Streamlining Policy Creation in Policy Frameworks

Mark Hills[1]

Centrum Wiskunde & Informatica, Amsterdam, The Netherlands

Abstract. *Policy frameworks* provide a technique for improving reuse in program analysis: the same language frontend, and a core analysis semantics, can be shared among multiple analysis policies for the same language, while analysis domains (such as units of measurement) can be shared among frameworks for different languages. One limitation of policy frameworks is that, in practice, adding a new policy can still require a significant level of knowledge about the internals of the semantics definition. This abstract describes work on extending policy frameworks to solve this limitation, making policies reflective over their requirements and generating the policy semantics from a higher-level policy description language.

Using executable language definitions, such as rewriting logic semantics [9] (RLS) or K [11] definitions running in Maude [4], program analysis can be treated as a form of non-standard program evaluation over appropriate domains of abstract values. An example of this approach is the unit safety analysis developed first for BC [3] (a small calculator language) and then for a small subset of C [10]. In this analysis, the abstract values were units of measurement [1] (e.g., meters, seconds, lumens). The analysis semantics modeled the operation of language constructs over these units, detecting errors in cases where units were used incorrectly, e.g., when two different units were added.

However, in this work, rules specific to the analysis were tangled with analysis-agnostic rules, making it challenging to reuse parts of the existing semantics in a new analysis. Solving this problem was the goal of the C Policy Framework [5], or CPF, an extensible analysis framework for C defined in Maude. The core of CPF includes an analysis-generic frontend, allowing annotations to be added (in comments) as function contracts or within function bodies, and an abstract C semantics. To define a specific analysis *policy*, this core is extended with an analysis-specific definition of abstract values, a specific annotation language, and equational definitions for a number of *hooks*, representing points in the semantics that differ between analysis policies. Later work extended this to the SILF language [7], which provides a simpler, more modular environment for experimenting with analysis policies. This work also extended the annotation mechanism to type-like annotations, a feature subsequently added to CPF.

Unfortunately, even though the frameworks for C and SILF were structured modularly [8,2], they were sometimes not modular *in practice*, a point raised in conversations with the author. To actually implement a new policy, the implementer requires a detailed knowledge of both Maude and of the entire provided core semantics. This includes knowing which hooks must be defined to provide the policy-specific semantics and which modules provide base variants of functionality that can be directly reused or extended. In this abstract we describe ongoing work on two features being added to the SILF policy framework (and later to CPF) to help solve this problem: the reflective extraction of extension point information, and a DSL for describing analysis policies.

Reflective Extraction of Extension Point Information Each hook operation defined in the core framework semantics is identified using a Maude `metadata` attribute. Additional operations are defined equationally to identify modules that can be used directly or extended to provide specific features, e.g. a base annotation language. Using a combination of standard rewriting and Maude's reflective capabilities, the policy description language, described below, can then "ask" a framework about defined extension points.

A Policy Description Language Using the extracted policy information, a policy description language is used to describe the three standard parts of an analysis policy. First, the domain of policy values is defined algebraically, with pretty-printing rules defined to display policy values appropriately in messages. Second, the analysis-specific behavior for each hook is defined, making use of the policy values and of predefined reporting operations for errors and warnings (potentially with source locations [6]). Third, the annotation language used for the policy is defined by extending a provided base annotation language with policy-specific annotations, e.g., `@unit(E)` to calculate the unit of an expression in a units policy. These three items are then used to generate a parser for the annotation language, used in conjunction with the language parser, and a Maude specification of the value domain and the analysis semantics. This generation process also creates many of the repetitive cases needed to properly handle the propagation of error information from children to parents (ensuring errors in child expressions do not trigger new, spurious errors in parents), something done manually now.

References

1. NIST Website, International System of Units (SI). http://physics.nist.gov/cuu/Reference/unitconversions.html.
2. C. Braga and J. Meseguer. Modular Rewriting Semantics in Practice. In *Proceedings of WRLA'04*, volume 117 of *ENTCS*, pages 393–416. Elsevier, 2005.
3. F. Chen, G. Roşu, and R. P. Venkatesan. Rule-Based Analysis of Dimensional Safety. In *Proceedings of RTA'03*, volume 2706 of *LNCS*, pages 197–207. Springer, 2003.
4. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *LNCS*. Springer, 2007.
5. M. Hills, F. Chen, and G. Roşu. A Rewriting Logic Approach to Static Checking of Units of Measurement in C. In *Proceedings of RULE'08*. Elsevier, 2008. To Appear.
6. M. Hills, P. Klint, and J. Vinju. RLSRunner: Linking Rascal with K for Program Analysis. In *Proceedings of SLE'11*, LNCS. Springer-Verlag, 2011. To Appear.
7. M. Hills and G. Roşu. A Rewriting Logic Semantics Approach To Modular Program Analysis. In *Proceedings of RTA'10*, volume 6 of *Leibniz International Proceedings in Informatics*, pages 151 – 160. Schloss Dagstuhl - Leibniz Center of Informatics, 2010.
8. J. Meseguer and C. Braga. Modular Rewriting Semantics of Programming Languages. In *Proceedings of AMAST'04*, volume 3116 of *LNCS*, pages 364–378. Springer, 2004.
9. J. Meseguer and G. Rosu. The rewriting logic semantics project. *Theoretical Computer Science*, 373(3):213–237, 2007.
10. G. Roşu and F. Chen. Certifying Measurement Unit Safety Policy. In *Proceedings of ASE'03*, pages 304 – 309. IEEE, 2003.
11. G. Roşu and T. F. Şerbănuţă. An Overview of the K Semantic Framework. *Journal of Logic and Algebraic Programming*, 79(6):397–434, 2010.