Towards Formalization and Sharing of Atelier B Proofs with Dedukti

Claude Stolze, Olivier Hermant, Romain Guillaumé

Towards Formalization and Sharing of Atelier B Proofs with Dedukti

Claude Stolze[1], Olivier Hermant[2], and Romain Guillaumé[1,2]

[1] Université Paris-Saclay, ENS Paris-Saclay, LMF, Inria, France
[2] CRI, Mines Paris, PSL University, Paris, France

Abstract. Atelier B is widely used to develop safe-by-construction programs. Numerous systems and pieces of software have been designed and built at the highest level of safety with this framework. But similar toolsets are used for the same purpose, and we advocate that they would all benefit from proof-exchange facilities. To this aim, we introduce an export tool of Atelier B proof obligations to Dedukti, a universal logical framework based on type theory, and designed for interoperability. We then describe a preliminary experimentation to reconstruct detailed demonstrations of those proof obligations, so that they can be double-checked by Dedukti, and be ready for a later export to the other tools. We also discuss the methodologies and potential framework architectures that could be helpful in such a proof-exchange objective.

Keywords: B Method · Type theory · Automated reasoning · Proof theory · Formal methods · Set theory · Interoperability

1 Introduction

Atelier B [20] is widely used to develop safe-by-construction programs, in particular in the rail industry [39]. It is currently used by manufacturers for the certification of microelectronic components at the EAL6+ level according to Common Criteria (ISO/IEC 15408), under the supervision of the French National Cybersecurity Agency (ANSSI). During the formalization and development processes, it generates numerous *proof obligations* to ensure that the program respects the given specifications.

However, the formal structure of the proofs of Atelier B is not explicit and difficult to piece together. The available traces are scripts that can call integrated automated theorem provers; use thousands of small inference rules that rely on checking guard conditions which are declared in separate files; or depend on implicit built-in simplification rules, like variable-free numeric computations, that are directely embedded in the kernel. Beyond these calls to these large steps, we have not been able to exhibit a more detailed proof structure or an explicit notion of proof tree.

As other toolsets exist in this domain, some of which are also set-based, like Rodin [3] or TLAPS [17], there would be huge benefits in double-checking

proof or sharing libraries across systems. This is one of the goals of the ICSPA French-agency-funded research project, which started in 2022.

Dedukti [4] is a logical framework designed for interoperability [10]. It has already been used to share proofs across different systems [46]. Our aim is to express, in Dedukti, the logic of Atelier B, and to translate the proof obligation statements, in order to reconstruct proofs in Dedukti, allowing us to share the results with other tools later.

After a short survey of the related work and a nutshell presentation of the prerequisites about Atelier B, we present both the foundations and the pratical results of our export tool of Atelier B proof obligations to Dedukti. We then describe an experiment to construct – using automated theorem provers – Dedukti proofs of those statements, which will be the starting point for export and sharing. Finally, we discuss the possible tools, methodologies and architectures for the last two steps of proof reconstruction and sharing, and conclude the paper.

2 Related work

Bolstering the proving strength of Atelier B with external provers is a longstanding effort. In 2012–2016, the BWare project [44] has tackled the question with numerous tools, some of which we will use in this work and describe below. More recently, SMT solvers, like VeriT [12], are being investigated to this aim, and a large benchmark consisting of more than $700,000$ anonymized proof obligations in two different formats has been made publicly available [19].

At the core of the BWare framework, the B Method [1] axioms and basic constructs have been expressed in Why3 [27], a platform based on an ML-polymorphic type theory. A proprietary tool of one of the BWare industrial participants has then been used to convert Atelier B proof obligations into Why3 specifications. This platform is then used as a hub to discharge the proofs to automated theorem provers through specific drivers towards, in particular, Alt-Ergo [21], and also iProver Modulo and Zenon Modulo [15,16].

The last two provers are based on Deduction modulo theory [24], that combines first-order logic with rewriting. This last feature has proved very useful to express the computational content of axioms (e.g. arithmetic) and gives rise to speedups both in time and size [13], begin able to express features that go beyond the first order [14]. A very similar rewriting feature is at the heart of Dedukti too. Moreover, Zenon Modulo has been designed to produce proof terms, that can be directly fed to Dedukti, just like its direct ancestor Zenon [11] is capable of producing proofs for TLAPS [35], FoCaLiZe [25], or Coq [45].

TPTP [43] is a common format to formally express problems under various forms. Zenon Modulo uses as input the TFF1 [7] form, which is quite close to the Why3 polymorphic logic, while iProver Modulo requires formulas to be monomophised first and thus expressed in the FOF form.

Dedukti [4] is a language based on $\lambda\Pi$-calculus modulo theory [22], designed for interoperability [10]. The base type system, also known as LF [30], is enriched with a rewriting feature, as mentioned above. This yields a highly configurable

logic [8], in which it is possible to express the calculus of constructions, Agda [28], or PVS [31], for instance. The two main dialects of Dedukti are a type-checker [41] called `dk` and a proof assistant, called Lambdapi. It boasts a wide range of helpful tools [23] that support the activity of sharing proofs across systems : import, reverse mathematics analysis, termination checking [9], export [46], or automated proof-term filling [26] (like SledgeHammer [38]) tools.

