Ekstrakto A tool to reconstruct Dedukti proofs from TSTP files (extended abstract)

Mohamed Yacine El Haddad, Guillaume Burel, Frédéric Blanqui

EKSTRAKTO
A tool to reconstruct *Dedukti* proofs from TSTP files (extended abstract)

Mohamed Yacine El Haddad
CNRS

Guillaume Burel
ENSIIE

Frédéric Blanqui
INRIA

LSV, CNRS, ENS Paris-Saclay, Université Paris-Saclay

Proof assistants often call automated theorem provers to prove subgoals. However, each prover has its own proof calculus and the proof traces that it produces often lack many details to build a complete proof. Hence these traces are hard to check and reuse in proof assistants. DEDUKTI is a proof checker whose proofs can be translated to various proof assistants: Coq, HOL, Lean, Matita, PVS. We implemented a tool that extracts TPTP subproblems from a TSTP file and reconstructs complete proofs in DEDUKTI using automated provers able to generate DEDUKTI proofs like ZenonModulo or ArchSAT. This tool is generic: it assumes nothing about the proof calculus of the prover producing the trace, and it can use different provers to produce the DEDUKTI proof. We applied our tool on traces produced by automated theorem provers on the CNF problems of the TPTP library and we were able to reconstruct a proof for a large proportion of them, significantly increasing the number of DEDUKTI proofs that could be obtained for those problems.

1 Introduction

In order to discharge more burden from the users of interactive theorem provers, and thus to widen the use of these tools, it is crucial to automate them more. To achieve this goal, in the process of checking the validity of formulas, proof assistants could use an external theorem prover to automate their tasks and obtain a proof of a specific formula. Once a proof is found, the proof assistant applies this proof on the current goal and tells the user that all is done in background. However, this can work only if the prover builds a complete proof that is easily checkable by the proof assistant. We distinguish two families of automated theorem provers: some provers, like *ZenonModulo* [5] and *ArchSAT* [3], output complete proofs but are not very efficient to find a proof; others, like *E prover* [7] and *ZipperPosition* [4], are more powerful but return only proof traces, i.e. proofs with less details.

In this paper we are interested in first-order automated theorem provers which can return TSTP [8] traces. We will use DEDUKTI [1] as proof checker because DEDUKTI files can be translated to many other proof assistants (Coq, HOL, Lean Matita, PVS) [10].

We start by presenting the TPTP/TSTP formats with an example. Then, we describe how proofs and formulas are encoded in DEDUKTI. We then present our solution implemented in a tool named EKSTRAKTO in two steps: extraction of sub-problems and proof reconstruction. Finally, we conclude and give some perspectives.

2 TPTP

TPTP [8] is a standard library of problems to test automated theorem provers [9]. Each TPTP file represents a problem in propositional, first-order or higher-order logic. We distinguish the type of formulas

Submitted to:
PxTP 2019

by using one of the keywords: CNF, FOF, TFF and THF, corresponding respectively to mono-sorted first-order formulas in clausal normal form, general mono-sorted first-order formulas, typed first-order formulas, and typed higher-order formulas.

In this work, we restrict our attention to CNF formulas since their proofs use logical consequences only, which is not the case of FOF formulas (e.g. Skolemisation).

Apart from an include instruction, each line in a TPTP file is a declaration of a formula given with its role, e.g. axiom, hypothesis, definition or conjecture:

```
cnf(name, role, formula, information).
```

TSTP [8] is a library of solutions to TPTP problems. In this paper, we call a TSTP file a trace. It is obtained after running an automated theorem prover on a TPTP problem. The syntax used in a TSTP file is the same as TPTP except for the content of the *information* field. This field contains general information about how the current formula is obtained. Here is the grammar used to describe a source in the *information* field:

```
<source>              :== <dag_source> | [ <sources> ] | ...
<dag_source>          :== <name> | inference(..., ..., <inference_parents>)
<inference_parents>   :== [] | [ <sources> ]
<sources>             :== <source> (, <source>)*
```

For our purpose only 3 cases are of interest as shown in the grammar above:

1) When it is the name of a formula previously declared.
2) When it is a list of several sources:

   ```
   [s_0, s_1, ..., s_n]
   ```

3) When it is an inference:

   ```
   inference(name, infos, [s_0, s_1, ..., s_n])
   ```

The name of the inference refers to the name of the rule used by the prover to prove the current step. The *infos* field contains more information about the inference like status, inference name, etc. Note that each s_i is a source and therefore can contain sub-inferences.

