Matching Logic
A New Program Verification Approach

Grigore Rosu and Andrei Stefanescu
University of Illinois at Urbana-Champaign

(work started in 2009 with Wolfram Schulte at MSR)
Usable Verification ...

• Relatively clear objectives:
  – Better tools, more connected, more user friendly
  – Teach students verification early
  – Get the best from what we have

• But ... could it be that, after 40 years of program verification, we still lack the right semantically grounded program verification foundation?
Current State-of-the-Art

Consider some programming language, L

• Formal semantics of L
  – Typically skipped: considered expensive and useless

• Model checkers for L
  – Based on some adhoc encodings of L

• Program verifiers for L
  – Based on some other adhoc encodings of L

• Runtime verifiers for L
  – Based on yet another adhoc encodings of L

• ...
Semantic Gap

• Why would I trust any of these tools for L?
• How do they relate to L?
• What is L?

• Example: the C (very informal) manual implies that \((x=0)+ (x=0)\) is undefined
  – Yet, all C verifiers we looked into “prove” it = 0
Ideal Scenario

- Have *one formal definition of L* which serves all the semantic and verification purposes

  Execution of L programs
  (use for extensive testing)

  Model checking of L programs

  Proving L programs correct

  ...
Our Approach

• Define languages using the K framework
  – A rewrite based framework which generalizes both evaluation contexts and the CHAM

• A programming language is a K system
  – Algebraic signature (syntax + configuration)
  – K rewrite rules (make read/write parts explicit)

• “Compile” K to different back-ends
  – To OCAML for efficient interpreters (experimental)
  – To Maude for execution, debugging, verification
Matching Logic

• Builds upon operational semantics
  – We use K, but in principle can work with any op semantics: a formal notion of configuration is necessary
  – With K, we do not modify anything in the original sem!

• Extends the PL semantics with matching logic specifications, or patterns; for example,

\[
\langle \langle \text{root} \mapsto \text{?root}, \ E \rangle_{\text{env}} \langle \text{tree(?root)}(T), \ H \rangle_{\text{heap}} \ C \rangle_{\text{config}}
\]

specifies all configurations in which program variable root points to a tree T in the heap
Demo
Highlights

• Matching logic builds upon giants’ shoulders
  – Matching and rewriting “modulo” have been researched extensively; efficient algorithms (Maude) despite its complexity (NP complete w/o constraints)
  – Mathematical universe axiomatized using well understood and developed algebraic specification

\[
\begin{align*}
\text{rev}(\text{nil}) &= \text{nil} \\
\text{rev}([a]) &= [a] \\
\text{rev}(A_1@A_2) &= \text{rev}(A_1)@\text{rev}(A_2)
\end{align*}
\]
Matching is Powerful

• The underlying rewrite machinery of K works by means of matching
  – So programming language semantics, which is most of the matching logic rules, is matching

• Pattern assertion reduces to matching

• Framing reduces to matching

• Separation reduces to matching

• Nothing special needs to be done for separation or for framing!
K and Matching Logic Scale

• We defined several real languages so far
  – Complete: C (C99), Scheme
  – Large subsets: Verilog, Java 1.4
  – In work: X10, Haskell, JavaScript
• And tens of toy or paradigmatic languages
• We next give an overview of the C definition
  – Defined by Chucky Ellison (PhD at UIUC)
Configuration of C

- 57 leaf cells
- 63 nested cells

Heap
Statistics for the C definition

- Syntactic constructs: 173
- Total number of rules: 812
- Total number of lines: 4688

- Has been tested on thousands of C programs (several benchmarks, including the gcc torture test – passed 95% so far)
Conclusion and Future Work

• Formal verification should start with a formal, executable semantics of the language
• Once a well-tested formal semantics is available, developing program verifiers should be an easy task, available to masses

• Matching logic aims at the above
• It makes formal semantics useful!
• It additionally encourages developing formal semantics to languages, which in K is easy and fun