The proof community has made interesting efforts towards the definition of standards for proof exchange or theory manipulations [37]. ProofCert is a project which [36] puts the focus on the structure of proofs, that one can divide in a succession of (clerk) mechanical phases, separated by (expert) choice points, arguing that only the choice points need to be recorded. It is of particular interest for first-order reasoning [18]. As of type-theoretic frameworks, the MMT [33] and Mathematical Knowledge Management [6] efforts give tools to manipulate the theory and the theorems of a wide range of frameworks, while OpenTheory [32] is a common proof format for all the HOL-based proof assistants.

Within the Dedukti logical framework, a recent work has introduced a modular way to define the most common logics, where features, like classical or intuitionistic connectives, higher-order types, impredicativity, and so on, can be picked up and composed in a modular way: the Theory U [8]. As we want to go beyond Atelier B, or even the B Method itself, and also because some of the steps of BWare were proprietary, we do not directly use the Why3-TPTP intermediate steps. We rather follow the Theory U to encode the logic and constructions of Atelier B, and a direct translation of proof obligations in this theory. By using a similar approach for the other target set-based tools, we will be able to represent different set-theoretic tools not only within the same language and framework, but by using a common set of symbols and axioms as far as it is possible.

3 The B method

The B method has been defined by J.-R. Abrial [1] as a framework for safe-by-construction software design and production. At its heart is a typed set theory, that consists in first-order logic with equality reasoning rules, basic set constructs, an axiomatization of the base axioms of set theory, like the cartesian product, the powerset, or the comprehension scheme; and derived constructs, like intersection, union, functions, etc.

The B method also defines a methodology to specify and prove correct a program, that is implemented by Atelier B. A *machine* consists of code along with logical formulas (invariant) which express the desired properties of the code. Atelier B then automatically derives, by computing a weakest pre-condition, the *proof obligations* that have to be shown to hold, within the first-order logic system describe above, in order for the invariants to be actual properties of the code.

Machines can be abstract and, if the user wants to generate executable code, he usually must go through several *refinement* steps, starting with an abstract machine and giving more and more precise and concrete versions of it at each

step. During the refinement process, Atelier B generates proof obligations, that must be proved to ensure that the refined machine still satisfies the invariants of the previous-step machine.

Proof obligations come as PO files, and can be exported to an XML format, also known as POG files. The user has to prove them through a dedicated interface of Atelier B, by combining the automated, often configurable, tactics and provers offered by Atelier B. Among the later tools, one can find `pp` and `mp`, that can be called at different levels of heuristics and deepness in proof search (the "forces").

Event B [2] is a simplification as well as an extension of the B method which has a more flexible refinement concept targeted at systems modelling. It allows for instance a better treatment of loops, that can be introduced at a higher abstraction level. Event B is implemented in the Rodin tool [3], and it is also supported by Atelier B, although we are not using this feature in our work. Differences between B and Event-B are listed in [34].

In the following example, there are three machines: the first one is the specification, the second one is a refinement, the third one is the implementation.

```
MACHINE
    example

VARIABLES
    set

INVARIANT
    set : FIN(NAT1)

INITIALISATION
    set := {}

OPERATIONS
    enter(nn) =
    PRE nn : NAT1
    THEN
        set :=
          set \/ {nn}
    END;

    mm <-- maximum =
    PRE set /= {}
    THEN
        mm := max(set)
    END
END
```

```
REFINEMENT example_r
REFINES example

VARIABLES
    zz

INVARIANT
    zz : NAT &
    zz =
      max(set \/ {0})

INITIALISATION
    zz := 0

OPERATIONS
    enter(nn) =
    PRE nn : NAT1
    THEN
        zz :=
          max({zz,nn})
    END;

    mm <-- maximum =
    PRE zz /= 0
    THEN
        mm := zz
    END
 END
```

```
IMPLEMENTATION example_i
REFINES example_r

CONCRETE_VARIABLES
    zz

INVARIANT
    zz : NAT

INITIALISATION
    zz := 0

OPERATIONS
    enter(nn) =
    BEGIN
        IF nn >= zz
        THEN zz := nn
        END
    END;

    mm <-- maximum =
    BEGIN
        mm := zz
    END
END
```

4 Translating B Proof Obligations

In order to translate B predicates into Dedukti, we need to formalize its type system, which is quite unusual compared to common type theories [5], because there is no arrow type. Indeed, it is not inspired by the λ-calculus, but by set theory, so the basic type constructs are the cartesian product $\_ \times \_$, the powerset type $\mathrm{Set}(\_)$, and, for practical reasons, the record type constructor $\mathrm{struct}(\ldots)$. A function from A to B has the typical set-theoretic type $\mathrm{Set}(A \times B)$, the type of relations between A and B, with extra properties ensuring functionality of the relation. As such, most of the reasoning on terms is done through an axiomatic semantics.

4.1 Formalizing the B Type System in Dedukti

Basic types of Atelier B, noted b in the syntax, are $\mathbb{Z}$ for integers, $\mathbb{R}$ and $\mathbb{FLOAT}$ for floats, $\mathbb{BOOL}$ for booleans, and $\mathbb{STRING}$ for strings, even though we may consider extra basic types.

We use four categories of terms:

- types T, which type objects
- objects M, which are programs and data (integers, functions, records...)
- kinds K, which types predicates
- predicates, which are either 0-ary (of type Prop), or unary (of type $T \to \mathrm{Prop}$), for some type T

The set of basic constructor with their signature type is called Σ. Basic first-order constructors, noted $c(M_1, \ldots, M_n)$ in the syntax, are too numerous to be listed comprehensively. An n-ary constructor c has a signature type $(T_1, \ldots, T_n) \to T$ (we can write $c : T$ if $n = 0$). All integer and float constants are considered as basic constructors, as well as arithmetic operators.