Here is an example of a TSTP file obtained after running *E prover* on the TPTP problem SET001-1:

SET001-1.p

```
cnf(c_0,axiom,
    ( subset(X1,X2)
    | ~ equal_sets(X1,X2) )).
cnf(c_1,hypothesis,
    ( equal_sets(b,bb) )).
cnf(c_2,axiom,
    ( member(X1,X3)
    | ~ member(X1,X2)
    | ~ subset(X2,X3) )).
cnf(c_3,negated_conjecture,
    ( ~ member(element_of_b,bb) )).
cnf(c_4,hypothesis,
```

```
    ( member(element_of_b,b) )).
cnf(c_5,hypothesis,
    ( subset(b,bb) ),
    inference(spm,[status(thm)],[c_0,c_1])).
cnf(c_6,hypothesis,
    ( member(X1,bb)
    | ~ member(X1,b) ),
    inference(spm,[status(thm)],[c_2,c_5])).
cnf(c_7,negated_conjecture,
    ( $false ),
    inference(cn,[status(thm)],[inference(rw,[status(thm)],
    [inference(spm,[status(thm)],[c_3,c_6]),c_4])]),
    [proof]).
```

We can represent this trace as the following tree:

$$\dfrac{\vdash \texttt{Form(c\_3)} \qquad \dfrac{\vdash \texttt{Form(c\_2)} \qquad \dfrac{\vdash \texttt{Form(c\_0)} \qquad \vdash \texttt{Form(c\_1)}}{\vdash \texttt{Form(c\_5)}}\,\text{spm} \qquad \vdash \texttt{Form(c\_6)}}{\dfrac{\vdash \texttt{Form(c\_7)}}{\vdash \texttt{Form(c\_4)}}\,\text{cn}}\,\text{spm}}{}\,\text{rw}$$

where:

```
Form(c_0) = subset(X1,X2) | ~equal_sets(X1,X2)
Form(c_1) = equal_sets(b,bb)
Form(c_2) = member(X1,X3) | ~member(X1,X2) | ~subset(X2,X3)
Form(c_3) = ~member(element_of_b,bb)
Form(c_4) = member(element_of_b,b)
Form(c_5) = subset(b,bb)
Form(c_6) = member(X1,bb) | ~member(X1,b)
Form(c_7) = $false
```

3 First-order logic in DEDUKTI

DEDUKTI is a proof checker based on the $\lambda\Pi$-calculus modulo rewriting [1]. In DEDUKTI, one can declare (dependent) types and function symbols, and rewriting rules for defining these symbols. We describe how a formula and its proof are encoded in DEDUKTI using the *Curry-Howard* correspondence, i.e., we interpret formulas as types and their proofs as terms. In the following, we recall the encoding of first-order logic in DEDUKTI, as described in [1]. This encoding is used in *ZenonModulo*, which is an extension to rewriting of the automated theorem prover *Zenon* [2]. *ZenonModulo* outputs DEDUKTI files after having found a proof using the tableaux method. The following file defines the type of sorts, the type of terms, the type of formulas and then the type of proofs.

zen.lp

```
symbol sort : TYPE          // Dedukti type for sorts
symbol ι : sort             // default sort

symbol term : sort ⇒ TYPE   // Dedukti type for sorted terms

symbol prop : TYPE          // Dedukti type for formulas
symbol ⊥̇ : prop
symbol ⊤̇ : prop
symbol ∸ : prop ⇒ prop
symbol ∧̇ : prop ⇒ prop ⇒ prop
symbol ∨̇ : prop ⇒ prop ⇒ prop
symbol ⇒̇ : prop ⇒ prop ⇒ prop
symbol ∀̇ : ∀ a, (term a ⇒ prop) ⇒ prop
symbol ∃̇ : ∀ a, (term a ⇒ prop) ⇒ prop
symbol ≐ : ∀ a, term a ⇒ term a ⇒ prop

symbol Proof : prop ⇒ TYPE  // interprets formulas as types
rule Proof (⇒̇ &a &b) → Proof &a ⇒ Proof &b
// rewriting rule defining the type of proofs for ⇒̇
```

