. Similarly, our generic partial order reduction technique can achieve rates of space and time reduction similar to those of language-specific tools such as SPIN.

Future Directions Related to the Grand Challenge

The above results, although encouraging, are very much work in progress; we would like to advance in addition the following directions:

1. Generation of provably correct compilers from the formal semantics $\mathcal{R}_L$ of a language L.

2. Language-generic theorem proving environments.

3. Language-generic program abstraction techniqes.

4. Modular programming language definitions in the spirit of MSOS.