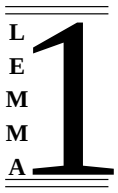


Lemma 1 Ltd.
27, Brook St.
Twyford
Berks
RG10 9NX

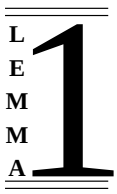
OpenTheory for ProofPower

Abstract

This document discusses the transfer of specifications and proofs between ProofPower and other members of the HOL family of interactive theorem provers via OpenTheory. It includes a literate script providing an experimental implementation of an OpenTheory reader and an OpenTheory writer for ProofPower.

Version: *Revision* : 1.17
Date: 10 August 2014
Reference: LEMMA1/QCZ/WRK084
Pages: 13

Prepared by: R.D. Arthan
Tel: +44 7947 030 682
E-Mail: rda@lemma-one.com



0.1 Contents

0.2 References

[1] Joe Hurd. The OpenTheory Standard Theory Library. In Mihaela Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi, editors, *Third International Symposium on NASA Formal Methods (NFM 2011)*, volume 6617 of *Lecture Notes in Computer Science*, pages 177–191. Springer, April 2011. See also `http://gilith.com/OpenTheory/`.

[2] Magnus O. Myreen, Michael J. C. Gordon, and Konrad Slind. Decompilation into logic - Improved. In Gianpiero Cabodi and Satnam Singh, editors, *FMCAD*, pages 78–81. IEEE, 2012.

[3] Colin O'Halloran. Automated verification of code automatically generated from Simulink®. *Automated Software Engg.*, 20(2):237–264, June 2013.

0.3 Changes History

Issues 1.1 (2014/02/14)–1.6 (2014/02/18) Initial drafts.

Issue 1.7 (2014/02/19) First issue for D-RisQ.

Issues 1.8 (2014/03/10) –1.15 (2014/03/25) Work in progress on full support for the OpenTheory base package.

Issue 1.16 (2014/03/27) First version that can read the full OpenTheory base package with no hacks.

Issue 1.17 (2014/04/01) Final draft for testing prior to first public issue.

2014/08/09 (a) Augmented old RCS version numbers in the changes history with dates. Dates will be used in place of version numbers in future.

2014/08/09 (b) First cut of upgrade for compatibility with OpenProofPower 3.1w2.

2014/08/10 Added tests and brought documentation up-to-date.

0.4 Changes Forecast

This implementation is essentially a proof of concept. However, it contains all the key elements needed to support interchange of specifications and proofs via OpenTheory. A production version will involve significant refactoring: specifically splitting it into separate ProofPower modules covering: *(i)* types and functions common to the reader and the writer; *(ii)* library support for the Gilith OpenTheory Repo; *(iii)* reader; *(iv)* writer.

1 INTRODUCTION

ProofPower is an interactive theorem prover in the HOL family, that *inter alia* provides proof support for the Z notation and is used as the proof engine for Z in D-RisQ's CLawZ toolset [3]. HOL4 is another system in the HOL family that is widely used in academia. In particular, HOL4 is the platform for the work of Myreen *et al.* on decompiling object code into logic [2]. D-RisQ are interested in combining the capabilities of the CLawZ tools and the technique of decompilation into logic.

OpenTheory [1] is an initiative to develop a concept of *theory engineering* for the development and management of large bodies of formal specifications and proofs, analogous to the concept of *software engineering* that arose in the 1960s for the development and management of large software systems. OpenTheory defines a file format that can be used to transfer specifications and proofs from one theorem proving system in the HOL family to another. This is supported by the `opentheory` tool which provides a number of useful services for grouping articles into packages and for managing a set of packages.

As part of a research programme funded by DSTL, D-RisQ have commissioned an experimental implementation of OpenTheory in ProofPower, in order to provide some level of interoperability between ProofPower and other implementations of HOL, specifically to allow specification to be transferred betweem HOL4 and ProofPower. This document is a ProofPower literate script that contains this implemementation and some discussion of its design.

The OpenTheory interface provided here comprises a *reader* for importing OpenTheory articles and a *writer* for exporting them. Both these interfaces are experimental and are likely to need further customisation. However, transfer of specifications and proofs in both direction between HOL4 and ProofPower has been demonstrated as has importing specifications from the Gilith OpenTheory Repo.

The remainder of this document is structured as follows:

Section 2 discusses OpenTheory and the present implementation of a reader and a writer for the OpenTheory article format in ProofPower.

Section 3 documents the interfaces provided in the style of the ProofPower reference manual.