Now, for each TSTP file, we generate a Dedukti file defining its signature by declaring a Dedukti symbol f for each function symbol f of the TSTP file:

SET001-1.lp

```
symbol element_of_b : zen.term ι
symbol subset       : zen.term ι ⇒ zen.term ι ⇒  zen.prop
symbol b            : zen.term ι
symbol member       : zen.term ι ⇒ zen.term ι ⇒  zen.prop
symbol bb           : zen.term ι
symbol equal_sets   : zen.term ι ⇒ zen.term ι ⇒  zen.prop
```

Hence, every formula of first-order logic can be represented in DEDUKTI by using the function φ defined as follows:

$$\begin{aligned}
\varphi(x) &:= x \\
\varphi(f(t_1,t_2,\ldots,t_n)) &:= f\ \varphi(t_1)\ \varphi(t_2)\ \ldots\ \varphi(t_n) \\
\varphi(\bot) &:= \dot{\bot} \\
\varphi(\top) &:= \dot{\top} \\
\varphi(\neg A) &:= \dot{\neg}\varphi(A) \\
\varphi(A \wedge B) &:= \varphi(A)\ \dot{\wedge}\ \varphi(B) \\
\varphi(A \vee B) &:= \varphi(A)\ \dot{\vee}\ \varphi(B) \\
\varphi(A \Rightarrow B) &:= \varphi(A) \dot{\Rightarrow} \varphi(B) \\
\varphi(\forall x A) &:= \dot{\forall}_\iota(\lambda x, \varphi(A)) \\
\varphi(\exists x A) &:= \dot{\exists}_\iota(\lambda x, \varphi(A)) \\
\varphi(x = y) &:= \varphi(x) \doteq_\iota \varphi(y)
\end{aligned}$$

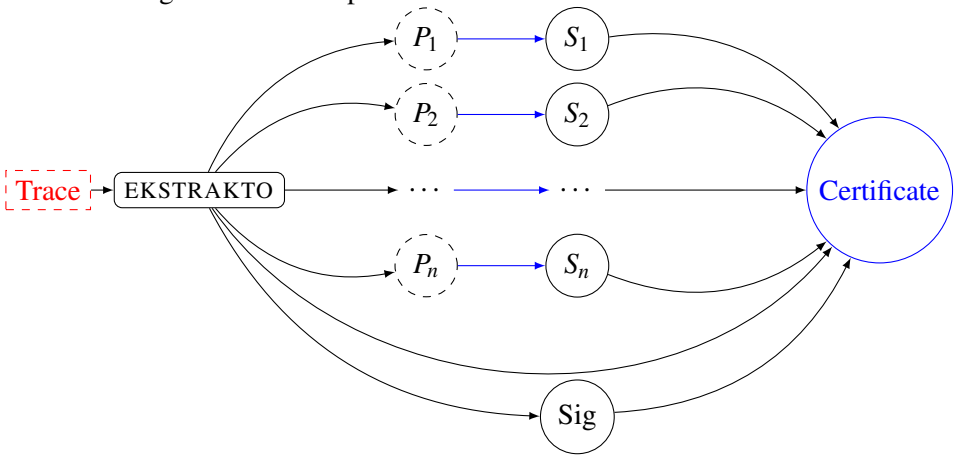
For example,

$$\varphi(\forall X_1, \forall X_2, s(X_1, X_2) \vee \neg e(X_1, X_2)) := \dot{\forall}_\iota(\lambda X_1, \dot{\forall}_\iota(\lambda X_2, (s\ X_1\ X_2)\ \dot{\vee}\ \dot{\neg}(e\ X_1\ X_2)))$$

For every formula A, its proof in DEDUKTI is a term that has the type $Proof(\varphi(A))$. One can define a similar embedding for proofs as the one we presented for first-order formulas, as shown in [1].

