Towards a translation from $\mathbb{K}$ to Dedukti

Amélie Ledein[1*], Valentin Blot[1], Catherine Dubois[2]

[1] Laboratoire Méthodes Formelles, Inria, Université Paris-Saclay
[2] Samovar, ENSIIE

In this talk, we present a shallow embedding of the $\mathbb{K}$ formalism allowing the definition of programming languages semantics, in the $\lambda\Pi$-calculus modulo theory. We propose a solution that makes use of the intermediate representation provided by the $\mathbb{K}$ compiler, i.e. Kore and relies on the rewriting engine of Dedukti in order to be able to execute a program in the translated formalism as it is possible to do with the execution tool provided by the $\mathbb{K}$ framework.

$\mathbb{K}$ presentation. $\mathbb{K}$ [2] is a semantical framework for formally describing the semantics of programming languages. It is also an environment that offers various tools to help programming with the languages specified in the formalism (Figure 1). It is for example possible to execute programs and to check some properties on them, using the automatic theorem KProver tool [10]. $\mathbb{K}$ is based on a theory of Matching Logic [9, 10, 6, 5, 4], named Kore, a 1st order untyped classic logic with an application between formulas and, fixed point, equality and typing operators, as well as an operator similar to the "next" operator of temporal logics. This last operator allows the semantics of programs to be encoded by rewriting. Kore is composed of the equality theory, the sort theory and the rewriting one.

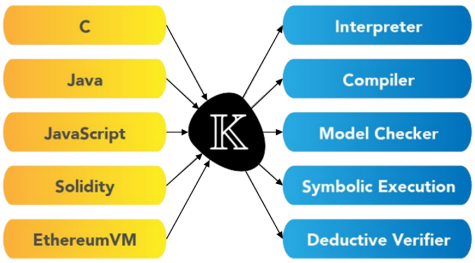

Figure 1: Pipeline of $\mathbb{K}$

Dedukti presentation. Dedukti [3] is a logical framework à la LF allowing the interoperability of proofs between different formal proof tools, as Coq or PVS (Figure 2). It has import and export plugins for proof systems as various as Coq, PVS or Isabelle/HOL. Dedukti is based on the $\lambda\Pi$-calculus modulo theory ($\lambda\Pi\equiv_{\mathcal{T}}$), an extension of the type theory by adding rewriting rules [8] in the conversion relation, introduced by Cousineau and Dowek [7]. The flexibility of this logical framework allows to encode many theories like 1st order logic or simple type theory.

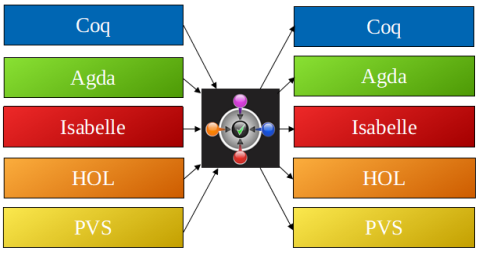

Figure 2: Pipeline of Dedukti

*The author is funded by Digicosme.

Contributions. In this talk, we present KaMeLo [1], a tool for translating from 𝕂 to Dedukti. This tool has the longer term goal of allowing the verification of proofs made within 𝕂, and the reuse of formal semantics of programming languages as well as properties of their programs in many proof systems, via Dedukti.

𝕂 is a rewrite-based framework in which programming languages semantics can be defined using configurations, computations and rules. Formally, a configuration is a multi-set of potentially nested labelled cells, e.g. a cell containing a program to be execute, another corresponding to an environment and a last one which is a memory store. Rewriting rules specify the evolution of a configuration during the execution. 𝕂 provides the user with many facilities to write a semantic definition, e.g. attributes to define a left to right evaluation strategy or an ellipsis like ... to indicate that there is no change in the rest of a configuration. However whatever the style used in the 𝕂 definition, in the Kore produced file everything is a plain and more standard form. Our translation from 𝕂 to Dedukti, is done via Kore, to make use of this *standardisation* that we assume correct.