2 DISCUSSION

2.1 Overview of OpenTheory

The goal of OpenTheory is to allow proofs and the specifications on which those proofs depend to be shared between different implementations of theorem provers for higher-order logic (HOL), specifically, HOL4, HOL Light and ProofPower. The OpenTheory initiative has also produced the Gilith OpenTheory Repo, a collection of theory packages containing a large body of specifications and proofs extending a base theory package containing definitions intended to be common to all HOL implementations.

The OpenTheory project has defined two file formats that are used to describe theory packages: the *article file format* is the basic mechanism for transferring specifications and proofs; the *theory file format* describes how one or more article files can be combined into a package. In this document we are only concerned with importing from and exporting to article files. (The `opentheory` tool can be used, if needed, to create theory files.)

An article file comprises a sequence of commands for a virtual machine. The virtual machine constructs HOL types, terms and theorems and outputs two sets: a set of assumptions, i.e., axioms, and a set of theorems, i.e., assertions that have been proved using the axioms. An implementation in a system like ProofPower can realise the virtual machine so that the output is added to its database of definitions, axioms and theorems. The OpenTheory reader for ProofPower does exactly that.

Initially, it was not clear whether the Gilith OpenTheory Repo would be relevant to the specific problem of transferring specifications from HOL4 to ProofPower. However, it turned out to be very useful both as a source of test material and in defining a common vocabulary that is used by the HOL4 implementation of an OpenTheory writer.

2.2 ProofPower OpenTheory Reader

The ProofPower OpenTheory reader is based on a generic implementation of the OpenTheory virtual machine with parameters defining the mapping of the vocabulary and definitions in the input to ProofPower vocabulary and extension mechanisms. This has then been instantiated to accept the vocabulary and definitions used by the Gilith OpenTheory Repo and the HOL4 OpenTheory writer. This is supported by a theory "open-theory-base" that provides definitions of the types and constants used in the OpenTheory base repo.

Once the mapping of vocabulary is settled importing proofs is relatively straightforward since one has all the capabilities of ProofPower available to implement the OpenTheory inference rules as derived rules. The base theory package in the Gilith OpenTheory Repo has been imported into ProofPower. To do this one needs to separate out the constituent packages that

prove theorems from those that provide definitions. A script that does this automatically is provided as a test in this script.

2.3 ProofPower OpenTheory Writer

The OpenTheory writers for HOL4 and HOL Light are implemented with a special version of the logical kernel that includes a proof tree in the representation of a theorem. ProofPower enables functions that monitor logical kernel operations to be registered. The ProofPower OpenTheory writer uses this facility to export a log of the kernel operations as an OpenTheory article. The process is parametrized by functions that map type and constant names to the desired external form.

Exporting proofs is a more intricate problem than importing them because OpenTheory does not provide high-level facilities for programming proofs. (This is intended, since it makes the OpenTheory kernel small, simple and dependable.) The writer is implemented as a library of functions that output the command sequences for tasks such as building a term or carrying out an inference step.

At version 5 OpenTheory only supported constant definitions of the form $c = t$ where c is the constant being defined, whereas ProofPower and HOL4 both support a more general mechanism whereby if an existence theorem $\vdash \exists x_1, \ldots x_k \bullet \phi(x_1, \ldots, x_k)$ has been proved, it can be used to specify new constants $c_1, \ldots, c_k$ with defining property $\vdash \phi(c_1, \ldots, c_k)$. We simulate the effect of this mechanism using the Hilbert choice function ϵ, defining $c_1 = \epsilon x_1 \bullet \exists x_2, \ldots x_k \bullet \phi(x_1, \ldots, x_k)$, $c_2 = \epsilon x_2 \bullet \exists x_3, \ldots x_k \bullet \phi(c_1, x_2, \ldots, x_k)$ and so on. The theorem $\vdash \phi(c_1, \ldots, c_k)$ can then be inferred from the existence theorem and these definitions. The resulting translated definition is less abstract than the original but this is inevitable.

A more general mechanism, *defineConstList* for constant definitions has been added to OpenTheory in draft version 6 of the format. This corresponds closely to ProofPower's *gen_new_spec*. Currently, pending implementation of version 6 in the `opentheory` tool, we continue to use Hilbert choice for *new_spec* but use *defineConstList* for *gen_new_spec*.

Type definitions in OpenTheory follow HOL Light in introducing constants for the abstraction and representation functions for the new type. In contrast, the new type definition mechanism in ProofPower and HOL4 returns a theorem that implies the abstraction and representation functions exist, but does not actually make the constant definitions. The translation of type definitions involves deducing the existence theorem from the defining properties of the new constants. The writer also has to invent names for the new constants, which will only be used in the proof of the existence theorem.