Many constructors in B are actually expressed in an ML-polymorphic style [29], for instance operations on sets, relations, functions, sequences, or trees. Unlike constructors, functions or sets themselves are always monomorphic. To keep the presentation simple, each constructor is introduced as a family of monomorphic constructors, although it is actually represented in Dedukti as a polymorphically encoded single symbol. For instance, the family of cardinal operators Card_T has type $\mathrm{Set}(T) \to \mathbb{Z}$, while in the actual Dedukti encoding, the operator Card has type $(T : \texttt{type}) \to \tau(\mathrm{Set}(T)) \to \tau\mathbb{Z}$.

There is only a handful of second-order constructors, used for comprehension and for defining functions as well as generalized versions of sums, products, unions, and intersections. Comprehension depends on a typed predicate $P : T \to \mathrm{Prop}$ and defines the set $\{P\}$ of all objects that satisfy P. All other second-order constructors have a signature type of the shape $(T_1 \to T_2) \to T_3$ and we factor their presentation and, in the next section, their typing rules. We collectively refer to these second-order constructors as $h(\lambda x : T_1.(P|M))$, where P is a predicate that restricts the domain T_1. For instance, the function constructor $\%_{T_1,T_2}$ has type $(T_1 \to T_2) \to \mathrm{Set}(T_1 \times T_2)$, and the generalized sum on integers Σ_T has type $(T \to \mathbb{Z}) \to \mathbb{Z}$.

Two variadic constructors exist: a constructor for sets by extension $\{M_1, \ldots, M_n\}$ and a constructor for sequences $[M_1, \ldots, M_n]$.

The operator $\mathrm{bool}(\_)$ converts predicates to booleans. The binary predicate operators, noted op, are the usual connectives $\vee$, $\wedge$, $\Rightarrow$, and $\Leftrightarrow$.

The syntax for the four categories is as follows:

$$
\begin{aligned}
T &::= b \mid T \times T \mid \mathrm{Set}(T) \mid \mathrm{struct}(l_1 : T_1, \ldots, l_n : T_n) \\
M &::= x \mid c(M_1, \ldots, M_n) \mid h(\lambda x : T.(P|M)) \mid \mathrm{bool}(P) \mid \{P\} \mid \{M_1, \ldots, M_n\} \\
&\quad \mid \ [M_1, \ldots, M_n] \mid (M_1, M_2) \mid \mathrm{pr}_i(M) \mid \mathrm{rec}(l_1 : M_1, \ldots, l_n : M_n) \mid l.M \\
K &::= \mathrm{Prop} \mid T \to \mathrm{Prop} \\
P &::= M_1 \in M_2 \mid M_1 = M_2 \mid P\ M \mid \forall x : T, P \mid \exists x : T, P \mid P\ op\ P \mid \lambda x : T, P \\
&\quad \mid \ \neg P \mid \mathrm{true}
\end{aligned}
$$

4.2 Typing rules

There are two kinds of judgments: $\Gamma \vdash M : T$ for typing objects, and $\Gamma \vdash P : K$ for typing predicates. The environment Γ is a finite set containing declarations of the form $x : T$. Rules for typing objects are found in Fig. 1, while rules for typing predicates are found in Fig. 2.

$$
\frac{(x : T) \in \Gamma}{\Gamma \vdash x : T}
\qquad
\frac{(c : (T_1, \ldots, T_n) \to T) \in \Sigma \quad \Gamma \vdash M_1 : T_1 \quad \cdots \quad \Gamma \vdash M_n : T_n}{\Gamma \vdash c(M_1, \ldots, M_n) : T}
$$

$$
\frac{\Gamma \vdash P : \mathrm{Prop}}{\Gamma \vdash \mathrm{bool}(P) : \mathbb{BOOL}}
\qquad
\frac{(h : (T_1 \to T_2) \to T_3) \in \Sigma \quad \Gamma, x : T_1 \vdash P : \mathrm{Prop} \quad \Gamma, x : T_1 \vdash M : T_2}{\Gamma \vdash h(\lambda x : T_1.(P|M)) : T_3}
$$

$$
\frac{\Gamma \vdash P : T \to \mathrm{Prop}}{\Gamma \vdash \{P\} : \mathrm{Set}(T)}
\qquad
\frac{\Gamma \vdash M_1 : T \quad \cdots \quad \Gamma \vdash M_n : T}{\Gamma \vdash \{M_1, \ldots, M_n\} : \mathrm{Set}(T)}
$$

$$
\frac{\Gamma \vdash M : T_1 \times T_2}{\Gamma \vdash \mathrm{pr}_i(M) : T_i}
\qquad
\frac{\Gamma \vdash M_1 : T \quad \cdots \quad \Gamma \vdash M_n : T}{\Gamma \vdash [M_1, \ldots, M_n] : \mathrm{Set}(\mathbb{Z} \times T)}
$$

$$
\frac{\Gamma \vdash M_1 : T_1 \quad \Gamma \vdash M_2 : T_2}{\Gamma \vdash (M_1, M_2) : T_1 \times T_2}
\qquad
\frac{\Gamma \vdash M : \mathrm{struct}(l_1 : T_1, \ldots, l_n : T_n)}{\Gamma \vdash l_i.M : T_i}
$$

$$
\frac{\Gamma \vdash M_1 : T_1 \quad \cdots \quad \Gamma \vdash M_n : T_n}{\Gamma \vdash \mathrm{rec}(l_1 : M_1, \ldots, l_n : M_n) : \mathrm{struct}(l_1 : T_1, \ldots, l_n : T_n)}
$$

Fig. 1. Rules for typing objects

4.3 Second-Order Encoding

We have written a translator from the POG format to Lambdapi using the constructors above. This encoding requires a use of the classical many-sorted second-order logic because of the second-order constructors.