𝕂 and Dedukti have the common point of being both based on rewriting, where this can be applied anywhere in a term, and can be non-linear. However, Dedukti allows higher order, while 𝕂 supports only 1st order. Conversely, conditional rewriting and modulo ACUI are not supported by Dedukti, unlike 𝕂. These rewriting rules are preserved in Dedukti. The next two paragraphs deal with more specific cases concerning the translation of rewriting rules.

Concerning conditional rewriting the general idea of the encoding, inspired from [11], is, for a symbol defined with conditional rules, to generate a fresh symbol with the same arguments as the initial symbol, plus as many arguments as there are conditions. Once these conditional arguments have been instantiated, it is possible to evaluate them, and then to rewrite the term as a whole according to the evaluation of these arguments.

Whatever the way used to define an evaluation strategy in 𝕂, conditional rewriting rules will be generated, using a *K computation*, i.e. a potentially nested list of computations to be performed sequentially, and *freezers*. Intuitively, a freezer is a symbol which encapsulate the part of the computation that shouldn't modify, i.e. the queue of the K computation, while waiting for the head of the K computation to be evaluated. These generated rules have the particularity of not being able to be transformed thanks to our variant of Viry's algorithm, under penalty of losing the confluence property. In this particular case, we have opted for the specialisation of the left-hand side terms of the rules, i.e. to refine the pattern-matching in order to precise the desired type of the left-hand side terms of the rules. To do this, we use the subtyping relations, but also the inductive structure.

Conclusion & Perspectives. Thanks to the logical framework Dedukti, the objective of this work is to verify formal proofs about the semantics of programming languages, described in the semantical framework 𝕂, and to reuse of such proofs in different proof tools.

Initially, we were interested in the translation in Dedukti of semantics written in 𝕂, in order to be able to execute in Dedukti programs written in the language described by the semantics. We have relied on the theory of Matching Logic, named Kore, without seeking to certify it. We therefore assume that the Kore file produced from a 𝕂 semantic is correct. This work required understanding the translation of a 𝕂 semantics into Kore but also being able to translate conditional rules into unconditional rules. For now, the translator KaMeLo, a tool for translating 𝕂 to Dedukti, currently under development, translates the syntax of the language, the configurations and symbols, some attributes and the rewriting rules.

The verification of the KProver proof objects, as well as the encoding of the theoretical foundations of 𝕂 in those of Dedukti will be future work. The translation presented here is nevertheless necessary to run a program and will be reused for the verification of proofs.

References

[1] GitLab of KaMeLo. https://gitlab.com/semantiko/kamelo.

[2] Website of $\mathbb{K}$. https://kframework.org/.

[3] A. Assaf, G. Burel, R. Cauderlier, D. Delahaye, G. Dowek, C. Dubois, F. Gilbert, P. Halmagrand, O. Hermant, and R. Saillard. Expressing theories in the $\lambda\Pi$-calculus modulo theory and in the Dedukti system. In *TYPES: Types for Proofs and Programs*, Novi SAd, Serbia, May 2016.

[4] X. Chen, D. Lucanu, and G. Roşu. Matching Logic Explained. Technical Report http://hdl.handle.net/2142/107794, University of Illinois at Urbana-Champaign and Alexandru Ioan Cuza University, July 2020.

[5] X. Chen and G. Roşu. Applicative Matching Logic: Semantics of K. Technical Report http://hdl.handle.net/2142/104616, University of Illinois at Urbana-Champaign, July 2019.

[6] X. Chen and G. Rosu. Matching mu-Logic: Foundation of K Framework (Invited Paper). page 4 pages, 2019. Artwork Size: 4 pages Medium: application/pdf Publisher: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany Version Number: 1.0.

[7] D. Cousineau and G. Dowek. Embedding Pure Type Systems in the Lambda-Pi-Calculus Modulo. In *TLCA*, 2007.

[8] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 243–320, 1990.

[9] G. Roșu and T. F. Șerbănuță. An overview of the K semantic framework. *The Journal of Logic and Algebraic Programming*, 79(6):397–434, Aug. 2010.

[10] A. Stefănescu, D. Park, S. Yuwen, Y. Li, and G. Roşu. Semantics-based program verifiers for all languages. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 74–91, Amsterdam Netherlands, Oct. 2016. ACM.

[11] P. Viry. Elimination of Conditions. *Journal of Symbolic Computation*, 28(3):381–401, 1999.