4 Architecture

In this section, we explain in details how EKSTRAKTO works. In order to produce a DEDUKTI proof from a TSTP file, EKSTRAKTO extracts a TPTP problem for each formula declaration containing at least one inference, and calls *ZenonModulo* (or any other automated prover producing DEDUKTI proofs, see discussion below) on each generated problem to get a DEDUKTI proof for this problem. If the external prover succeeds to find a proof of all the generated problems, then we combine those proofs in another DEDUKTI file to get a DEDUKTI proof of the whole TSTP file.



4.1 Extracting TPTP problems

To extract a TPTP problem from a trace step, we need to find the premises used in it. We define the function $\mathscr{P}$ which takes a TSTP source as input and returns the set of premises used by the prover:

$$\mathscr{P}(name) = \{name\}$$

$$\mathscr{P}([s_0, s_1, \ldots, s_n]) = \bigcup_{i=0}^{n} \mathscr{P}(s_i)$$

$$\mathscr{P}(inference(name, infos, [s_0, s_1, \ldots, s_n])) = \bigcup_{i=0}^{n} \mathscr{P}(s_i)$$

Note that if we have an inference t inside another one, say s, we will repeat the process for each sub-inference and omit s from the set of premises, i.e., if we represent an inference step by a proof tree we take only the leaves of this tree as premises.

We omit all information that is not needed (*status*, *name*, ...). In particular we do not consider the inference name field. Even if it could be used to fine-tune the problem, we prefer to ignore it in order to remain generic since the names are specific to the prover that produced the trace. Hence, we have:

$$\mathscr{P}(inference([inference([inference([c\_3, c\_6]), c\_4])])) = \{c\_3, c\_6, c\_4\}$$

After getting all the premises used for proving $\texttt{Form}(name)$, say $name_0, \ldots, name_k$, we generate the following TPTP problem:

$$\texttt{Form}(name_0) \Rightarrow \ldots \Rightarrow \texttt{Form}(name_k) \Rightarrow \texttt{Form}(name)$$

Note that the generated TPTP problem is a FOF formula. The reason of this choice is to keep the same formula when we combine the sub-proofs. If we generated a CNF problem, then we would need to negate the goal and it would be more complex to reconstruct the proof.

Since we are using FOF formulas in sub-problems that are obtained from a CNF trace, we need to quantify over each free variable to get a closed formula.

In our example, there are 3 steps (colored in blue in the file `SET001-1.p` above). EKSTRAKTO will generate the following 3 first-order formulas:

```
Form(c_0) ⇒ Form(c_1) ⇒ Form(c_5)

Form(c_2) ⇒ Form(c_5) ⇒ Form(c_6)

Form(c_3) ⇒ Form(c_6) ⇒ Form(c_4) ⇒ Form(c_7)
```

Each formula will be written in a separate TPTP file as follows:

c_5.p

```
fof(c_5, conjecture, (
        (![X1, X2] : (s (X1, X2)|~equal_sets (X1, X2)))
    =>  ((equal_sets (b, bb))
    =>  (subset (b, bb))))).
```

c_6.p

```
fof(c_6, conjecture,(
        (![X1, X2, X3] : (member (X1, X3)|~member (X1, X2)
                          |~subset (X2, X3)))
    =>  ((subset (b, bb))
    =>  (![X1] : (member (X1, bb)|~member (X1, b)))))).
```

c_7.p

```
fof(c_7, conjecture, (
        (~member (element_of_b, bb))
    =>  ((![X1] : (member (X1, bb)|~member (X1, b)))
    =>  ((member (element_of_b, b))
    =>  ($false))))).
```

4.2 Proof reconstruction

If the automated theorem prover succeeds to solve all the generated TPTP problems, then we can reconstruct a proof in DEDUKTI directly by using the proof tree of the trace that we are trying to certify and all the proofs of the sub-problems. The proof term of each sub-problem is irrelevant since it has the right type.