We have benchmarked our encoding on a dataset [19] of more than 700,000 proof obligations given in 5434 POG files (6.3 GiB). We manage to translate all, but 47 files, into Lambdapi. The remaining failures stem from an ill-typing of the POG files themselves, that trigger a typing error in Lambdapi during the translation. These errors are a side effect of the anonymization of the original Atelier B PO files, and the feedback of our experiment has led to improvements of this latter step.

$$\frac{\Gamma \vdash M_1 : T \quad \Gamma \vdash M_2 : \mathrm{Set}(T)}{\Gamma \vdash M_1 \in M_2 : \mathrm{Prop}} \qquad \frac{\Gamma, x : T \vdash P : \mathrm{Prop}}{\Gamma \vdash \forall x : T, P : \mathrm{Prop}}$$

$$\frac{\Gamma \vdash M_1 : T \quad \Gamma \vdash M_2 : T}{\Gamma \vdash M_1 = M_2 : \mathrm{Prop}} \qquad \frac{\Gamma, x : T \vdash P : \mathrm{Prop}}{\Gamma \vdash \exists x : T, P : \mathrm{Prop}}$$

$$\frac{\Gamma \vdash P_1 : \mathrm{Prop} \quad \Gamma \vdash P_2 : \mathrm{Prop}}{\Gamma \vdash P_1 \mathit{op} P_2 : \mathrm{Prop}} \qquad \frac{\Gamma \vdash P : \mathrm{Prop}}{\Gamma \vdash \neg P : \mathrm{Prop}}$$

$$\frac{\Gamma, x : T \vdash P : \mathrm{Prop}}{\Gamma \vdash \lambda x : T.P : T \to \mathrm{Prop}} \qquad \frac{}{\Gamma \vdash \mathrm{true} : \mathrm{Prop}}$$

$$\frac{\Gamma \vdash P : T \to \mathrm{Prop} \quad \Gamma \vdash M : T}{\Gamma \vdash P\, M : \mathrm{Prop}}$$

Fig. 2. Rules for typing predicates

4.4 First-Order Encoding

In order for our system to interoperate with solvers, we are working on an encoding in classical many-sorted first-order logic, which could be done both in Dedukti and in Why3. The main difference with the second-order encoding is the treatment of second-order constructors and set constructors, which requires the use of *ad hoc* axioms for each use of these constructors. Sets defined by some second-order operation will be defined as an atomic set, and an axiom will define its content. For instance, the sequence [TRUE, FALSE, TRUE] will be defined as a fresh atomic set s, with the accompanying axiom:

$$\forall x : \mathbb{Z} \times \mathbb{BOOL}.x \in s \Leftrightarrow x = (1, \mathrm{TRUE}) \vee x = (2, \mathrm{FALSE}) \vee x = (3, \mathrm{TRUE})$$

instead of the second-order definition by comprehension $s = \{\lambda x : \mathbb{Z} \times \mathbb{BOOL}.x = (1, \mathrm{TRUE}) \vee x = (2, \mathrm{FALSE}) \vee x = (3, \mathrm{TRUE})\}$.

Similarly, the function $\%_{T_1,T_2}(\lambda x : T_1.(P(x)|f(x))$ is encoded by creating a fresh atomic set s, and, assuming $P(x)$ and $f(x)$ have been encoded properly, we add the accompanying axiom:

$$\forall z : T_1 \times T_2.z \in s \Leftrightarrow \exists x : T_1.P(x) \wedge z = (x, f(x))$$

Generalized sums and generalized products are a little bit trickier. To illustrate this difficulty, let us consider the sum of integers $\Sigma_T(\lambda x : T.(P(x)|f(x))$. We can consider a fresh atomic function σ taking a set of type $\mathrm{Set}(T)$ as an argument. $\sigma(A)$ is defined as the sum of $f(A)$, with the axioms:

$$\sigma(\varnothing) = 0$$
$$\forall (x : T)(A : \mathrm{Set}(T)).x \notin A \Rightarrow \sigma(\{x\} \cup A) = f(x) + \sigma(A)$$

5 Reconstruction of Proofs

5.1 Pipeline

Following the BWare project, we have designed a pipeline for generating proof of Atelier B proofs obligations in Dedukti. This pipeline (Fig. 3) first consists in translating the

proof obligations of Atelier B from the PO or POG formats to the WhyML format of the Why3 platform. This relies on an implementation of the B theory in the WhyML language. We then make use of the different Why3 drivers available to convert the proof obligations from WhyML to the automated theorem provers native format. In our case, we used the drivers for iProver and Zenon Modulo to respectively produce problems in TPTP-FOF and TPTP-TFF1 formats. Both provers then attempt to solve the problem and output a Dedukti proof certificate in case of success, which validity is verified by the Dedukti proof checker. As a proof obligation is transformed multiple times and because each TPTP file must contain the theory defining the objects of the statements to prove, the file sizes increase considerably throughout the pipeline.

