6th International Conference on Applied Informatics
Eger, Hungary, January 27–31, 2004.

Type Systems and the Program Verification [*]

Zoltán Csörnyei

Department of Programming Languages and Compilers,
Eötvös Loránd University, Hungary
e-mail: csz@inf.elte.hu

Abstract

The famous slogan by Robin Milner said that ”well-typed programs do not go wrong”. This slogan essentially asserts the soundness of the type system of the programming language. This is the question whether the type system prevents us from writing meaningful and error-free programs.

The proof generation capabilities of proof construction systems based on type theory. The ground of the theory is the typed λ-calculus. The higher-order type system of higher-order subtyping, known as $F^{\omega}_{\leq}$, has been used as a core calculus for typed languages [1, 4]. There are practical type inference mechanisms that are applicable to any explicitly typed polymorphic language [5]. The most commonly used methods are the Hindley-Milner system for polymorphic type inference and the Milner-Mycroft algorithm for polymorphic recursion.

The Curry-Howard isomorphism is a correspondence between type systems and the intuitionistic logic: ”types are formulas, and expressions are proofs”. Types correspond to formulas and the term E of type T correspond to a proof of the formula T where E is a representation, or encoding, of the proof. For instance, minimal propositional logic corresponds to simply typed λ-calculus, first-order logic corresponds to dependent types, second-order logic corresponds to polymorphic types [6].

Program verification deals with the question whether a triple $\{Pre\}P\{Post\}$ is consistent. This can be formally defined as $\forall s.(Pre \Rightarrow wp(P, Post))$. Type systems allow to express program properties which are automatically verified.

Techniques for formally specifying, understanding and verifying program behavior are available, but the program proving is very expensive. Type systems for program languages are well studied, and there are efforts to refine type systems to allow rich classes of program properties to be expressed and to combine ideas of type theories, verification and interpretation [2, 3].

Categories and Subject Descriptors: D.2.4 [Software Engineering]: Software/Program Verification - *Formal methods*; F.4.1 [Mathematical Logic and Formal Languages]: Mathematical logic - *Lambda calculus and related systems*; F.4.1 [Mathematical Logic and Formal Languages]: Mathematical logic - *Proof theory*;

Key Words and Phrases: λ-calculus, $F^{\omega}_{\leq}$, proof theory, type system, verification

[*]Supported by the Hungarian National Science Research Grant (OTKA), Grant Nr. T037742.

References

[1] Csörnyei, Z.: *Type Systems*, Lecture Notes (2003), http://people.inf.elte.hu/csz (In Hungarian)

[2] Dunfield, J., Pfenning, F.: Tridirectional Typechecking, in *POPL'04*, January 14-16, 2004, Venice, Italy

[3] Harper, R., Pfenning, F.: *Type Refinements*, Project Description, 2001. http://www-2.cs.cmu.edu/1triple/triple.pdf

[4] Pierce, B.C.: *Types and Programming Languages*, The MIT Press, 2002.

[5] Schwartzbach, M.I.: Polymorphic Type Inference *BRICS Lecture Series*, LS-95-3 (1995)

[6] Sørensen, M.H.B., Urzyczyn, P.: *Lectures on Curry-Howard Isomorphism*, Lecture Notes, University of Copenhagen, University of Warsaw (1999).