Rechecking KProver proof objects into Dedukti

Amélie LEDEIN

In this talk, we present two works in progress to recheck KProver proof objects into Dedukti, restricting ourselves to one specific reachability property: concrete execution.

The first one is to generate proof objects from the trace provided by the KProver and then check them in Dedukti. This approach requires finding an encoding of Matching Logic in Dedukti, and then automatically encoding the $\mathbb{K}$ semantics as well as the proof objects in this encoding.

The second approach builds on the generation of proof objects already done for Metamath. This approach requires translating the encoding of Matching Logic into Metamath, within Dedukti, and then automatically translating the Metamath proof objects into Dedukti.

$\mathbb{K}$ presentation. $\mathbb{K}$ 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 prover named KProver [11]. $\mathbb{K}$ is based on a theory of Matching Logic [10, 5, 4, 3], 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. Kore is composed of the equality, the sort and the rewriting theories.

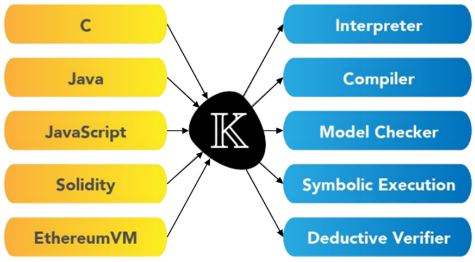

Figure 1: Pipeline of $\mathbb{K}$

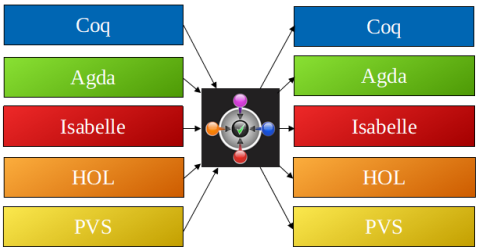

Figure 2: Pipeline of Dedukti

Dedukti presentation. Dedukti [1] is a logical framework 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 [7] in the conversion relation, introduced by Cousineau and Dowek [6]. The flexibility of this logical framework allows to encode many theories like 1st order logic or simple type theory.

The direct approach. This approach is to generate proof objects from the trace provided by the KProver and then check them in Dedukti. To achieve this goal, we need to encode Matching Logic into Dedukti, but also automatically translated the $\mathbb{K}$ semantics as well as the trace into this encoding.

This approach is based on the work of formalisation carried out within Metamath [2], as well as a work about the translation of $\mathbb{K}$ semantics into Dedukti, via Kore, making it possible to execute $\mathbb{K}$ semantics in Dedukti [8].

This work involved understanding the logical foundations of $\mathbb{K}$, i.e., Matching Logic and the particular Matching Logic theory called Kore, as well as how the $\mathbb{K}$ semantics of an language L can be automatically translated into a Kore extension theory Γ^L.

The approach via Metamath. The second approach builds on the generation of proof objects already done for Metamath. This approach requires, also, translating the encoding of Matching Logic into Metamath, within Dedukti, and then automatically translating the Metamath proof objects into Dedukti. Using an example from formal number theory of Mendelson, we detail how to construct a λ-term from a Metamath proof.

The language of Metamath is composed only of declarations of constants (line 1), variables (line 2), axioms (lines 8-13 and 16) and proofs (See below). Several types of hypothesis are possible: typing or floating (lines 3-7), logical or essential (lines 14-15) or restriction (with the keyword `$d`).

```
1   $c 0 + = -> ( ) term wff |- $.
2   $v t r s P Q $.
3   tt $f term t $.
4   tr $f term r $.
5   ts $f term s $.
6   wp $f wff P $.
7   wq $f wff Q $.
8   tze $a term 0 $.
9   tpl $a term ( t + r ) $.
10  weq $a wff t = r $.
11  wim $a wff ( P -> Q ) $.
12  a1 $a |- ( t = r -> ( t = s -> r = s ) ) $.
13  a2 $a |- ( t + 0 ) = t $.
14  ${    min $e |- P $.
15        maj $e |- ( P -> Q ) $.
16        mp $a |- Q $.  $}
```

The proof mechanism of Metamath is simple since it is only based on substitution and the notion of stack. Indeed, the stack is used to store the different pieces of construction of the proof, whereas substitution allows several pieces to be combined. Metamath considers that a proof is correct if, at the end of its verification, the stack contains a single element which is syntactically equal to the statement.

This principle of proof checking is very simple but gives rise to very verbose and therefore difficult to read proofs, as shown in the proof on the right. Moreover, the steps of arity checking, typing checking, and correct application of the rules are completely mixed up.

```
th1 $p |- t = t $=
   tt tze tpl tt weq tt tt weq tt a2
   tt tze tpl tt weq tt tze tpl tt weq
   tt tt weq wim tt a2 tt tze tpl tt tt a1
   mp mp $.
```

The general idea for constructing a λ-term associated with a Metamath proof is to follow exactly the same verification mechanism as Metamath: the steps of arity and typing checking are skipped, while the assembly proof steps use axioms already translated in Dedukti. For our example, we obtain the λ-term:
λ **t. mp (a2 t) (mp (a2 t) (a1 (t + 0) t t)**

For Dedukti to check this term, it will be necessary that, manually, the constants 0, +, =, ->, and the axioms `a1`, `a2`, and `mp` have been translated by symbols in Dedukti.

Conclusion & Perspectives The direct approach to Dedukti constitutes a part of the author's thesis work. The approach via Metamath is the subject of a 3-month internship supervised by the author.
The approaches we present here rely heavily on the formalisation work carried out within Metamath [2]. Later, we would like to extend our approaches to symbolic execution, such as [9]

References

[1] 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 λΠ-calculus modulo theory and in the Dedukti system. In TYPES: Types for Proofs and Programs, Novi SAd, Serbia, May 2016.

[2] X. Chen, Z. Lin, M.-T. Trinh, and G. Roşu. Towards a Trustworthy Semantics-Based Language Framework via Proof Generation. In Proceedings of the 33rd International Conference on Computer-Aided Verification. ACM, July 2021.

[3] 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.

[4] 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.

[5] X. Chen and G. Roşu. Matching mu-logic. In Proceedings of the 34$^{\text{th}}$ Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'19), pages 1–13. ACM/IEEE, June 2019.

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

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

[8] A. Ledein, V. Blot, and C. Dubois. Vers une traduction de K en Dedukti. In JFLA 2022 - Journées Francophones des Langages Applicatifs, Saint-Médard-d'Excideuil, France, 2022.

[9] X. T. M.-T. W. J. R. G. Lin, Zhengyao; Chen. Making Formal Verification Trustworthy via Proof Generation. Technical Report http://hdl.handle.net/2142/112785, November 2021.

[10] 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.

[11] 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.