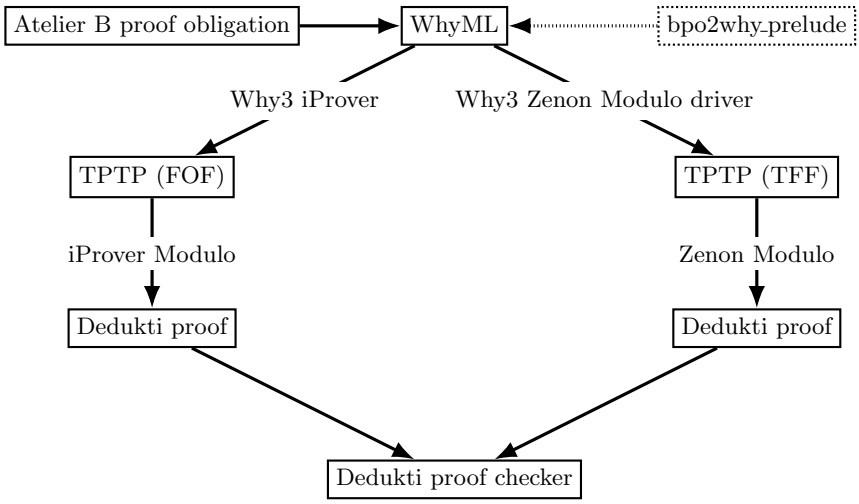

Fig. 3. Pipeline for proof reconstruction

5.2 Example

We illustrate the different stages and transformations a proof obligation goes through while it is traversing the pipeline. In this example, the proof obligation consists in proving one implication. One can track the predicates f_1, f_2, f_3, f_4, f_5, and f_6, and the variables `_locks`, `_card`, `_K0`, `_HS`, `_Dmin`, `_Dmax`, `_DATE`, and `_CARD`, on which they depend, as visual anchors to understand the structure. Note that the predicate f_6 is originally defined as the boolean `bfalse` and f_5 as `btrue`.

The PO file (Fig. 4) gives the goal to prove in the theory `ProofList`, which states that the predicates f_1 through f_5 should imply f_6. These predicates are listed in the theory `Formulas`.

The WhyML translation (Fig. 5) defines all the predicates (omitted in Fig. 5), which takes all the variables `_locks`, `_card`... as arguments, and restates the goal in WhyML.

```
   THEORY ProofList IS
     _f(1) & _f(2) & _f(3) &
     b_close_disk.1, (_f(4) & _f(5) => _f(6))
   END
   &
   THEORY Formulas IS
f1   K0: CARD & HS: CARD & not(K0 = HS) & CARD: FIN(INTEGER) &
     not(CARD={}) & Dmin: INTEGER & 0<=Dmin & Dmin<=2147483647 &
     Dmax: INTEGER & 0<=Dmax & Dmax<=2147483647 & Dmin+1<=Dmax &
     DATE = Dmin..Dmax;
f2   locks: BOOL & card: BOOL & (card = FALSE => locks = FALSE);
f3   card = TRUE;
f4   card = FALSE;
f5   btrue
f6   bfalse
   END
```

Fig. 4. PO file

```
goal b_close_disk_1 :
    forall _locks: bool, _card: bool, _K0: int, _HS: int,
           _Dmin: int, _Dmax: int, _DATE: set int, _CARD: set int.
       ((f1 _locks _card _K0 _HS _Dmin _Dmax _DATE _CARD)
     /\ (f2 _locks _card _K0 _HS _Dmin _Dmax _DATE _CARD)
     /\ (f3 _locks _card _K0 _HS _Dmin _Dmax _DATE _CARD)
     /\ (f4 _locks _card _K0 _HS _Dmin _Dmax _DATE _CARD)
     /\ (f5 _locks _card _K0 _HS _Dmin _Dmax _DATE _CARD))
     -> (f6 _locks _card _K0 _HS _Dmin _Dmax _DATE _CARD)
```

Fig. 5. WhyML file

Why3 can generate a translation of the goal in the TFF format (Fig. 6) for Zenon Modulo, or in the FOF format (Fig. 7) for iProver. The goal translated by Zenon Modulo in Dedukti is given in Fig. 8.

```
tff(b_close_disk_1, conjecture,
  ![_locks:bool, _card:bool, _K0: $int, _HS:$int, _Dmin:$int,
    _Dmax:$int, _DATE:infix_mngt($int, bool),
    _CARD:infix_mngt($int, bool)]:
      ((f1(_locks, _card, _K0, _HS, _Dmin, _Dmax, _DATE, _CARD) &
       (f2(_locks, _card, _K0, _HS, _Dmin, _Dmax, _DATE, _CARD) &
       (f3(_locks, _card, _K0, _HS, _Dmin, _Dmax, _DATE, _CARD) &
       (f4(_locks, _card, _K0, _HS, _Dmin, _Dmax, _DATE, _CARD) &))))
      => ($false)).
```