The global proof is reconstructed from each sub-proof. We just need to apply each proof term of a sub-proof to its premises by following the proof tree of the TSTP file. Indeed, the type of the sub-proof of $\texttt{Form}(name)$ using premises $name_0, \ldots, name_k$ is

$$\texttt{zen.Proof}\ (\Rightarrow\ \varphi(\texttt{Form}(name_0))\ (\Rightarrow\ \varphi(\texttt{Form}(name_1))\ \ldots\ (\Rightarrow\ \varphi(\texttt{Form}(name_k))\ \varphi(\texttt{Form}(name)))\ldots))$$

Thanks to the rule given in zen.lp in Section 3, this type is convertible to

$$\texttt{zen.Proof}\ (\varphi(\texttt{Form}(name_0)))\ \Rightarrow\ \ldots\ \Rightarrow\ \texttt{zen.Proof}\ (\varphi(\texttt{Form}(name_k)))\ \Rightarrow\ \texttt{zen.Proof}\ (\varphi(\texttt{Form}(name)))$$

Hence, the proof term of a sub-problem is a function whose arguments are proofs of the premises and which returns a proof of its conclusion. Since we are handling only CNF formulas, the proof that we want to reconstruct at the end is always a proof of $\bot$. Before applying those proof terms we need to declare our hypotheses. With our example file we get:

proof_SET001-1.lp

```
definition proof_trace
 (hyp_c_0 : zen.Proof (φ(Form(c_0))))
 (hyp_c_1 : zen.Proof (φ(Form(c_1))))
 (hyp_c_2 : zen.Proof (φ(Form(c_2))))
 (hyp_c_3 : zen.Proof (φ(Form(c_3))))
 (hyp_c_4 : zen.Proof (φ(Form(c_4))))
  : zen.Proof ⊥
  :=
  let lemma_c_5 = c_5.delta hyp_c_0 hyp_c_1           in
  let lemma_c_6 = c_6.delta hyp_c_2 lemma_c_5         in
  let lemma_c_7 = c_7.delta hyp_c_3 lemma_c_6 hyp_c_4 in
  lemma_c_7
```

where *delta* is the name of the proof term in each file.

All this has been implemented in a tool called EKSTRAKTO[1] consisting of 2,000 lines of OCaml.

5 Experiments

We run the *E prover* (version 2.1) on the set of CNF problems of TPTP library v7.2.0 (7922 files) with 2GB of memory space and a timeout of 5 minutes. We obtained 4582 TSTP files. On these TSTP files, EKSTRAKTO generated 362556 TPTP files. *ZenonModulo* generated a DEDUKTI proof for 90% of these files, *ArchSAT* generated 96% and the union of both produced 97% DEDUKTI proofs:

[1] https://github.com/elhaddadyacine/ekstrakto

Table 1: Percentage of DEDUKTI proofs on the 362556 extracted TPTP files

Prover	% TPTP
ZenonModulo	90%
ArchSAT	96%
ZenonModulo ∪ *ArchSAT*	97%

However, as it suffices that no DEDUKTI proof is found for only one TPTP file for getting no global proof, EKSTRAKTO can generate a complete proof for only 48% of TSTP files using *ZenonModulo*, 61% using *ArchSAT* and 72% using at least one of them:

Table 2: Percentage of DEDUKTI proofs on the 4582 TSTP files generated by *E prover*

Prover	% TSTP
ZenonModulo	48%
ArchSAT	61%
ZenonModulo ∪ *ArchSAT*	71%

Consequently, we are now able to produce 2189 DEDUKTI proofs from the TPTP library using *E prover* and *Zenon Modulo* (resp. 2793 using *E prover* and *ArchSAT* and 3285 using *E prover*, *Zenon Modulo* and *ArchSAT*), whereas under the same conditions, *Zenon Modulo* alone is only able to produce 1026 DEDUKTI proofs (resp. 500 for *ArchSAT* alone).

Sometimes, *ZenonModulo* and *ArchSAT* fail to find a proof even if the sub-problem is simpler than the main one. This is justified by the fact that the proof calculus used in *ZenonModulo* and *ArchSAT* is based on a different method from the one used in *E prover*. In fact, some steps that are trivial for a prover based on resolution or superposition may not be trivial for *ZenonModulo* or *ArchSAT* which use the tableaux method.

iProverModulo is another candidate to prove TSTP steps, but it performs some transformations before outputting a DEDUKTI proof. Therefore the proof reconstruction is hard in the sense that we need to justify each transformation made by *iProverModulo*.

