Reducibility method: an overview

Silvia Ghilezan*
Faculty of Technical Sciences
University of Novi Sad, Serbia

This is an overview of the development and application of the reducibility method in different aspects of logic and computation. The results are obtained over the last twenty years in collaboration with Henk Barendregt, Mariangiola Dezani-Ciancaglini, Daniel Dougherty, Jelena Ivetić, Pierre Lescanne, Silvia Likavec and Viktor Kunčak.

The reducibility method is a well known framework for proving reduction properties of terms typeable in different type systems. It was introduced by Tait in [10] for proving the strong normalization property for the simply typed lambda calculus. The main idea of this method is to relate terms typeable in a certain type system and terms satisfying certain reduction properties such as strong normalisation, head normalisation etc. Emerging from these proofs, the reducibility method became a widely accepted technique for proving various reduction properties of terms typeable in different type systems. This method was used to prove strong normalization of polymorphic (second-order) lambda calculus, intersection type systems, calculus with explicit substitutions and various other type systems. The reader is referred to [1], a comprehensive reference book on type systems.

The basic concept of the method can be represented by a unary predicate $P_\alpha(t)$, which means that a term t typeable by α satisfies the property P. To this aim types are interpreted as suitable sets of terms called saturated or stable sets. Then, the soundness of type assignment is obtained with respect to these interpretations. A consequence of soundness is that every term typeable in the type system belongs to the interpretation of its type. This is an intermediate step between the terms typeable in the given type system and terms satisfying the considered property P. In general, the principal notions of the reducibility method are:

- type interpretations (based on the considered property P);
- term valuations;
- saturation and closure conditions;
- soundness of the type assignment.

*Partially supported by the Ministry of Education Science and Technological Development of Serbia, projects ON174026 and III44006.

In [4] the reducibility method is applied to completely characterise the strongly normalizing lambda terms in the lambda calculus with intersection types. Suitable modifications of the reducibility method lead to uniform proofs of other reduction properties. An overview can be found in [6]. In [2] the reducibility method is applied to characterise normalising, head normalising and weak head normalising terms as well as their persistent versions. In [5] the reducibility is developed for a resource aware term calculus.

In the setting of classical logic, the reducibility method is not well suited to prove strong normalization for $\lambda\mu$-calculus, the simply typed classical term calculus. The symmetric candidates technique used to prove strong normalisation employs a fixed-point technique to define the reducibility candidates in [3].

Extending on the reducibility method, *logical relations* were introduced by Statman in [9] to proof the confluence (the Church-Rosser property) of $\beta\eta$-reduction of the simply typed λ-terms. It became a well-known method for proving confluence and standardisation in various type systems. Similarly to the reducibility method, the key notions are type interpretations. Logical relations in turn is a method based on binary relations $R_\alpha(t, t')$, which relate terms t and t' typeable by the type α that satisfy the relation R. Types are then interpreted as admissible relations.

In programming languages it is often necessary to relate terms either from the same language or from different languages in order to show their equivalence. To this aim logical relations became a powerful tool in programming languages, see Pierce [8]. Some of the most important applications.

- Observational equivalence: logical relations prove that terms obtained by optimisation are equivalent.
- Compiler correctness: logical relations are employed to relate the source and target language.
- Security information flow: logical relations prove that the system prevents high security data to leak in low security output.

References

[1] H.P. Barendregt, W. Dekkers, R. Statman, Lambda Calculus with Types, Cambridge University Press, 2013.

[2] M. Dezani-Ciancaglini, S. Ghilezan and S. Likavec, Behavioural inverse limit models, Theoretical Computer Science 316:49-74 (2004).

[3] D. Doughert, S. Ghilezan and P. Lescanne, Characterizing strong normalization in the Curien-Herbelin symmetric lambda calculus: extending the Coppo-Dezani heritage, Theoretical Computer Science 398:114-128 (2008).

[4] S. Ghilezan, Strong normalization and typability with intersection types. Notre Dame Journal of Formal Logic 37:44–52 (1996).

[5] S. Ghilezan, J. Ivetić, P. Lescanne, S. Likavec, Intersection Types for the Resource Control Lambda Calculi. ICTAC 2011, Lecture Notes in Computer Science 6916:116-134 (2011).

[6] S. Ghilezan and S. Likavec, Reducibility: A Ubiquitous Method in Lambda Calculus with Intersection Types, Electronic Notes in Theoretical Computer Science 70 (2003).

[7] S. Ghilezan, V. Kuncak, Confluence of Untyped Lambda Calculus via Simple Typesi. ICTCS 2001, Lecture Notes in Computer Science 2206:38-49 (2001).

[8] B. Pierce, Types and Programming Languages, MIT Press, 2002.

[9] R. Statman, Logical relations and the typed λ-calculus. Information and Control 65:85-97 (1985).

[10] W. W. Tait, Intensional interpretations of functionals of finite type I. Journal of Symbolic Logic 32:198–212 (1967).