Fig. 6. TPTP-TFF1 file

```
fof(b_close_disk_1, conjecture,
  ![_locks, _card, _K0, _HS, _Dmin, _Dmax, _DATE, _CARD]:
    ((sort(bool, _locks) = _locks) =>
    ((sort(bool, _card) = _card) =>
    ((sort(int, _K0) = _K0) =>
    ((sort(int, _HS) = _HS) =>
    ((sort(int, _Dmin) = _Dmin) =>
    ((sort(int, _Dmax) = _Dmax) =>
    ((sort(infix_mngt(int, bool), _DATE) = _DATE) =>
    ((sort(infix_mngt(int, bool), _CARD) = _CARD) =>
      ~ (f1(_locks, _card, _K0, _HS, _Dmin, _Dmax, _DATE, _CARD) & (
      f2(_locks, _card, _K0, _HS, _Dmin, _Dmax, _DATE, _CARD) & (
      f3(_locks, _card, _K0, _HS, _Dmin, _Dmax, _DATE, _CARD) & (
      f4(_locks, _card, _K0, _HS, _Dmin, _Dmax, _DATE, _CARD))
))))))))))).
```

Fig. 7. TPTP-FOF file

5.3 Benchmark

We have used a test suite of 23 proof obligations coming from Atelier B. These proof obligations were provided already translated into the WhyML format by the Mitsubishi Electric. Indeed, their `bpo2why` translator is proprietary software. The test suite represents typical proof obligations of a tutorial project utilizing Atelier B. More precisely, the proof obligations come from the specification of an Automated Teller Machine software. We compare 4 provers : Zenon Modulo with and without Deduction Modulo

```
def zenon_G : proof (not (
  forall bool (a : (El bool) =>
   forall bool (b : (El bool) =>
    forall Z (c : (El Z) =>
     forall Z (d : (El Z) =>
      forall Z (e : (El Z) =>
       forall Z (f : (El Z) =>
        forall (infix_mngt Z bool) (g : (El (infix_mngt Z bool)) =>
         forall (infix_mngt Z bool) (h : (El (infix_mngt Z bool)) =>
          imp (
           and (f1 a b c d e f g h)(
           and (f2 a b c d e f g h)(
           and (f3 a b c d e f g h)(
           and (f4 a b c d e f g h)
          ))))
  False))))))))))) -> seq.
```

Fig. 8. Dedukti file

Theory, iProver Modulo (with OCaml 4.02.1) and Alt-Ergo for reference. Alt-Ergo is not yet capable of outputting proof certificates in Dedukti and is called directly from the Why3 platform.

The commands used for converting problems from WhyML to the TPTP format are as follows:

- for Zenon Modulo

  `why3 prove -D zenon_modulo.drv -L <prelude_dir> <file.why>`

- for iProver Modulo

  `why3 prove -D iProver.drv -L <prelude_dir> <file.why>`

- for Zenon Modulo without Deduction Modulo Theory (`-odk` flag is for outputting Dedukti)

  `zenon_modulo -itptp -odk -max-time 5 -max-size 3G <file.tptp>`

- for Zenon Modulo with Deduction Modulo Theory

  ```
  zenon_modulo -itptp -modulo-heuri -max-time 5
               -max-size 3G <file.tptp>
  ```

- for iProver Modulo

  `sh /path/to/iprover_modulo_launcher.sh <file.tptp> 120`

- for Alt-Ergo

  `why3 prove -P alt-ergo -L <prelude_dir> <file.why>`

  We get the following results:

	Numbers of proof obligations proved
Zenon Modulo (`-odk`)	1/23
Zenon Modulo (`-modulo-heuri`)	11/23
iProver Modulo	1/23
Alt-Ergo	16/23

6 Conclusion

In this paper, we have introduced a direct translation schema from Atelier B to Dedukti, that makes explicit the notion of types, and expresses the basic constructs. This translation is implemented in a tool, that is effectively able to translate most of the POG files of the apero suite. Moreover, the benchmarking of the tool has allowed to discover and correct several errors in Atelier B's POG file generator.

After that, we have described our investigations in rebuilding, in Dedukti, the proofs of the translated proof obligations. Reviving the BWare pipeline, through an intermediate Why3 translation that is no more freely available, and a call to automated theorem provers that can produce proof terms, has proved more difficult than expected, but nevertheless allowed us to validate the approach.

We have yet to align the Zenon Modulo produced proofs in Dedukti with the translation of the proof obligations that we have. More importantly, we have to investigate other directions for this proof reconstruction step, in particular the ability of ekstrakto [26] to take Dedukti incomplete proof terms as an input, and to call Zenon Modulo on it. Likewise, we also plan to use GDV [42], which is a tool allowing us to use a prover that outputs TSTP proof traces (e.g. Vampire [40]) and uses Zenon Modulo to transform this trace into a Dedukti proof term. Another direction is to take advantage of the proof scripts of Atelier B, which we are not exploiting for the moment.

As for proof sharing, it is first and foremost important to make sure to be able to translate the reconstructed proofs of Atelier B proof obligations to Atelier B itself. This first essential step already bears its own complexity and already provides an interesting use case: exporting the proof obligation to Dedukti, building a proof in this framework through a call to automated theorem provers, and importing the proofs back. Independently of this work about homomorphic translations, we also have to align the two dialects of the B method, Atelier B and Rodin, in Dedukti, in order to be able to effectively share proof between systems. For those two objectives, an infrastructure working even on a small fraction of the benchmark, e.g. with a restricted set of constructs or when the proof scripts are simple, would already be a success.