An issue with the chosen implementation approach is that theorems that have been proved when logging is not turned on will appear as assumptions in the article file if they are used in a proof that is exported. Typically such theorems will be lemmas used in derived inference

rules. One solution to this would be to record some or all of the inferences made when the system is built, so that the proofs of these lemmas can be recreated. This is a topic for future work.

3 THE STRUCTURE OpenTheory

The OpenTheory implementation is packaged in a structure (i.e., a Standard ML module) with the following signature. (As usual in ProofPower we document a structure by annotating its signature with narrative describing the intended implementation.)

SML
```
signature OpenTheory = sig
```

3.1 The Reader

3.1.1 Command Interface

SML
```
val open_theory_import :
      {files : string list, parent : string, theory : string} -> unit;
```

Description This function provides a simple interface to the OpenTheory reader to import one or more article files into a ProofPower theory. It expects the article to use the naming conventions and definitions of the Gilith OpenTheory Repo, but also supports the forms of type definition implemented in HOL4 and ProofPower. The article files are loaded into the named theory, which is newly created for the purpose (any existing theory of that name is deleted). The theory is created as a child of the indicated parent theory and the theory "open-theory-base" is made a parent if it is not an ancestor of the specified parent.

3.1.2 Programmatic Interface

For experimental purposes, the core functionality of the OpenTheory reader is presented as a programmable interface to the OpenTheory virtual machine. A name in an article file is a pair comprising a hierarchical name space and the name of a component within that namespace.

SML
```
type NAME = string list * string;
```

The types *TYPE_DEFN_INFO* and *ABS_REP_THMS* represent the inputs and outputs of the *defineTypeOp* command. The virtual machine is parametrized by a function that provides the semantics for this command. This function may, for example, create the outputs using an existing type definition rather than trying to define a new type.

SML
```
type TYPE_DEFN_INFO = {
        thm : THM,
        pars : string list,
        rep : string,
        abs : string,
        tyname : string
};
type ABS_REP_THMS = {
        abs_rep : THM,
        rep_abs : THM
};
```

The type *CONST_DEFN_INFO* represents the inputs of the *defineConst* command while *CONST_LIST_DEFN_INFO* represents the inputs of the *defineConstList* The virtual machine is parametrized by a function that maps these inputs to the desired designing theorem, e.g., by making the necessary definitions.

SML
```
type CONST_DEFN_INFO = {
        name : string,
        def_rhs : TERM
};
type CONST_LIST_DEFN_INFO = {
        thm : THM,
        names : (string * TERM) list
};
```

The import interface then comprises *(i)* a function *declare_axiom* to handle the *axiom* command, *(ii)* a function *define_type_op* to handle the *defineTypeOp* command, *(iii)* a function *define_const* to handle the *defineConst* command, *(iv)* a function *define_const_list* to handle the *defineConstList* command, *(v)* a function *const_name* to map OpenTheory names to constant names, *(vi)* a function *type_name* to map OpenTheory names to type names and *(vii)* an initial set of assumptions.

SML
```
type IMPORT_INTERFACE = {
        declare_axiom : SEQ -> THM,
        define_type_op : TYPE_DEFN_INFO -> ABS_REP_THMS,
        define_const : CONST_DEFN_INFO -> THM,
        define_const_list : CONST_LIST_DEFN_INFO -> THM,
        const_name : NAME -> string,
```

```
|       type_name : NAME -> string,
|       assumptions : THM list
|};
```

The *const_name* and *type_name* functions in the import interfaces may be defined using tables mapping OpenTheory names to the ProofPower names of types and constants.

SML
```
|type NAME_MAPS = {
|       type_map : (NAME * string) list,
|       const_map : (NAME * string) list
|};
```

The function *mk_import_interface* sets up an import interface from name map tables and a list of assumptions given as parameters. The functions *declare_axiom*, *define_type_op* and *define_const* it constructs map to the corresponding ProofPower operations or use the Gilith OpenTheory Repo definitions (proved as theorems using existing ProofPower definitions for the relevant operators).

SML
```
|val mk_import_interface : NAME_MAPS -> THM list -> IMPORT_INTERFACE;
```

The objects that the virtual machine deals with range over the type *OBJECT*:

SML
```
|datatype OBJECT =
|           ONum of INTEGER          (* A number *)
|       |   OName of NAME             (* A name *)
|       |   OList of OBJECT list      (* A list (or tuple) of objects *)
|       |   OTypeOp of string         (* A higher order logic type operator *)
|       |   OType of TYPE             (* A higher order logic type *)
|       |   OConst of NAME            (* A higher order logic constant *)
|       |   OVar of string * TYPE     (* A higher order logic term variable *)
|       |   OTerm of TERM             (* A higher order logic term *)
|       |   OThm of THM               (* A higher order logic theorem *);
```

The state of the virtual machine includes a dictionary mapping integers to objects:

SML
```
|type OBJECT_DICT = (int, OBJECT) SEARCH_TREE;
```

The state includes a cache giving fast access to the assumptions and theorems.

