Types as grammars

Gil Silva
LASIGE, University of Lisbon
Portugal

Bernardo Almeida
LASIGE, University of Lisbon
Portugal

Diana Costa
LASIGE, University of Lisbon
Portugal

Andreia Mordido
LASIGE, University of Lisbon
Portugal

Diogo Poças
Instituto de Telecomunicações
University of Lisbon
Portugal

Vasco T. Vasconcelos
LASIGE, University of Lisbon
Portugal

Abstract

Type equivalence. At the heart of every compiler lies a type equivalence algorithm. While this algorithm is relatively straightforward for nominal type systems, where simple checks on type names suffice, it gets more complex in structural type systems, where the structure of types must correspond in a certain sense. For the simple types of $\lambda^{\rightarrow}$ [5], this algorithm merely descends on their abstract syntax tree: there is, after all, little difference between what is meant by structure and by syntax. But this gap builds up as we move along increasingly expressive type systems:

1. System F [8, 14] introduces variables and quantification, requiring α-equivalence;
2. System F^{μ} [7] introduces equirecursion, requiring an equivalence relation that is preserved under unfolding;
3. System F^{μ}_{ω} [4] introduces type-level abstraction and application, requiring $\alpha/\beta/\eta$-equivalence.

Type equivalence in F^{μ}_{ω} is at least as expressive as equivalence for deterministic context-free languages, for which no practical algorithms are known; however, if recursion is restricted to proper types—$F^{\mu*}_{\omega}$—the complexity is reduced to that of equivalence for regular languages [4].

Why do we need grammars then? Simply because we are dealing with an even more expressive system, obtained by incorporating $F^{\mu*}_{\omega}$ into FreeST [1, 2, 6], a functional programming language based on F^{μ} with *session types*. This type discipline provides safe concurrency by describing and enforcing structured message-passing protocols on heterogeneous and bidirectional channels [9, 10, 15]. Furthermore, session types in FreeST are context-free, allowing the sequential composition of arbitrary protocols [16]. We consequently progress in the hierarchy to:

4. System $F^{\mu*;}_{\omega}$ introduces context-free session types, requiring an equivalence relation that respects the algebraic properties of sequential composition.

Introducing sequential composition lifts the restriction to tail recursion, but in turn precludes the use of a least fixed-point construction to decide type equivalence for context-free session types. The type equivalence problem in this setting is reduced to the bisimilarity of *simple grammars* [11], an amenable subset of context-free grammars. However, $F^{\mu*}_{\omega}$ types and context-free session types cannot be dealt with separately: sessions may carry functions, and functions may act on sessions. As such, we need a type equivalence algorithm that seamlessly deals with both. Reducing type equivalence in $F^{\mu*;}_{\omega}$ to simple grammar bisimilarity is the way to go.

How do we do it? First, we need an appropriate notion of bisimilarity for $F^{\mu*;}_{\omega}$ types that subsumes the standard type equivalence relation for $F^{\mu*}_{\omega}$ types. This is defined by means of a weak bisimulation on types that takes into account bindings, recursion, reduction and sequential composition [12].

Next, we define a procedure to convert types into a simple grammar. At the core of the procedure lies a function, *word*, that maps a type to a sequence of nonterminal symbols, introducing their corresponding productions in the process. This conversion procedure should be fully abstract: a type and its corresponding *word* should have the exact same transitions. This guarantees soundness and completeness for grammar bisimilarity with respect to type bisimilarity [12].

Finally, given two types, we decide whether their corresponding *word*s are bisimilar using a simple grammar bisimilarity algorithm. The goal of such algorithms is finding a finite set of pairs of nonterminals, called a basis, from which a bisimulation, even if infinite, can be induced. Throughout the development of FreeST, two algorithms have been used. The first, proposed by Almeida et al. [3], took a basis-refining approach and ran in 2-EXPTIME. The current one, proposed by Poças et al. [13], takes a basis-updating approach and runs in polynomial time; it is due to appear in print, along with an implementation as a standalone Haskell library.