References

1. Abrial, J.R.: The B-book: Assigning Programs to Meanings. Cambridge University Press, New York, NY, USA (1996)
2. Abrial, J.R.: Modeling in Event-B - System and Software Engineering. Cambridge University Press (2010)
3. Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: RODIN: an open toolset for modelling and reasoning in Event-B. International journal on software tools for technology transfer **12**(6), 447–466 (2010)
4. Assaf, A., Burel, G., Cauderlier, R., Delahaye, D., Dowek, G., Dubois, C., Gilbert, F., Halmagrand, P., Hermant, O., Saillard, R.: Expressing theories in the $\lambda\Pi$-calculus modulo theory and in the Dedukti system. In: TYPES. Novi Sad, Serbia (2016)
5. Barendregt, H., Dekkers, W., Statman, R.: Lambda calculus with types. Cambridge University Press (2013)
6. Bercic, K., Kohlhase, M., Rabe, F.: Towards a heterogeneous query language for mathematical knowledge. In: Benzmüller, C., Miller, B.R. (eds.) Intelligent Computer Mathematics - 13th International Conference, CICM 2020, Bertinoro, Italy,

July 26-31, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12236, pp. 39–54. Springer (2020)
7. Blanchette, J.C., Paskevich, A.: TFF1: the TPTP typed first-order form with rank-1 polymorphism. In: Bonacina, M.P. (ed.) Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7898, pp. 414–420. Springer (2013)
8. Blanqui, F., Dowek, G., Grienenberger, É., Hondet, G., Thiré, F.: Some axioms for mathematics. In: Kobayashi, N. (ed.) 6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference). LIPIcs, vol. 195, pp. 20:1–20:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021)
9. Blanqui, F., Genestier, G., Hermant, O.: Dependency pairs termination in dependent type theory modulo rewriting. In: Geuvers, H. (ed.) 4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany. LIPIcs, vol. 131, pp. 9:1–9:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
10. Boespflug, M., Carbonneaux, Q., Hermant, O.: The $\lambda\Pi$-calculus modulo as a universal proof language. In: Pichardie, D., Weber, T. (eds.) Proceedings of the Second International Workshop on Proof Exchange for Theorem Proving, PxTP 2012, Manchester, UK, June 30, 2012. CEUR Workshop Proceedings, vol. 878, pp. 28–43. CEUR-WS.org (2012)
11. Bonichon, R., Delahaye, D., Doligez, D.: Zenon : An extensible automated theorem prover producing checkable proofs. In: Dershowitz, N., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning, 14th International Conference, LPAR 2007, Yerevan, Armenia, October 15-19, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4790, pp. 151–165. Springer (2007)
12. Bouton, T., Caminha de Oliveira, D., Déharbe, D., Fontaine, P.: veriT: An open, trustable and efficient SMT-solver. In: Schmidt, R. (ed.) 22nd Intl. Conf. Automated Deduction. LNCS, vol. 5663, pp. 151–156. Springer, Montreal, Canada (2009)
13. Burel, G.: Unbounded proof-length speed-up in deduction modulo. In: Duparc, J., Henzinger, T.A. (eds.) Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4646, pp. 496–511. Springer (2007)
14. Burel, G.: Efficiently simulating higher-order arithmetic by a first-order theory modulo. Log. Methods Comput. Sci. **7**(1) (2011)
15. Burel, G., Bury, G., Cauderlier, R., Delahaye, D., Halmagrand, P., Hermant, O.: First-order automated reasoning with theories: When deduction modulo theory meets practice. J. Autom. Reason. **64**(6), 1001–1050 (2020)
16. Bury, G., Delahaye, D., Doligez, D., Halmagrand, P., Hermant, O.: Automated deduction in the B set theory using typed proof search and deduction modulo. In: Fehnker, A., McIver, A., Sutcliffe, G., Voronkov, A. (eds.) 20th International Conferences on Logic for Programming, Artificial Intelligence and Reasoning - Short Presentations, LPAR 2015, Suva, Fiji, November 24-28, 2015. EPiC Series in Computing, vol. 35, pp. 42–58. EasyChair (2015)
17. Chaudhuri, K., Doligez, D., Lamport, L., Merz, S.: Verifying safety properties with the TLA+ proof system. In: Giesl, J., Hähnle, R. (eds.) Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010.

