An overview of a proof theoretical approach to reasoning about computation

Dale Miller[1]

Abstract

Typing rules and structural operational semantics are often given via inference rules: that is, the justification of a typing or an evaluation is actually a proof. Thus it is not surprising that proof theory can be used to benefit the specification of and the reasoning about computation. An additional advantage of using proof theory is that it can support such "intensional" aspects of computation as resources (say, via linear logic) and bindings (say, via term-level and proof-level bindings). In this talk, I will overview recent work on designing a proof theoretic framework for reasoning about both the static and dynamic semantics of specifications languages and programming languages. A synthesis of the following topics will be provided: λ-tree syntax, mobility of binders, ∇-quantification, two-level logic architecture, induction and coinduction, and focusing proof systems.

An annotated, partial bibliography

The technical material for this overview is contained in a number of papers, some of which are briefly described in the following annotated bibliography.

λ-tree syntax. An earlier illustration of the kinds of computations that are possible in λProlog by directly manipulating λ-terms is contained in [MN87]. Later this style of manipulation was called *higher-order abstract syntax*, but since that term came to mean different things to different communities, the term λ-*tree syntax* was introduced in [Mil00] to denote the original form of λ-term manipulation.

[MN87] D. Miller and G. Nadathur. A logic programming approach to manipulating formulas and programs. In Seif Haridi, editor, *Sym. on Logic Programming*, pp. 379–388, 1987.

[Mil00] D. Miller. Abstract syntax for variable binders: An overview. In John Lloyd and et. al., editors, *Computational Logic - CL 2000*, LNAI 1861, pp. 239–253. Springer, 2000.

Definitions and induction. In order to reason about what can and cannot be proved from a given specification, it is necessary to be able to think of a logic specification as being *closed* or *defined*. Induction can then also be described on such definitions. A suitable proof theory presentation of these ideas can be applied

[1] INRIA Saclay - Île-de-France & LIX/École Polytechnique, 91128 Palaiseau, France. Support from INRIA through the "Equipes Associées" Slimmer is gratefully acknowledged.

to not only first-order terms (*i.e.*, parse-tree syntax) but also simply typed λ-terms (*i.e.*, λ-tree syntax). McDowell developed a proof theory approach to definitions and induction in his PhD thesis [McD97] and in [MM00]. A two-level approach to reasoning about operational semantic was presented in [MM02]: since that paper did not contain the ∇-quantifier, certain encoding techniques used in that paper were rather heavy and painful.

[McD97] R. McDowell. *Reasoning in a Logic with Definitions and Induction.* PhD thesis, University of Pennsylvania, December 1997.

[MM00] R. McDowell and D. Miller. Cut-elimination for a logic with definitions and induction. *Theoretical Computer Science*, 232:91–119, 2000.

[MM02] R. McDowell and D. Miller. Reasoning with higher-order abstract syntax in a logical framework. *ACM Trans. on Computational Logic*, 3(1):80–136, 2002. Extended version of a paper in LICS97.

∇-quantification and finite behaviors. A proof theoretic presentation of "generic judgments" and the associated ∇-quantifier [MT05] were introduced, in part, to help address the above mentioned encoding problem of [MM02]. The proof theory of ∇ was developed in Tiu's PhD thesis [Tiu04]. To help validate the design of ∇, logic based specifications of the (finite) π-calculus were explored in detail in subsequent papers, such as, [Tiu05] and [TM].

[Tiu04] A. Tiu. *A Logical Framework for Reasoning about Logical Specifications.* PhD thesis, Pennsylvania State University, May 2004.

[MT05] D. Miller and A. Tiu. A proof theory for generic judgments. *ACM Trans. on Computational Logic*, 6(4):749–783, October 2005. Extended version of LICS03 paper.

[Tiu05] A. Tiu. Model checking for π-calculus using proof search. In Martín Abadi and Luca de Alfaro, editors, *CONCUR*, LNCS 3653, pp. 36–50. Springer, 2005.

[TM] A. Tiu and D. Miller. Proof Search Specifications for Bisimulation and Modal Logics for the π-calculus. Submitted May 2008.

∇-quantification and infinite behaviors. While ∇-quantification works well with "finite" systems, a number of questions remained about how best to extend it to infinite systems, *i.e.*, systems in which induction and coinduction are needed to establish proofs. Tiu's thesis [Tiu04] provided general inference rules for induction and coinduction but these did not interact sufficiently well with ∇-quantification. Tiu has proposed [Tiu06] adding some structural rules to the ∇-quantifier and these have allowed him to develop a more expressive form of induction. Baelde has devised another approach to (co)induction [Bae08] that does not need to extend (with structural rules) the original, "minimal" description of ∇ in [MT05]. Gacek *et. al.* show that if the definition mechanism is lifted from atomic judgments to generic judgments, the resulting logic gains an important aspect of expressiveness: one that is particularly useful for reasoning about the context of object-level proof contexts [GMN08].

[Tiu06] A. Tiu. A logic for reasoning about generic judgments. In A. Momigliano and B. Pientka, editors, *LFMTP 2006.*

[Bae08] D. Baelde. On the expressivity of minimal generic quantification. In Andreas Abel and Christian Urban, editors, *LFMTP 2008: International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice*, 2008.

[GMN08] A. Gacek, D. Miller, and G. Nadathur. Combining generic judgments with recursive definitions.

In F. Pfenning, editor, *23th Symp. on Logic in Computer Science*, 2008.

Implementations. An effective implementation of λProlog is available via the Teyjus compiler [NM99]: this compiler is able to animate directly a number of semantic specifications involving, for example, the λ-calculus or the π-calculus. The Bedwyr system [BGMNT07] is a model checker that supports λ-tree syntax by implementing proof search in a logic with finite fixed points and the ∇-quantifier. To account for (potentially) infinite behaviors, induction and coinduction play critical roles. The Abella prover [Gac08] provides an interactive proof editor for a two-level approach [MM02] to reasoning about operational semantics based on Tiu's LG logic [Tiu06] and the ability to define generic judgments. The Taci prototype theorem prover [BSV08] is being developed to automate theorem proving in this domain and to support Baelde's approach [Bae08] to integrating ∇ and fixed points.

[NM99] G. Nadathur and D. Mitchell. System description: Teyjus — A compiler and abstract machine based implementation of λProlog. In H. Ganzinger, editor, *16th Conference on Automated Deduction (CADE)*, LNAI 1632, pp. 287–291, Trento, 1999. Springer.

[BGMNT07] D. Baelde, A. Gacek, D. Miller, G. Nadathur, and A. Tiu. The Bedwyr system for model checking over syntactic expressions. In F. Pfenning, editor, *21th Conference on Automated Deduction (CADE)*, LNAI 4603, pp. 391–397. Springer, 2007.

[BSV08] D. Baelde, Z. Snow, and A. Viel. Taci: an interactive theorem proving framework. Active development of prototype, 2008.

[Gac08] A. Gacek. The Abella interactive theorem prover (system description). In *Fourth International Joint Conference on Automated Reasoning*, 2008.