WITS '26, January 17, 2026, Rennes, France

ACM ISBN 978-1-4503-XXXX-X/2026/06
https://doi.org/XXXXXXX.XXXXXXX

Acknowledgments. This work was supported by FCT - Fundação para a Ciência e a Tecnologia, I.P., through the LASIGE Research Unit, ref. UID/00408/2025, and Instituto de Telecomunicações, ref. UID/50008/2025.

References

[1] Bernardo Almeida, Andreia Mordido, Peter Thiemann, and Vasco T. Vasconcelos. 2022. Polymorphic lambda calculus with context-free session types. *Inf. Comput.* 289, Part (2022), 104948. https://doi.org/10.1016/J.IC.2022.104948

[2] Bernardo Almeida, Andreia Mordido, and Vasco T. Vasconcelos. 2019. FreeST: Context-free Session Types in a Functional Language. In *PLACES (EPTCS, Vol. 291)*. 12–23. https://doi.org/10.4204/EPTCS.291.2

[3] Bernardo Almeida, Andreia Mordido, and Vasco T. Vasconcelos. 2020. Deciding the Bisimilarity of Context-Free Session Types. In *TACAS (LNCS, Vol. 12079)*. Springer, 39–56. https://doi.org/10.1007/978-3-030-45237-7_3

[4] Yufei Cai, Paolo G. Giarrusso, and Klaus Ostermann. 2016. System F-omega with equirecursive types for datatype-generic programming. In *POPL*. ACM, 30–43. https://doi.org/10.1145/2837614.2837660

[5] Alonzo Church. 1940. A Formulation of the Simple Theory of Types. *J. Symb. Log.* 5, 2 (1940), 56–68. https://doi.org/10.2307/2266170

[6] freest 2019. The FreeST programming language. https://freest-lang.github.io/.

[7] Nadji Gauthier and François Pottier. 2004. Numbering matters: first-order canonical forms for second-order recursive types. In *ICFP*. ACM, 150–161. https://doi.org/10.1145/1016850.1016872

[8] J.-Y. Girard. 1971. Une extension de l'interpretation fonctionnelle de Gödel à l'analyse et son application à l'eliminnation des coupures dans l'analyse et la théorie des types. In *Proc. 2nd Scandinavian Logic Syrup*, J. F. Fenstad (Ed.). North-Holland, 63–92.

[9] Kohei Honda. 1993. Types for Dyadic Interaction. In *CONCUR (LNCS, Vol. 715)*. Springer, 509–523. https://doi.org/10.1007/3-540-57208-2_35

[10] Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. 1998. Language Primitives and Type Discipline for Structured Communication-Based Programming. In *ESOP (LNCS, Vol. 1381)*. Springer, 122–138. https://doi.org/10.1007/BFb0053567

[11] A. J. Korenjak and John E. Hopcroft. 1966. Simple Deterministic Languages. In *SWAT*. IEEE Computer Society, 36–46. https://doi.org/10.1109/SWAT.1966.22

[12] Diogo Poças, Diana Costa, Andreia Mordido, and Vasco T. Vasconcelos. 2023. System F^{μ}_{ω} with Context-free Session Types. In *ESOP (LNCS, Vol. 13990)*. Springer, 392–420. https://doi.org/10.1007/978-3-031-30044-8_15

[13] Diogo Poças, Gil Silva, and Vasco T. Vasconcelos. 2025. Simple grammar bisimilarity, with an application to session type equivalence. arXiv:2407.04063

[14] John C. Reynolds. 1974. Towards a theory of type structure. In *Programming Symposium, Proceedings Colloque sur la Programmation, Paris, France, April 9-11, 1974 (LNCS, Vol. 19)*, Bernard J. Robinet (Ed.). Springer, 408–423. https://doi.org/10.1007/3-540-06859-7_148

[15] Kaku Takeuchi, Kohei Honda, and Makoto Kubo. 1994. An Interaction-based Language and its Typing System. In *PARLE (LNCS, Vol. 817)*. Springer, 398–413. https://doi.org/10.1007/3-540-58184-7_118

[16] Peter Thiemann and Vasco T. Vasconcelos. 2016. Context-free session types. In *ICFP*. ACM, 462–475. https://doi.org/10.1145/2951913.2951926