SML
```
type VM_STATE = {
        interface : IMPORT_INTERFACE,
        command_count : int,
        stack : OBJECT list,
        dictionary : OBJECT_DICT,
        assumptions : THM list,
        theorems : THM list,
        cache : THM NET};
```

SML
```
val vm_diagnostics : int ref;
```

vm_diagnostics can be set to give various levels of diagnostic output. It is checked after each command is executed resulting in output as follows:

< 0	All diagnostic output is suppressed.
0	A progress pragma command will output its operand.
1	A progress pragma command will output its operand and the number of commands executed (including this command).
2	A progress pragma command will cause its operand, the number of commands executed, the new top of stack and the old stack to be printed.
3	The command and the number of commands executed are printed. A progress pragma command does not output its operand.
4	The command, the number of commands executed and the new top of stack are printed if the command has inferred a theorem or fetched a theorem from the dictionary. A progress pragma command outputs its operand.
5	The command, the number of commands executed, the new top of stack and the old stack are printed if the command has inferred a theorem or fetched a theorem from the dictionary. A progress pragma command outputs its operand.
> 5	The command, the number of commands executed, the new top of stack and the old stack are printed unconditionally. A progress pragma command outputs its operand.

The virtual machine records its current state in the variable *diag_state*.

SML
```
val diag_vm_state : VM_STATE OPT ref;
```

The programmatic interface to the virtual machine is the following function. The second parameter represents the input stream.

SML
```
val open_theory_vm : IMPORT_INTERFACE ->
        ('s -> (string * 's) OPT) -> 's -> VM_STATE;
```

3.2 The Writer

3.2.1 Parameters

The writer is parametrized by functions that map type and constant names to OpenTheory names, by strings used as suffixes for the abstraction and representation functions for new types and by the target version of OpenTheory to be used (this should currently be 5 or 6).

SML
```
type EXPORT_INTERFACE = {
        type_name : string -> NAME,
        const_name : string -> NAME,
        abs_suffix : string,
        rep_suffix : string,
        version : int};
```

SML
```
val export_interface : EXPORT_INTERFACE ref;
```

3.2.2 Interface

SML
```
val set_open_theory_version : int -> int;
```

Description This function sets the version number of the article format to be used by the OpenTheory writer. It returns the value that was previously being used. (The default is 6.)

SML
```
val begin_open_theory_export : string -> unit;
val end_open_theory_export : unit -> unit;
```

Description These functions control the OpenTheory writer enabling definitions and proofs to be exported in the OpenTheory article format. *begin_open_theory_export file* reinitialises the writer state and initiates exporting into the specified article file. The article file is overwritten if it already exists.

end_open_theory_export closes the current article file and terminates logging.

See Also *proof_to_article, term_to_article*

SML

val ***proof_to_article*** : {*file*: *string*, *prf_fun*: *'a* -> *THM list*} -> *'a* -> *unit*;

Description *proof_to_article* generates an OpenTheory article from a function that proves a list of theorems. This provides a simple mechanism for transferring proofs from ProofPower to another HOL system or to the Gilith OpenTheory Repository.

See Also *term_to_article, begin_open_theory_export, end_open_theory_export*

SML

val ***term_to_article*** : {*tm*: *TERM*, *file*: *string*} -> *unit*;

Description This function generates an OpenTheory article from a term, t. The article proves the theorem $t = t$. This provides a simple mechanism for transferring terms from ProofPower to another HOL system.

See Also *proof_to_article, begin_open_theory_export, end_open_theory_export*

SML

end (* *of signature OpenTheory* *);

3.3 Errors

The OpenTheory interfaces define the following error messages.

Errors

122001	*invalid stack. Expected ?0*
122002	*unexpected empty stack*
122003	*invalid command ?0*
122004	*invalid name (final quotation symbol missing)*
122005	*invalid name (junk after final quotation symbol)*
122006	*invalid name (ends with a backslash)*
122007	*invalid variable or type variable name (not in global namespace)*
122008	*invalid list*
122009	*⌜?0⌝ is not of type ⌜:BOOL⌝*
122010	*attempt to access non−existent dictionary entry*
122011	*ill−formed substitution*
122012	*invalid command*
122013	*Cannot define ?0 = ?1 (?0 clashes with an existing constant, original exception: ?2)*
122014	*ill−formed list of pairs*
122015	*ill−formed constant name specifier*
122016	*Cannot make definition using ?0 with gen_new_spec (original exception: ?1)*
122101	*an OpenTheory export is not in progress*