6 Conclusion and perspectives

We have presented a tool that reconstructs proofs generated by first-order theorem provers. We described how proofs and formulas are represented in DEDUKTI and how we can implement a simple proof reconstruction.

The advantage of EKSTRAKTO is to be generic since it does not depend on the rules used by the automated prover to find the proof. Another advantage is the fact that the proofs are expressed in DEDUKTI, i.e., we can translate them to many other systems (Coq, HOL, Lean, Matita, PVS).

In our experiments, we used *ZenonModulo* and *ArchSAT* to prove each trace step since they are tools that produce DEDUKTI proof terms.

EKSTRAKTO should be extended to handle non-provable steps like Skolemisation. This latter technique could possibly be implemented using the method described in [6]. We should also be more generic, by supporting more features of TSTP like typed formulas and definitions introduced by the prover.

Acknowledgements. The authors thank the anonymous reviewers for their useful comments. This

research was partially supported by the Labex DigiCosme (ANR11LABEX0045DIGICOSME) operated by ANR as part of the program "Investissement d'Avenir" Idex ParisSaclay (ANR11IDEX000302).

References

[1] Ali Assaf, Guillaume Burel, Raphaël Cauderlier, Gilles Dowek, Catherine Dubois, Frédéric Gilbert, Pierre Halmagrand, Olivier Hermant & Ronan Saillard: *Dedukti: a Logical Framework based on the λΠ-Calculus Modulo Theory*. Available at `http://www.lsv.fr/~dowek/Publi/expressing.pdf`.

[2] Richard Bonichon, David Delahaye & Damien Doligez (2007): *Zenon : An Extensible Automated Theorem Prover Producing Checkable Proofs*. In: *Logic for Programming, Artificial Intelligence, and Reasoning, 14th International Conference, LPAR 2007, Yerevan, Armenia, October 15-19, 2007, Proceedings*, pp. 151–165, doi:10.1007/978-3-540-75560-9_13.

[3] Guillaume Bury, Simon Cruanes & David Delahaye (2018): *SMT Solving Modulo Tableau and Rewriting Theories*. Available at `https://hal.archives-ouvertes.fr/hal-02083232`.

[4] Simon Cruanes (2015): *Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond*. Theses, École polytechnique. Available at `https://hal.archives-ouvertes.fr/tel-01223502`.

[5] David Delahaye, Damien Doligez, Frédéric Gilbert, Pierre Halmagrand & Olivier Hermant (2013): *Zenon Modulo: When Achilles Outruns the Tortoise Using Deduction Modulo*. In Ken McMillan, Aart Middeldorp & Andrei Voronkov, editors: *Logic for Programming, Artificial Intelligence, and Reasoning*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 274–290, doi:10.1007/978-3-642-45221-5_20.

[6] Gilles Dowek & Benjamin Werner: *A constructive proof of Skolem theorem for constructive logic*. Available at `http://www.lsv.fr/~dowek/Publi/skolem.pdf`.

[7] Stephan Schulz (2013): *System Description: E 1.8*. In Ken McMillan, Aart Middeldorp & Andrei Voronkov, editors: *Proc. of the 19th LPAR, Stellenbosch*, *LNCS* 8312, Springer, doi:10.1007/978-3-642-45221-5_49.

[8] G. Sutcliffe (2017): *The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0*. *Journal of Automated Reasoning* 59(4), pp. 483–502, doi:10.1007/s10817-017-9407-7.

[9] Geoff Sutcliffe (2018): *The 9th IJCAR Automated Theorem Proving System Competition - CASC-J9*. *AI Commun.* 31(6), pp. 495–507, doi:10.3233/AIC-180773.

[10] François Thiré (2018): *Sharing a Library between Proof Assistants: Reaching out to the HOL Family*. In Frédéric Blanqui & Giselle Reis, editors: *Proceedings of the 13th International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, LFMTP@FSCD 2018, Oxford, UK, 7th July 2018.*, *EPTCS* 274, pp. 57–71, doi:10.4204/EPTCS.274.5.