Intuitionistic Ancestral Logic

Liron Cohen[1] and Robert Constable[2]

[1] Tel Aviv University,Tel-Aviv, Israel
[2] Cornell University, Ithaca, NY, USA

Type theories implemented by strong proof assistants have become highly effective as specification languages for a wide range of computational tasks, from operating systems and compiler verification [8,4] to the synthesis of correct-by-construction distributed protocols [11]. These type theories are rich logical systems which are difficult to grasp all at once. It is therefore interesting to see how they can be built from the ground up, starting with First-Order Logic (FOL) as is the practice for set theory. This turns out to be quite challenging in the case of *constructive type theories*. In this work we take another step toward a standard explication of constructive type theory.

Pure First-Order Logic is one of the most widely studied systems of logic.[3] It is the base logic in which two of the most studied mathematical theories, Peano Arithmetic (PA) and Zermelo-Fraenkel set theory with choice (ZFC), are presented. The intuitionistic versions of these systems, $iFOL$, Heyting Arithmetic (HA), Intuitionistic ZF (IZF) [6] and the related CZF [1], are also well studied. These intuitionistic logics are important in constructive mathematics, linguistics, philosophy and especially in computer science. Computer scientists exploit the fact that intuitionistic theories can serve as programming languages [3,10] and that $iFOL$ can be read as an abstract programming language with dependent types.

We are interested in natural extensions of $iFOL$ that clearly reveal the duality between logic and programming, and can capture general logical principles that have applicable computational content. It is clear that reasoning effectively about programs requires having some version of a transitive closure operator so that one can describe such notions as the set of nodes reachable from a program's variable. *Ancestral Logic* (AL) is a well known extension of FOL (e.g., [2,5,9]) appropriate for defining the transitive closure of binary relations. In this work we develop an intuitionistic version of AL, iAL, as a refinement of AL and an extension of $iFOL$, capable of giving computational explanations of the same com-

[3] We use the term *pure* to indicate that equality, constants, and functions are not built-in primitives.

monly occurring fundamental notions. iAL is a *dependently typed abstract programming language* with computational functionality beyond $iFOL$ given by its realizer for the transitive closure, TC. We derive this operator from the natural type theoretic definition of TC using intersection. Many proofs in iAL turn out to have interesting computational content that exceeds that of $iFOL$ in ways of interest to computer scientists. We prove that iAL is sound with respect to constructive type theory by showing that provable formulas are *uniformly realizable*. Furthermore, we show that iAL subsumes Kleene Algebras with tests [7] and thus serves as a natural programming logic for proving properties of program schemes. We also extract schemes from proofs that iAL specifications are solvable.

References

1. Peter Aczel. The type theoretic interpretation of constructive set theory: Inductive definition. In *Logic, Methodology and Philosophy of Science VII*, pages 17–49. Elsevier Science Publishers, 1986.
2. Arnon Avron. Transitive closure and the mechanization of mathematics. In *Thirty Five Years of Automating Mathematics*, pages 149–171. Springer, 2003.
3. Joseph L. Bates and Robert L. Constable. Proofs as programs. *ACM Trans. Program. Lang. Syst.*, 7(1):113–136, January 1985.
4. Adam Chlipala. A Verified Compiler for an Impure Functional Language. In *POPL*, pages 93–106, 2010.
5. Liron Cohen. Ancestral logic and equivalent systems. Master's thesis, Tel Aviv University, Tel Aviv, Israel, 2010.
6. Harvey Friedman. The consistency of classical set theory relative to a set theory with intuitionistic logic. *The Journal of Symbolic Logic*, 38(2):pp. 315–319, 1973.
7. Dexter Kozen and Allegra Angus. Kleene algebra with tests and program schematology. Technical report, 2001.
8. Xavier Leroy. Formal certification of a compiler back-end or: programming a compiler with a proof assistant. volume 41, pages 42–54. ACM, 2006.
9. Immerman N. Reps T. Sagiv M. Srivastava S. Lev-Ami, T. and G. Yorsh. Simulating reachability using first-order logic with applications to verification of linked data structures. In *Automated Deduction*, volume 3632 of *Lecture Notes in Computer Science*, pages 99–115. Springer Berlin Heidelberg, 2005.
10. Per Martin-Löf. Constructive mathematics and computer programming. *Studies in Logic and the Foundations of Mathematics*, 104:153–175, 1982.
11. Vincent Rahli, Nicolas Schiper, Robbert Van Renesse, Mark Bickford, and Robert L. Constable. A diversified and correct-by-construction broadcast service. In *The 2nd International Workshop on Rigorous Protocol Engineering*, 2012.