Proceedings. Lecture Notes in Computer Science, vol. 6173, pp. 142–148. Springer (2010)
18. Chihani, Z., Miller, D., Renaud, F.: A semantic framework for proof evidence. J. Autom. Reason. **59**(3), 287–330 (2017)
19. ClearSy: (2022), `https://github.com/CLEARSY/apero/tree/main/DATA`
20. ClearSy System Engineering: Atelier B. `https://www.atelierb.eu/en/`
21. Conchon, S., Iguernelala, M.: Tuning the Alt-Ergo SMT solver for B proof obligations. In: Ameur, Y.A., Schewe, K. (eds.) Abstract State Machines, Alloy, B, TLA, VDM, and Z - 4th International Conference, ABZ 2014, Toulouse, France, June 2-6, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8477, pp. 294–297. Springer (2014)
22. Cousineau, D., Dowek, G.: Embedding pure type systems in the lambda-pi-calculus modulo. In: Rocca, S.R.D. (ed.) Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007. Lecture Notes in Computer Science, vol. 4583, pp. 102–117. Springer (2007)
23. Deducteam: Dedukti, a logical framework, `https://deducteam.github.io/` and `https://github.com/Deducteam`
24. Dowek, G., Hardin, T., Kirchner, C.: Theorem proving modulo. Journal of Automated Reasoning **31**, 33–72 (2003)
25. Dubois, C., Pessaux, F.: Termination proofs for recursive functions in FoCaLiZe. In: Serrano, M., Hage, J. (eds.) Trends in Functional Programming - 16th International Symposium, TFP 2015, Sophia Antipolis, France, June 3-5, 2015. Revised Selected Papers. Lecture Notes in Computer Science, vol. 9547, pp. 136–156. Springer (2015)
26. El Haddad, M.Y., Burel, G., Blanqui, F.: EKSTRAKTO A tool to reconstruct dedukti proofs from TSTP files (extended abstract). In: Reis, G., Barbosa, H. (eds.) Proceedings Sixth Workshop on Proof eXchange for Theorem Proving, PxTP 2019, Natal, Brazil, August 26, 2019. EPTCS, vol. 301, pp. 27–35 (2019)
27. Filliâtre, J., Paskevich, A.: Why3 - where programs meet provers. In: Felleisen, M., Gardner, P. (eds.) Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7792, pp. 125–128. Springer (2013)
28. Genestier, G.: Encoding Agda programs using rewriting. In: Ariola, Z.M. (ed.) 5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference). LIPIcs, vol. 167, pp. 31:1–31:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
29. Halmagrand, P.: Automated deduction and proof certification for the B method. Theses, Conservatoire national des arts et metiers - CNAM (2016)
30. Harper, R., Honsell, F., Plotkin, G.D.: A framework for defining logics. J. ACM **40**(1), 143–184 (1993)
31. Hondet, G., Blanqui, F.: Encoding of predicate subtyping with proof irrelevance in the $\lambda\Pi$-calculus modulo theory. In: de'Liguoro, U., Berardi, S., Altenkirch, T. (eds.) 26th International Conference on Types for Proofs and Programs, TYPES 2020, March 2-5, 2020, University of Turin, Italy. LIPIcs, vol. 188, pp. 6:1–6:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
32. Hurd, J.: The OpenTheory standard theory library. In: Bobaru, M., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) NASA Formal Methods. pp. 177–191. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
33. Kohlhase, M., Rabe, F.: Experiences from exporting major proof assistant libraries. J. Autom. Reason. **65**(8), 1265–1298 (2021)

34. Leuschel, M.: Spot the difference: a detailed comparison between B and Event-B. Logic, Computation and Rigorous Methods: Essays Dedicated to Egon Börger on the Occasion of His 75th Birthday pp. 147–172 (2021)
35. Merz, S.: Proofs and proof certification in the TLA$^+$ proof system. In: Pichardie, D., Weber, T. (eds.) Proceedings of the Second International Workshop on Proof Exchange for Theorem Proving, PxTP 2012, Manchester, UK, June 30, 2012. CEUR Workshop Proceedings, vol. 878, pp. 16–20. CEUR-WS.org (2012)
36. Miller, D.: ProofCert: Broad Spectrum Proof Certificates (Feb 2011), an ERC Advanced Grant funded for the five years 2012–2016, `http://www.lix.polytechnique.fr/Labo/Dale.Miller/ProofCert/`
37. Miller, D.: A distributed and trusted web of formal proofs. In: Hung, D.V., D'Souza, M. (eds.) Distributed Computing and Internet Technology - 16th International Conference, ICDCIT 2020, Bhubaneswar, India, January 9-12, 2020, Proceedings. Lecture Notes in Computer Science, vol. 11969, pp. 21–40. Springer (2020)
38. Paulson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: Sutcliffe, G., Schulz, S., Ternovska, E. (eds.) The 8th International Workshop on the Implementation of Logics, IWIL 2010, Yogyakarta, Indonesia, October 9, 2011. EPiC Series in Computing, vol. 2, pp. 1–11. EasyChair (2010)
39. Peleska, J., Haxthausen, A.E., Lecomte, T.: Standardisation considerations for autonomous train control. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Practice - 11th International Symposium, ISoLA 2022, Rhodes, Greece, October 22-30, 2022, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 13704, pp. 286–307. Springer (2022)
40. Riazanov, A., Voronkov, A.: The design and implementation of vampire. AI communications **15**(2-3), 91–110 (2002)
41. Saillard, R.: Rewriting modulo β in the $\lambda\Pi$-calculus modulo. In: Cervesato, I., Chaudhuri, K. (eds.) Proceedings Tenth International Workshop on Logical Frameworks and Meta Languages: Theory and Practice, LFMTP 2015, Berlin, Germany, 1 August 2015. EPTCS, vol. 185, pp. 87–101 (2015)
42. Sutcliffe, G.: Semantic derivation verification: Techniques and implementation. International Journal on Artificial Intelligence Tools **15**(06), 1053–1070 (2006)
43. Sutcliffe, G.: The TPTP problem library and associated infrastructure. J. Autom. Reason. **43**(4), 337–362 (2009)
44. The BWare ANR Project: (2012 – 2016), ANR-12-INSE-0010, `http://bware.lri.fr/`
45. The Coq Development Team: The Coq proof assistant (Sep 2022). `https://doi.org/10.5281/zenodo.7313584`
46. Thiré, F.: Sharing a library between proof assistants: Reaching out to the HOL family. In: Blanqui, F., Reis, G. (eds.) Proceedings of the 13th International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, LFMTP@FSCD 2018, Oxford, UK, 7th July 2018. EPTCS, vol. 274, pp. 57–71 (2018)